Evolutionary Computing

Edited by: **Luciana Rocha**

BIBLIOTEX
Digital Library

Evolutionary Computing

Evolutionary Computing

Editor:

Luciana Rocha

BIBLIOTEX
Digital Library
www.bibliotex.com

Evolutionary Computing

Editor: Luciana Rocha

www.bibliotex.com

email: info@bibliotex.com

e-book Edition 2022

ISBN: 978-1-98467-775-4 (e-book)

In Collaboration with Intelliz Press. Originally Published in printed book format by Intelliz Press with ISBN 978-1-68251-835-9

TABLE OF CONTENTS

PREFACE

Evolutionary computing is particularly suited to the adaptation (learning) of neural and fuzzy systems. Evolutionary computing is a versatile problem solver inspired by natural evolution. It models the critical elements of biological evolution and investigates the space of solution through gene inheritance, mutation and selection of the most suitable candidate solutions. Evolutionary computing is a significant field of study for adaptation and optimization. The approach actually originated from the Darwin concept of natural selection, also known as the survival of the fittest. Evolutionary computing has seen a significant increase in both theoretical and industrial applications over the last decade. Its scope has grown beyond its original sense of "biological evolution" to a broad range of nature-inspired computational algorithms and techniques, covering evolutionary, neural, ecological, social and economic computing, etc., in a unified context. In the Darwinian model, information gained by an individual cannot be transmitted to its genome and consequently passed on to the next generation. The synthesis of learning and evolution, represented by evolving neural networks, is more adaptable to a changing world. The interaction of learning with evolution accelerates evolution, which can take the form of the Lamarckian evolution or be based on the Baldwin effect. The Lamarckian strategy enables the inheritance of inherited traits in the genetic code of an

individual's life so that the offspring will inherit its characteristics. Today, many research topics in evolutionary computing are not inherently "evolutionary".

The current book provides an overview of some recent advances in evolutionary computation. It concentrates on evolutionary computing, which is viewed as one of the most promising paradigms of computational intelligence. It covers a wide range of topics in optimization, learning and design using evolutionary approaches and techniques, and theoretical results in the computational time complexity of evolutionary algorithms. The dialects of evolutionary algorithms include genetic algorithms, evolutionary strategies, genetic programming, particle swarm optimization, ant colony optimization, artificial immune systems, estimation of distribution algorithms, differential evolution, and memetic algorithms. These evolutionary methods have proven their success on various hard and complex optimization problems. During this time, new metaheuristic optimization approaches, like evolutionary algorithms, genetic algorithms, swarm intelligence, etc., were being developed and new fields of usage in artificial intelligence, machine learning, combinatorial and numerical optimization, etc., were being explored. Some issues related to future development of evolutionary computation are also discussed. Presenting some new theoretical as well as practical aspects of evolutionary computation, this book will be of great value to undergraduates and graduate students in computer science.

CHAPTER 1

INTRODUCTION TO EVOLUTIONARY COMPUTATION

INTRODUCTION

In computer science, evolutionary computation is a family of algorithms for global optimization inspired by biological evolution, and the subfield of artificial intelligence and soft computing studying these algorithms. In technical terms, they are a family of population-based trial and error problem solvers with a met heuristic or stochastic optimization character.

In evolutionary computation, an initial set of candidate solutions is generated and iteratively updated. Each new generation is produced by stochastically removing less desired solutions, and introducing small random changes. In biological terminology, a population of solutions is subjected to natural selection (or artificial selection) and mutation. As a result, the population will gradually evolve to increase in fitness, in this case the chosen fitness function of the algorithm.

Evolutionary computation techniques can produce highly optimized solutions in a wide range of problem settings, making them popular in computer science. Many variants and extensions exist, suited to more specific families of problems and data structures. Evolutionary computation is also sometimes used in evolutionary biology as an in silico experimental procedure to study common aspects of general evolutionary processes.

1.1 OVERVIEW OF EVOLUTIONARY COMPUTATION

Surprisingly enough, the idea to apply Darwinian principles to automated problem solving originates from the fifties, long before the breakthrough of computers. During the sixties three different implementations of this idea have been developed at three different places. In the USA Fogel introduced evolutionary programming, while Holland called his method a genetic algorithm. In Germany Rechenberg and Schwefel invented evolution strategies. For about 15 years these areas developed separately; it is since the early nineties that they are envisioned as different representatives ("dialects") of one technology, called evolutionary computing. It was also in the early nineties that a fourth stream following the general ideas has emerged – genetic programming. The contemporary terminology denotes the whole field by evolutionary computing and considers evolutionary programming, evolution strategies, genetic algorithms, and genetic programming as sub-areas.

1.1.1 Evolutionary Computation

Evolutionary computation (EC) is a series of stochastic optimization algorithms that are inspired originally by natural selection and "survival of the fittest" and further developed by ant colony optimization, artificial immune systems, partial swarm intelligence, and others. From the algorithmic taxonomy viewpoint, EC takes the probability theory as its philosophy and

methodology. The fundamentals of its search mechanism are established in the basics of probability theory. The system behavior of an EC algorithm can therefore be presented as a probability transition matrix, and its dynamic optimization process can be described as a Markov chain. However, other theoretical analysis methods from deterministic theory, such as fixed point theory, are also introduced into the EC in order to study its fundamental aspects, such as efficiency, effectiveness, and convergence. Drawing inspiration from chaos theory and its ergodicity, a chaotic evolution algorithm has been recently proposed and studied. This is very different from the conventional deterministic and stochastic optimization algorithms. Chaotic evolution can be considered as an implementation of the chaotic optimization algorithm, whose theoretical fundamental is supported by chaotic philosophy and methodology.

There are two components in EC from the viewpoint of the algorithm design framework. One is an iterative process, and the other is one or several evolutionary operations that are implemented by a variety of methods. Algorithm selection and parameter settings are two critical issues, when we apply EC to an optimization problem. The objective of the former issue is to answer the question as to which is the best EC algorithm to solve a concrete problem. The latter one seeks to obtain the best parameter setting of an EC algorithm to obtain a better optimization performance.

What is an evolutionary algorithm?

The common underlying idea behind all these techniques is the same: given a population of individuals, the environmental pressure causes natural selection (survival of the fittest) and hereby the fitness of the population is growing. It is easy to see such a process as optimization. Given an objective function to be maximized we can randomly create a set of candidate solutions and use the objective function as an abstract fitness measure (the higher the better). Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination and/or mutation. Recombination is applied to two selected candidates,

the so-called parents, and results one or two new candidates, the children. Mutation is applied to one candidate and results in one new candidate. Applying recombination and mutation leads to a set of new candidates, the offspring. Based on their fitness these offspring compete with the old candidates for a place in the next generation. This process can be iterated until a solution is found or a previously set time limit is reached. Let us note that many components of such an evolutionary process are stochastic. According to Darwin, the emergence of new species, adapted to their environment, is a consequence of the interaction between the survival of the fittest mechanism and undirected variations. Variation operators must be stochastic, the choice on which pieces of information will be exchanged during recombination, as well as the changes in a candidate solution during mutation, are random.

On the other hand, selection operators can be either deterministic, or stochastic. In the latter case fitter individuals have a higher chance to be selected than less fit ones, but typically even the weak individuals have a chance to become a parent or to survive. The general scheme of an evolutionary algorithm can be given as follows.

```
Initialize population with random
  individuals (candidate solutions)
Evaluate (compute fitness of) all
individuals
WHILE not stop DO
  Select genitors from parent population
  Create offspring using
    variation operators on genitors
  Evaluate newborn offspring
  Replace some parents by some offspring
OD
```

Let us note that this scheme falls in the category of generate-and-test, also known as trialand-error, algorithms. The fitness function represents a heuristic estimation of solution quality and the search process is driven by the variation operators (recombination and mutation creating new candidate solutions) and the selection operators. Evolutionary algorithms (EA) are distinguished within in the family of generate-and-test methods by being population

based, i.e. process a whole set of candidate solutions and by the use of recombination to mix information of two candidate solutions. The aforementioned "dialects" of evolutionary computing follow the above general outlines and differ only in technical details.

1.1.2 Critical issues

There are some issues that one should keep in mind when designing and running an evolutionary algorithm. These considerations concern all of the "dialects", and will be discussed here in general, without a specific type of evolutionary algorithm in mind. One crucial issue when running an EA is to try to preserve the genetic diversity of the population as long as possible. Opposite too many other optimization methods, EAs use a whole population of individuals – and this is one of the reasons for their power. However, if that populations starts to concentrate in a very narrow region of the search space, all advantages of handling many different individuals vanish, while the burden of computing their fitnesses remains. This phenomenon is known as premature convergence. There are two main directions to prevent this: a priori ensuring creation of new material, for instance by using a high level of mutation or a posteriori manipulating the finesses of all individuals to create a bias against being similar, or close to, existing candidates. A well-known technique is the so-called niching mechanism.

Exploration and exploitation are two terms often used in EC. Although crisp definitions are lacking there has been a lot of discussion about them. The dilemma within an optimization procedure is whether to search around the best-so-far solutions (as their neighborhood hopefully contains even better points) or explore some totally different regions of the search space (as the bestso-far solutions might only be local optima). An EA must be set up in such a way that it solves this dilemma without a priori knowledge of the kind of landscape it will have to explore. The exploitation phase can sometimes be "delegated" to some local optimization procedure, whether called as a mutation operator, or systematically applied to all newborn individuals, moving

them to the nearest local optimum. In the latter case, the resulting hybrid algorithm is called a memetic algorithm

In general, there are two driving forces behind an EA: selection and variation. The first one represents a push toward quality and is reducing the genetic diversity of the population. The second one, implemented by recombination and mutation operators, represents a push toward novelty and is increasing genetic diversity. To have an EA work properly, an appropriate balance between these two forces has to be maintained. At the moment, however, there is not much theory supporting practical EA design.

1.1.3 Components of Evolutionary Algorithms

Representation

Solving a given problem with an EA starts with specifying a representation of the candidate solutions. Such candidate solutions are seen as phenotypes that can have very complex structures. Applying variation operators directly to these structures might not be possible, or easy. Therefore these phenotypes are represented by corresponding genotypes. The standard EC machinery consists of many off-the-shelf variation operators acting on a specific genotype space, for instance bit-strings, real-valued vectors, permutations of integers, or trees. Designing an EA thus often amounts to choosing one of the standard representations with the corresponding variation operators in mind. However, one strength of EAs is their ability to tackle any search space provided that initialization and variation operators are available. Choosing a standard option is, therefore, not necessary.

Fitness or evaluation function

Fitness-based selection is the force that represents the drive toward quality improvements in an EA. Designing the fitness function (or evaluation function) is therefore crucial. The first important

feature about fitness computation is that it represents 99% of the total computational cost of evolution in most real-world problems. Second, the fitness function very often is the only information about the problem in the algorithm: Any available and usable knowledge about the problem domain should be used.

Representation dependent

Initialization

The initial population is usually created by some random sampling of the search space, generally performed as uniformly as possible. However, in some cases, uniform sampling might not be well-defined, e.g. on parse-tree spaces, or on unbounded intervals for floating-point numbers. A common practice also is to inoculate some known good solutions into the initial population. But beware that no bias is better than a wrong bias!

Crossover

Crossover operators take two (or more) parents and generate offspring by exchange of information between the parents. The underlying idea to explain crossover performance is that the good fitness of the parents is somehow due to precise parts of their genetic material (termed building blocks), and the recombining those building blocks will result in an increase in fitness. Nevertheless, there are numerous other ways to perform crossover. For instance, crossing over two vectors of floating-points values can be done by linear combination (with uniformly drawn weights) of the parent's values. The idea is that information pertaining to the problem at hand should be somehow exchanged. Note that the effect of crossover varies from exploration when the population is highly diversified to exploitation when it starts to collapse into a small region of the search space.

Mutation

Mutation operators are stochastic transformations of an individual. The usual compromise between exploration and exploitation must be maintained: large mutations are necessary from theoretical reasons (it ensures the ergodicity of the underlying stochastic process), that translate practically (it is the only way to reintroduce genetic diversity in the end of evolution); but of course too much too large mutation transform the algorithm into a random walk – so most mutations should generate offspring close to their parents. There is no standard general mutation, but general trends are to modify the value of a component of the genotype with a small probability (e.g. flip one bit of a bitstring, or, in case of real-valued components, add zero-mean Gaussian noise with carefully tuned standard deviation).

The historical debate

There has long been a strong debate about the usefulness of crossover. The GA community considers crossover to be the essential variation operator, while mutation is only a background necessity. The general agreement nowadays is that the answer is problem-dependent: If there exists a "semantically meaningful" crossover for the problem at hand, it is probably a good idea to use it. But otherwise mutation alone might be sufficient to find good solutions – and the resulting algorithm can still be called an Evolutionary Algorithm.

1.1.4 Representation-independent

Artificial Darwinism

Darwin's theory states that the fittest individuals reproduce and survive. The evolution engine, i.e. the two steps of selection (of some parents to become genitors) and replacement (of some parents by newborn offspring) are the artificial implementations

of these two selective processes. They differ in an essential way: during selection step, the same parent can be selected many times; during replacement step, each individual (among parents and offspring) either is selected, or disappears forever. Proportional selection (aka roulette-wheel) has long been the most popular selection operator: each parent has a probability to be selected that is proportional to its fitness. However, the difficulty is to scale the fitness to tune the selection pressure: even the greatest care will not prevent some super-individual to take over the population in a very short time. Hence the most widely used today is tournament selection: to select one individual, T individuals are uniformly chosen, and the best of these T is returned. Of course, both roulette-wheel and tournament repeatedly act on the same current population, to allow for multiple selection of the very best individuals.

There are two broad categories of replacement methods: either the parents and the offspring "fight" for survival, or only some offspring are allowed to survive. Denoting by μ (resp. λ) the number of parents (resp. offspring) as in ES history the former strategy is called $(\mu + \lambda)$ and the latter (μ, λ). When $\mu = \lambda$, the comma strategy is also known as generational replacement: all offspring simply replace all parents. When $\lambda = 1$, the (plus!) strategy is then termed steady-state and amounts to choosing one parent to be replaced.

An important point about the evolution engine is the monotonicity of the best fitness along evolution: for instance, ES plus strategies are elitist, i.e. ensure that the best fitness can only increase from one generation to another, while the comma strategies are not elitist – though elitism can be a posteriori added by retaining the best parent when a decrease of fitness is foreseen.

Termination criterion

There has been very few theoretical studies about when to stop an Evolutionary Algorithm. The usual stopping criterion is a fixed amount of computing time (or, almost equivalently, of fitness

computations). A slightly more subtle criterion is to stop when a user-defined amount of time has passed without improvement of the best fitness in the population.

Setting the parameters

EAs typically have a large number of parameters (e.g. population size, frequency of recombination, mutation step-size, selective pressure, . . .). The main problem in this respect is that even the individual effect of one parameter is often unpredictable, let alone the combined influence of all parameters. Most authors rely on intensive trials (dozens of independent runs for each possible parameter setting) to calibrate their algorithms – an option that is clearly very time consuming. Another possibility is to use longexisting statistical techniques like ANOVA. A specific evolutionary trend is to let the EA calibrate itself to a given problem, while solving that problem

Result analysis

As with any randomized algorithm, the results of a single run of an EA are meaningless. A typical experimental analysis will run say over more than 15 independent runs (everything equal except the initial population), and present averages, standard deviations, and T-test in case of comparative experiments. However, one should distinguish design problems, where the goal is to find at least one very good solution once, from day-to-day optimization (e.g. control, scheduling,. . .), where the goal is to consistently find a good solution for different inputs. In the design context, a high standard deviation is desirable provided the average result is not too bad. In the optimization context, a good average and a small standard deviation are mandatory.

1.2 THEORY AND APPLICATIONS OF EVOLUTIONARY COMPUTATION

Evolutionary computation is a powerful problem solver inspired from natural evolution. It models the essential elements of biological evolution and explores the solution space by gene inheritance, mutation, and selection of the fittest candidate solutions. The dialects of evolutionary algorithms include genetic algorithms, evolutionary strategies, genetic programming, particle swarm optimization, ant colony optimization, artificial immune systems, estimation of distribution algorithms, differential evolution, and memetic algorithms. These evolutionary methods have proven their success on various hard and complex optimization problems.

1.2.1 Discrete Dynamics in Evolutionary Computation and Its Applications

Evolutionary computation (EC) is considered to be a natural and artificial system with discrete dynamics. EC has been successfully applied to various real-world problems for optimization purposes. The aim of this special issue is to publish original and high-quality articles related to discrete dynamics in EC and its applications.

This special issue was opened in November of 2015 and closed in February of 2016. There were a total of 29 submissions. All of them were peer-reviewed according to the high standards of this journal and only 5 of them were accepted for publication, which gave important developments in discrete dynamics in EC and its applications. The guest editors of this special issue hope that the presented results could outline new ideas for further studies.

In EC, selection or mating is one of the most important operations. "A New Adaptive Hungarian Mating Scheme in Genetic Algorithms," C. Jung et al. suggested an adaptive mating scheme from Hungarian mating schemes, which consist of maximizing the sum of mating distances, minimizing the sum, and random matching. They presented an adaptive algorithm to elect one of

these Hungarian mating schemes. Each mated pair of individuals voted for the next generation mating scheme. The distance between parents and the distance between parent and offspring were considered during voting. Two well-known combinatorial optimization problems, the traveling salesman problem and the graph bisection problem, which are NP-hard, were considered to show the effectiveness of their adaptive method. Since various factors affect the fluctuation of network traffic, accurate prediction of network traffic is considered as a challenging task of the time series prediction process. "A Network Traffic Prediction Model Based on Quantum-Behaved Particle Swarm Optimization Algorithm and Fuzzy Wavelet Neural Network," K. Zhang et al. proposed a novel prediction method of network traffic based on quantum-behaved particle swarm optimization (QPSO) algorithm and fuzzy wavelet neural network (FWNN). The authors introduced QPSO and presented the structure and operation algorithms of FWNN. The parameters of FWNN were optimized by a QPSO algorithm. This optimized QPSO-FWNN was applied to the prediction of network traffic successfully when compared to different prediction models such as BP neural network, RBF neural network, fuzzy neural network, and FWNN-GA neural network.

The exponential growth in data traffic due to the modernization of smart devices has resulted in the need for a high-capacity wireless network in the future. To successfully deploy 5G networks, we should be able to handle the growth in the data traffic. The increasing amount of traffic volume puts excessive stress on the important factors of the resource allocation methods such as scalability and throughput. "A Genetic Algorithm with Location Intelligence Method for Energy Optimization in 5G Wireless Networks," R. Sachan et al. defined network planning as an optimization problem with the decision variables such as transmission power and transmitter location in 5G networks, leading to interesting implementation using some heuristic approaches such as differential evolution and real-coded genetic algorithm (RCGA). The authors modified an RCGA-based method to find the optimal configuration of transmitters by not only

offering optimal coverage of underutilized transmitters, but also optimizing the amounts of power consumption.

1.2.2 Using Evolutionary Computation on GPS Position Correction

Global positioning System, GPS, has been successfully applied in various areas such as navigation, meteorology, military tasks, mapping, tour design, path tracking tools, and more. Recently many mobile devices have been equipped with embedded GPS such as tablet PCs and smart phones. They provide maps to help users not to lose their way or search the shortest route to their destination.

Many techniques are proposed to improve GPS position accuracy. A commonly used technique is to use relative positioning. Relative positioning methods, including static, rapid static, pseudokinematic, kinematic, and real-time kinematic, have proved their ability of improving GPS accuracy. In, Berber et al. claimed that pseudokinematic technique produces closest results, which could significantly reduce the error to 2 centimeters.

Differential correction is an effective method to improve GPS positional accuracy. A GPS receiver with such technique is called dGPS. A typical differential correction requires a reference stationary receiver at a known location. Figure 1 shows a typical scenario of the dGPS environment. The exact location information of reference stationary receiver is known. It receives GPS signals and calculates its position. Under the assumption that close GPS receivers suffer similar noises and after evaluating the difference between the exact known position information and the calculated position information, the reference stationary receiver communicates with roving GPS receivers to correct their position information. dGPS can be used to eliminate affections of ionospheric and tropospheric delay, ephemeris error, and satellite clock error. However, when the error is due to multipath error, or poor satellite measurement geometry, the improvement effectiveness of dGPS technique is relatively low.

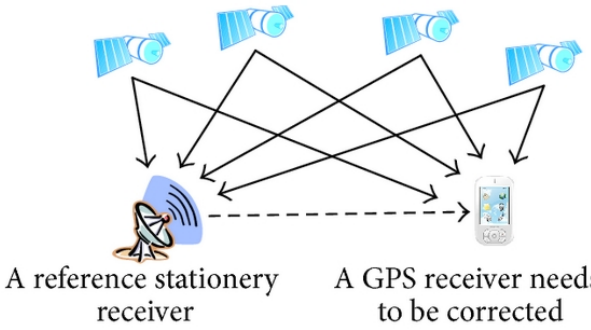


Figure 1: An illustration of dGPS scenario. The precise location of the reference stationary receiver is known.

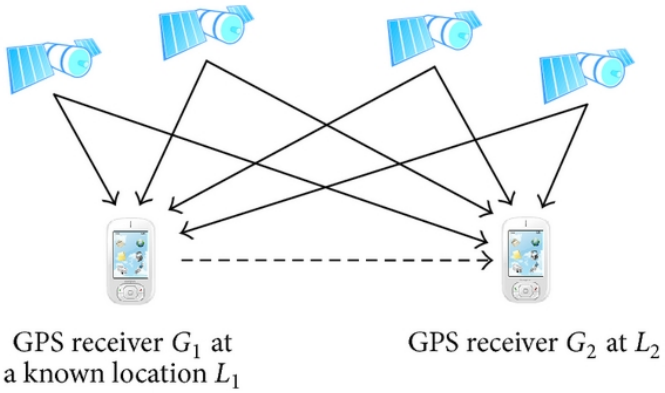


Figure 2: A consumer-grade GPS receiver at a location which has exact known location.

This technique is based on differential correction and genetic programming (GP). GP will be used to generate a correction function from NMEA information derived from the GPS receiver at the known location and the GPS receiver which needs to be corrected. The receiver which requires to be corrected will apply the function to obtain its corrected location information.

1.2.3 Layered Architecture Genetic Programming

Genetic programming is a research area of evolutionary computation. It has been proved that GP is capable of finding a solution efficiently. GP, like other techniques in evolutionary computation, generates possible solutions—in this case, correction functions—randomly for the given problem under given constrains. The fitness value which of an individual is used to measure the degree of the individual fitting with the given problem is determined by a predefined fitness function. The set with fixed size of individuals is named a *population*. In order to produce new solutions, genetic operators such as crossover and mutation are applied on selected individuals, called *parents*, to create offspring and mutant. Comparing the fitness degree of those offspring and mutant with parents, which have higher fitness value, will be kept as survived individuals. All survived individuals will replace the original population. A *generation* is finished once the original population is fully replaced. After a number of generations, evolutionary process completes and the individual with highest fitness is regarded as the result.

We use the improved version of genetic programming called layered architecture genetic programming, LAGEP. LAGEP is only usable with functional expression individuals. It utilizes the layer architecture to arrange populations. Populations in the same layer evolve independently. Once every population finishes evolutionary progress, the best individual of each population evaluates with its training instances, T, to generate a series of numerical results. The number of results is equal to $|T|$. Combining those values, a new training set T' having $|T|$ instances could be produced. Supporting that the number of populations in the layer is n, T' will be an n-dimensional training set. The final layer of LAGEP contains one population only. The individual produced by this population is the evolutionary result. The flowchart of LAGEP is shown in Figure 3.

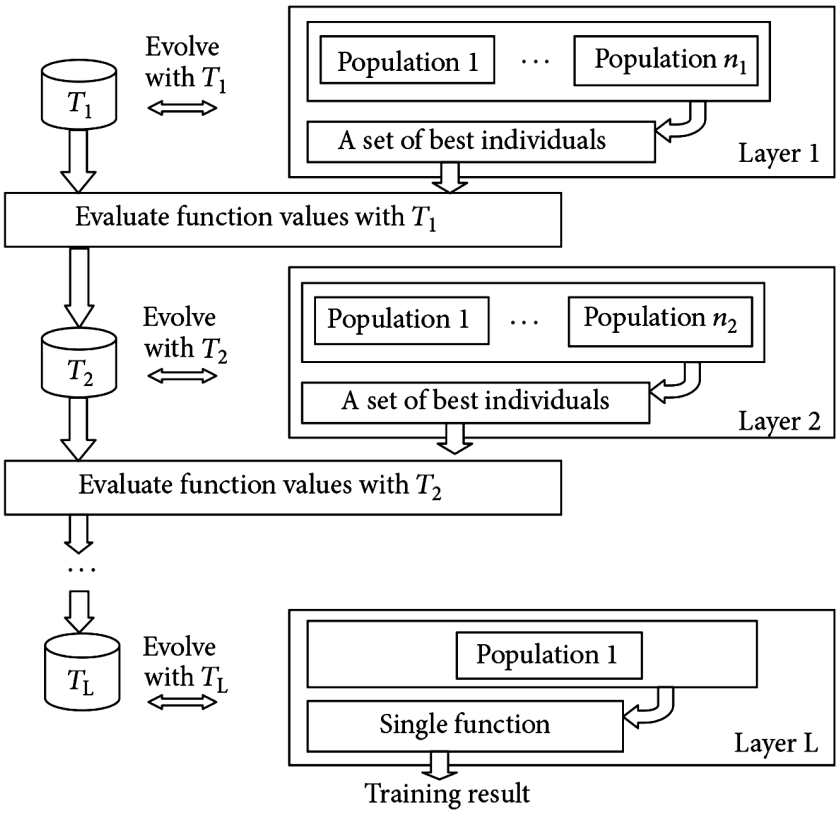


Figure 3: The flowchart of LAGEP.

Training instances are constructed by raw information obtained from two GPS receivers and the known location. GPS receivers are capable of transferring different types of NMEA interpreted sentences. In this work, we used GPGGA to represent position information, as shown in Table 1. The third, fifth, tenth, and twelfth field are symbols that can be harmlessly eliminated. The value of sixth field indicates GPS quality which is fixed. The thirteenth and fourteenth are usable when dGPS is available. The fifteenth is the checksum used to identify correctness of received data. In conclusion, 8 out of 15 fields can be removed. Two GPS receivers construct a 13-feature training instance after eliminating a redundant UTC time feature since those GPS receivers would have identical UTC time. Those features with longitude and latitude of the known location form a 15-feature training instance,

as shown in Table 2. The target value is either known latitude or known longitude to which we intent to correct GPS receiver as close as possible.

Table 1: Fifteen fields of GPGGA sentence.

Number	Meaning
1	UTC of position
2	Latitude
3*	N or S
4	Longitude
5*	E or W
6*	GPS quality indicator 1: invalid 2: GPS fix 3: dGPS fix
7	Number of satellites in use
8	Horizontal dilution of position
9	Antenna altitude above/below mean sea level (geoid)
10*	Meters (antenna height unit)
11	Geoidal separation
12*	Meters (units of geoidal separation)
13*	Age in seconds since last update from differential reference station
14*	Differential reference station ID
15*	Checksum
Removed features.	

Table 2: Features of a training instance.

Number	Meaning
Target value	Known latitude/known longitude
1–7	$GPGGA from GPS receiver 1 as shown in Table 1
8–13	$GPGGA from GPS receiver 2 as shown in Table 1 without the UTC of position

14	Known latitude (where GPS receiver 1 is located)
15	Known longitude (where GPS receiver 1 is located)

1.2.4 Algorithmic Mechanism Design of Evolutionary Computation

We consider algorithmic design, enhancement, and improvement of evolutionary computation as a mechanism design problem. All individuals or several groups of individuals can be considered as self-interested agents. The individuals in evolutionary computation can manipulate parameter settings and operations by satisfying their own preferences, which are defined by an evolutionary computation algorithm designer, rather than by following a fixed algorithm rule. Evolutionary computation algorithm designers or self-adaptive methods should construct proper rules and mechanisms for all agents (individuals) to conduct their evolution behaviour correctly in order to definitely achieve the desired and preset objective(s). As a case study, we propose a formal framework on parameter setting, strategy selection, and algorithmic design of evolutionary computation by considering the Nash strategy equilibrium of a mechanism design in the search process. The evaluation results present the efficiency of the framework. This primary principle can be implemented in any evolutionary computation algorithm that needs to consider strategy selection issues in its optimization process. The final objective of our work is to solve evolutionary computation design as an algorithmic mechanism design problem and establish its fundamental aspect by taking this perspective.

Game theory is the methodology used to research strategic interaction among several self-interested agents. Some important concepts, such as *type*, *strategy*, and *utility*, are useful to an understanding of the theoretical framework of game theory. Agent type indicates the preferences of the agent over different outcomes in a game. A strategy is a plan or a rule, which defines the actions that an agent will select in a game. The utility of an agent determines different allocations and payments under its and

other agents' types and strategy profiles; for example, an agent rationality in game theory is to implement the expected utility to be maximum. An agent will select a strategy that maximizes its expected utility, given its preferences with regard to outcomes, beliefs about the strategies of other agents, and structure of the game.

Nash equilibrium (NE) is one of the solution concepts that game theory provides to compute outcomes of a game from self-interested agents under certain assumed information that is available to each agent, such as agents types and strategies. It states that every agent in a game should select a maximum utility strategy taking account of other agents' strategies so as to achieve equilibrium. The fundamental aspect of game theory lies in the Nash solution concept which, however, requests stronger assumptions on agents' information (type, strategy, utility, etc.). There are some related solution concepts, such as dominant strategy and Bayesian-Nash strategy in game theory.

The history of mechanism design can be traced back to the 1920s to the 1930s, when there was an economic controversy concerning the socialist economic system. Liberal economists, such as von Mises and von Hayek, believed that socialist economics cannot obtain effective information to make their economic system operate efficiently. However, other economists, such as Lange, thought socialist economics can solve the problem of requesting more information about economic operation so as to promote efficient resource allocation. The contention of this controversy focuses on the information issue, which is also a core problem of mechanism design. Hurwicz established the fundamentals of mechanism design theory from the information viewpoint, which proposed a common framework to compare the issue of efficiency among different economic systems.

Mechanism design theory can be considered as a comprehensive utilization of game theory and social choice theory, which is referred to as principal agent theory and implementation theory as well. Its primary philosophy is to design a series of rules to implement the trust between principal and agent and to ensure the

mechanism runs well under an asymmetric information condition. The fundamental issues of mechanism design refer to (1) whether there is a set of rules in a game and (2) how to implement these rules. The objective of mechanism design is to achieve a preset objective that a game establishes, when all agents act for their own benefit with their private information.

When we make reference to individuals in EC, they are neither rational nor self-interested participants in an EC algorithm. The individuals just follow the fixed rules of the EC algorithm. The design principle of an EC algorithm should be more reasonable based on this assumption, rather than merely simulating natural phenomenon in an iterative process of evolution. There are several issues that need to be discussed in this context: (1) what kind of information should be involved in mechanism design of EC algorithm; (2) how to distinguish principal(s) and agent(s) in a game of EC algorithm design; (3) which solution concept should be implemented in this game; (4) how to define the expected utility function for an established mechanism or model; (5) whether the established mechanism or model is the optimal one; and (6) what are the characteristics and properties of designed game induced by EC. After modelling EC as a game, we can introduce theoretical principles of game theory and microeconomic theory into the fundamentals of EC for designing, enhancing, and improving more efficient and effective EC algorithm. This is the primary motivation and original contribution of this work.

1.2.5 Evolutionary Computation Meets Game Theory and Mechanism Design

Game theory attempts to determine the outcome of a game with a set of given strategies from self-interested agents, and mechanism design seeks to design the strategies of agents to obtain the desired outcome in a game. The research objectives of game theory and mechanism design are to find outcomes corresponding to strategies and to design strategies under a desired outcome, respectively. However, EC tries to find the optimal solution(s) of

an optimization problem. These three disciplines, game theory, mechanism design, and EC, are quite different with regard to research philosophies, approaches, or objectives.

Game theory and mechanism design have been introduced into some fields, such as distributed artificial intelligence, resource allocation, and scheduling. It is easy to establish concrete models of agent and utility function by using game theory in corresponding applications. Mechanism design was also reported in computation related topics and algorithmic design problems, which presents complexity bounds and worst case approximation. These studies focus definitely on the same viewpoint, that is, a mechanism design problem of deterministic optimization. These works do not relate to EC and do not pursue our proposal, that is, an algorithmic mechanism design of the EC, which is a stochastic optimization method rather than deterministic optimization one.

From the related literatures and to the best of our knowledge, EC can act as an optimization tool to find the best response, equilibrium of strategy in game theory and mechanism design. Some EC algorithms, such as genetic algorithm, genetic programming, and coevolution, are applied to game theory problems to obtain the best strategy or parameters. Scant literature reports having applied the philosophy and methodology of game theory or mechanism design to the fundamental aspects of EC.

1.2.6 Algorithmic Mechanism Design of Evolutionary Computation: A Strategy Equilibrium Implementation Problem

Motivation of the Proposal

Conventional EC algorithm as a search method is applied to an optimization problem with a set of fixed algorithm parameters and operations. Although the inspiration of EC seeks to find adaptive mechanisms in its search scheme, the fixed parameter setting restricts its optimization capability. From the system theory

viewpoint, the whole EC algorithm system can be considered as a control system and its parameters decide the system behaviour. If we aim to obtain the best optimization performance, the parameters of EC algorithm should be controllable, and the relationship between parameter settings and optimization performance should be observable. However, because the EC algorithm belongs to stochastic method, such deterministic methods (e.g., automatic control method) have not been applied to the EC area in order to study on its theoretical fundamentals.

There are primary three research directions for improving optimization performance of an EC algorithm. The first is obtaining information from a fitness landscape, such as fitness landscape approximation, and using the information to conduct a special operation or to develop new search schemes for tuning the parameter of an EC algorithm. The second is developing new mechanisms in extant EC algorithm to enhance its performance or to implement the parameter adaptive mechanism. Individuals in conventional EC algorithms or in some population-based optimization algorithm are common elements, which present the search space and structure aspect of an optimized problem. Individuals search for the optimum/optima with the information shared between one another, so they are influenced each by the other from one generation to the next. They operate under the same evolutionary operations with fixed operation rates from parameter settings in a certain EC algorithm. This work scheme restricts the EC algorithm search capability.

If we consider individuals in EC algorithm as agents, the EC algorithm therefore can be considered as a game, whose outcome is optimal solution(s) or some other metric(s). Furthermore, these agents play the game (i.e., EC algorithm) with self-interested preference and conduct optimization strategies with their own preferences. An EC algorithm designer can play this game by using noncooperative or cooperative game concepts. The objective of the EC algorithm design is to find optimum/optima by designing a proper strategy for these individuals (i.e., agents). This description can be abstracted as a mechanism design problem. That is, design,

parameter setting, and operation selection of the EC algorithm can be decided by the individuals, and the desired outcome is to find the final optimum/optima. EC algorithm can be modelled as a game, that is, an agent system, so the corresponding theoretical fundamentals of agent system, game theory, or mechanism design can be brought to bear on the study of the fundamental theoretical aspects of EC algorithm.

1.2.7 Strategy Equilibrium Implementation in Evolutionary Computation Algorithm

We propose that the design of an EC algorithm can be considered as a mechanism design problem. Based on these concepts, we establish a formal EC algorithm framework by using the equilibrium concept and solve this mechanism design problem by finding the Nash strategy equilibrium in EC algorithm.

Agent and Its Type

An agent is an abstract concept in game theory. It refers to a participant in a game who will make strategic decisions based on its type. Individuals in EC are definitely considered as agents under the basic philosophy of our proposal. In game theory, an agent is treated as being self-interested. However, in EC, we consider it as a more rational one that allows itself to select certain operations, even though its utility will become low. For example, the simulated annealing mechanism in EC is such a case, if the EC algorithm allows individuals to be replaced by their offspring with worse fitness. The agent in a game can determine its own behaviour, so the individuals of EC should follow this rule by encoding operation types and their rates in themselves. In the EC, the type of an agent can be considered as information, such as fitness and fitness landscape, or some metrics of the evolution.

Strategy Equilibrium

The agent in EC algorithm is the individual, which is a participant in a game of EC. Each individual can decide their own strategy, that is, operation and operation rate, by their utilities and types, which can be measured as fitness improvement information or other algorithm evaluation metrics. All the EC algorithm implementations can be abstracted as a mechanism design problem, whose equilibrium concept is a solution of the problem. The objective of mechanism design is to implement some strategy equilibrium concepts in a game; however, it is an optimization problem, in which equilibrium implementation can lead to the best performance and fast convergence of EC algorithms. We initially discuss, design, and evaluate our proposed framework by implementing Nash equilibrium, but it is not limited to within Nash equilibrium.

A Case Study: Nash Strategy Equilibrium-Based Differential Evolution Algorithm

After we introduce the optimization process of EC as an algorithmic mechanism design problem, there are a variety of ways to implement EC algorithms by designing concrete implementations of a game. There are two design issues that should be concentrated on especially. One is the definition of strategy; the other is equilibrium calculation. In Nash strategy equilibrium-based DE, operations and their parameters are coded with each individual, so that the individual can select their own operation and rate. Mutation method, crossover rate, and scale factor rate are three primary parameter settings in DE. For simplifying the design objectives, we design the algorithm by splitting individuals into two groups (Group A and Group B) with equal population size and strategy sets within the following criteria:

1.3 ANALYSES AND DISCUSSIONS

The objective of this study is not only to find a method for designing, enhancing, and accelerating EC from the viewpoint of an algorithmic mechanism design problem, but also to establish EC fundamental aspects by borrowing from game theory and mechanism design.

There are three parallel ways to research and consider our world from the philosophies of determinism, probability, and chaos. In the optimization field, there are also three categories of optimization method from the corresponding philosophy and methodology, that is, deterministic, stochastic, and chaotic optimization methods. EC belongs to the stochastic one, and its fundamental aspect should be described from the probability viewpoint. This restricts the fundamental development of EC and explanation capability of its algorithms. This study tries to use fundamentals from game theory and mechanism design (deterministic theory) to explain EC (stochastic algorithm) and establish its fundamental contents.

1.3.1 Prediction of Drifter Trajectory Using Evolutionary Computation

The technology for predicting particle trajectories in the ocean can be used in a variety of ways. For example, it can provide a method to track objects in the ocean during a distress situation or an accident through the last observed time and location data, as well as predicting the path of icebergs floating at the sea. It also presents the possibility of tracing pollutants in the event of accidents such as the 2010 Deepwater Horizon oil spill in the Gulf of Mexico; as a result, numerous studies have been conducted on the matter. Conventionally, a specific equation is used to predict the movement of an object, and the constant parameters used are based on previously studied values. In this study, we set this equation in a form suitable for parameter optimization irrespective of fluid dynamics and predicted the particle trajectory by setting the constant parameter used here to the optimal value through

evolutionary computation. This is a novel prediction method, and it is significant in that it suggests a new method of designing a prediction model.

Figure 4: Surface drifters

1.3.2 A Review of Gait Optimization Based on Evolutionary Computation

Compared to wheeled robots, legged robots usually possess superior mobility in uneven and unstructured environments. This is because they can use discrete footholds to overcome obstacles, climb stairs, and so forth, instead of relying on a continuous support surface.

A gait is a cyclic, periodic motion of the joints of a legged robot, requiring the sequencing or coordination of the legs to obtain reliable locomotion. In other words, gait is the temporal and spatial relationship between all the moving parts of a legged robot. Gait optimization is very important for legged robots, because it determines the optimal position, velocity and acceleration for each Degree of Freedom (DOF) at any moment in time, and the gait pattern will directly affect the robot's dynamic stabilization,

harmony, energy dissipation and so on. Gait optimization determines a legged robot's quality of movement.

Why Evolutionary Computation Is Suitable for Gait Optimization

Gait Generation Is a Multiconstrained, Multiobjective Optimization Problem

Gait generation, which incorporates mobility and stability, is a very challenging task for legged robots, because their system of locomotion has multiple DOFs and a variable mechanical structure during locomotion. As a result a large number of parameters have to be established. For example, 54 motion parameters have to be considered for the walk gait of the Sony AIBO robot. To obtain a natural and efficient gait for a legged robot, two kinds of strategies for sequencing or coordination of the leg movements can be followed.

The first strategy assumes that the gates of humans or animals are optimal, as otherwise they would not have been able to survive the competition and natural selection proposed by Darwin's Theory of Evolution. This assumption has been proved accurate. The constrained optimization hypothesis suggests that gait parameters are selected to optimize (minimize) the objective function of the cost of transport (metabolic cost/distance) within the limitations of imposed constraints. A lot of research has shown that humans and animals move in a way that minimizes the metabolic cost of locomotion and validates the idea that the gait synthesis of legged robot is a constrained optimization problem

Robots simulate human or animal behavior. Therefore, it is quite natural to use biological locomotion data to control the gait of robots. For example, Human Motion Captured Data has been adopted to drive a humanoid robot. However, some research indicates that biological locomotion data cannot be used directly for a legged robot due to kinematic and dynamic inconsistencies

between humans/animals and the legged robot. This implies the need for kinematic corrections when calculating joint angle trajectories

The second strategy formulates the gait generation problem of the legged robot as an optimization problem with constraints. It generates the optimal gait cycle by minimizing some performance indexes, for example, velocity of motion, stability criteria, actuating forces, energy consumption, and so forth. The gait generation problem of legged robots often has several objectives, and some of these objectives may be contradictory to each other (for example, speed and stability). Thus the gait generation problem can be stated as a multi-constrained and multi-objective optimization problem.

These two gait generation strategies may reach the same goal by different routes because both of them actually solve the gait synthesis problem as a multi-constrained multi-objective optimization problem. Once a database of precomputed optimal gaits has been created, the robot can cover the entire interval of precomputed optimal gaits by interpolation and thus realize smooth real time locomotion.

1.3.3 Evolutionary Computation Is Suitable for the Gait Optimization Problem

The dynamic equations of legged robot locomotion are high order highly coupled and nonlinear, and gait optimization for legged robots requires searching a set of parameters in a highly irregular, multidimensional space. As a result, the standard gradient search-based optimization methods are not useful for legged systems with high DOF.

Evolutionary Computation (EC), including the Genetic Algorithm (GA), Genetic Programming (GP), Evolutionary Programming (EP), and Evolutionary Strategy (ES), is a natural choice for the gait optimization of legged robots.

First, EC uses optimization methods based on Darwin's Natural Evolution Theory. According to this theory, the locomotion mechanisms of life forms resulted from natural selection and the interaction between individuals and the natural environment. This makes the use of EC a natural choice, as it is biologically inspired and can generate biologically plausible solutions.

Second, from the computational point of view, EC also fits well with the gait optimization of legged robots, because of the following:

- Gait optimization problems can have multiple criteria, multiple constraints, as well as multiple design variables, and EC has been shown effective for these kinds of large-dimension, multi-objective, multi-constraint optimization problems.
- EC has been seen to be robust for search and optimization problems and has been used to solve difficult problems with objective functions where local information such as continuity, differentiability, and so on is not available, even though it is very important for gait optimization, as the objective functions of gait optimization may be very complex and it is very difficult to obtain this local information.
- Because of the complexity and high DOF of the mechanical structure, it is difficult to obtain a precise dynamic model of a legged robot. EC will be efficient as this method is resistant to noise in the evaluation function and offers a model-free approach to optimization, only requiring feedback from the environment to improve performance when online evolution is deployed with a real robot.
- EC has strong global search capability and is also insensitive to the initial population. Therefore EC decreases the risk of being trapped in a local minimum for finding a true optimum solution.
- EC can easily be parallelized. Since gait optimization of legged robots is often a large-scale problem and the objective function and constraints are often complex, the process of evolutionary optimization may be very

time-consuming because of the high computational cost of EC due to iterative evaluations of candidate solutions. Therefore it is advantageous to use parallel implementations of EC to gain efficiency and improve the solution quality of EC-based gait optimization.

1.3.4 How to Evolve the Optimal Gait

The Multiform EC Models Adopted in Gait Optimization

Gait optimization based on EC is actually a combination of EC procedures and gait optimization problems. A general block diagram of EC-based gait optimization is given in Figure 5. This offers a first glance at the application of EC technique for gait optimization of legged robots.

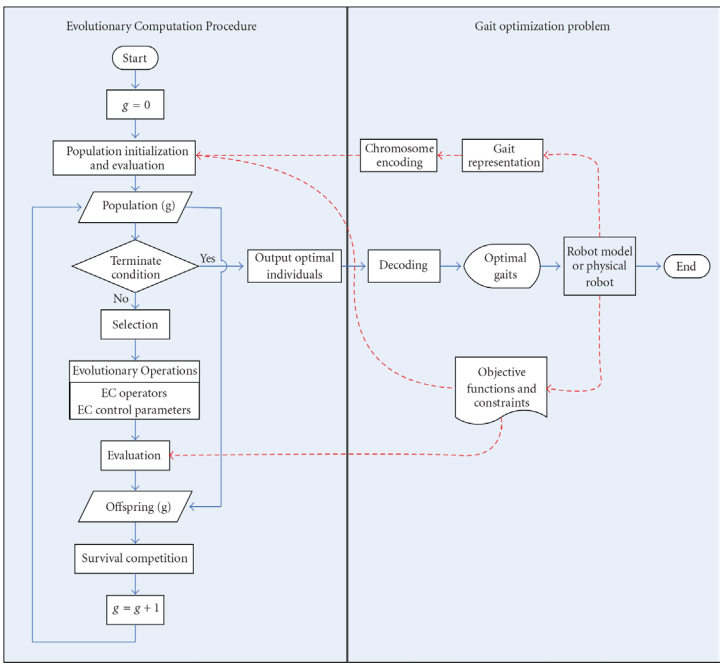


Figure 5: A general block diagram for EC-based gait optimization

A lot of EC models have been adopted to solve gait optimization problems. The gaits most often studied include the gaits of biped, quadruped, and hexapod robots engaged in walking, running, negotiating sloping surfaces, and going up and down stairs

The Genetic Algorithm (GA) is the gait optimization tool which is most often used, and some modifications can be introduced to fit the specific problems of gait optimization. For example, interpolating and extrapolating operators, two-point crossover, Gaussian mutation, overlapping populations, and Elitism strategy have been adopted. The explicit fitness sharing mechanism has also been adopted to prevent premature convergence to suboptimal extremes. This speciation technique divides the population into a fixed number of species, where each species contains individuals that are similar to each other, and can force similar members of the same species to "share" one representative score, thereby penalizing species with a large number of individuals and allowing new species to form even if they do not perform as well as other, larger, species.

Adaptability that can adaptively change the probabilities of crossover and mutation is introduced in GA to balance global and local exploitation and exploration towards the progress of evolutionary optimization. For instance, Adaptive GA is used to optimize the gait of a humanoid robot ascending and descending a staircase by searching optimal trajectory parameters in blending polynomials.

Adaptive mechanisms may also be applied to control mutation rate. This method places radiation (the level of radiation decreases over time) into the middle of a region where a large group of individuals is clustered within the same locality to dramatically increase the mutation rate in this area, causing all the individuals to mutate in the next generation and to disperse to other areas of the space. It is reported that this mechanism can be useful in controlling the learning behavior of GA and makes GA more robust with respect to noise in parameter evaluations preventing premature convergence to suboptimal extremes.

Simulation results obtained using GP on an AIBO quadruped in the Webots environment are reported much better than those obtained using simple GA-based approaches. In this approach, the gait is defined using joint angle trajectories instead of locus of paw to reduce the search space of optimization. An elite archive mechanism (EAM) is used to prevent premature convergence and improve the search capability of GP. EAM can preserve elite individuals at an early stage and flow them into in later stage. In this way, genetic material from elite individuals at an early stage is used to refresh an evolutionary convergent state and to create a role for preserving diversity as long as possible.

GE is one of the most popular forms of grammar-based GP. The advantage of GE lies in that it allows the user to conveniently specify and modify the grammar, whilst ignoring the task of designing specific genetic search operators. Thus GE can be used to optimize pre-existing motion data or generate novel motions. Using a Fourier gait representation to encode the chromosome and the dynamic similarity principles as a constraint, GE is employed to optimize the gait retargeting problem in animation. It successfully modified one animal's gait cycle data into a different animal's gait cycle data in computer simulations of animal locomotion. The same method can also be used to optimize the gait of a walking horse from a veterinary publication into a physics-based horse mode.

ES is also employed to solve the gait optimization problem, and some encouraging results have been obtained.

Using a hand-tuned gait as a seed, the bipedal gait is directly evolved on a physical robot by an ES approach with parametric mutation and structural mutation. After hundreds of evaluations significant improvements were obtained for a functioning but nonoptimized bipedal gait that improved the walking speed by around 65% compared to the hand-tuned gait.

A hybrid approach of space-time optimization and covariance matrix adaptation evolution strategy (CMA-ES) has been proposed to generate gaits and morphologies for legged animal

locomotion. It effectively generated dynamic locomotion gait of bipeds, a quadruped, as well as an imaginary five-legged creature by simulation. The gaits and morphologies produced are reported lifelike and exhibit many qualitative traits seen in real animals. This hybrid approach may combine the efficiency in high-dimensional spaces and the ability to handle general constraints of space-time optimization with the ability to handle nondifferentiable variables and to avoid many local minima from CMA.

Apart from traditional EC and its variations, some relatively new types of EC have also been applied to gait optimization research.

Estimation of Distribution Algorithms (EDAs) are evolutionary algorithms based on probabilistic models that replace the operators of mutation and crossover used in GAs. The main advantage of EDA lies in the fact that the knowledge about the problem acquired previously can be used to set the initial probability model, and the global statistical information about the search space can be extracted directly by EDA to modify the probability model with promising solutions. This can reduce the search space and obtain good solutions in a shorter time interval. For this reason EDA has been used to study the gait optimization problem. For example, EDA has been applied to optimize the gait of the AIBO robot. A fitness function based on direct evaluation of the robots was adopted, and significant improvement of the previous gait was achieved over a short training period

In some cases of gait optimization, the performance of a gait cannot be directly measured or calculated based on certain functions. In this case human preferences, intuition, emotions, and other psychological aspects can be introduced into the target system. Interactive evolutionary computation (IEC) is a form of evolutionary computation where the fitness function can be replaced by the user. A prominent advantage of IEC is that it can reflect user preference and allow optimization of the solution with a minimum of required knowledge in the problem domain.

The multiobjective multiconstraint problem is often solved by combining the multiple objectives and constraints into a single

scalar objective problem using weighting coefficients. To do this, some problem-specific information is needed, and the relative importance of the objectives and constraints should be decided. In the complex problem of gait optimization, it is difficult to know this information in advance. In addition, there is no rational basis for determining adequate weights for these competitive or conflicting criteria, and the objective function that will be formed may lose significance due to the combination of noncommensurable objectives. Therefore, more and more gait optimization problems are parameterized and optimized using tailored Multiple Objective EC procedures, for example, the Strength Pareto Evolutionary Algorithm and Nondominated Sorting Genetic Algorithm with Fitness Sharing method, and the obtained Pareto-optimal gaits, which is a set of nondominated or noninferior gaits that satisfies different objective functions. These methods have shown good performance

1.3.5 Gait Representation and Chromosome Encoding

The gait of a legged robot may be represented in three-dimensional space or in joint space. In order to control a legged robot's movement, it is necessary to generate the trajectories of all the joints. Therefore, gait is usually represented by a sequence of key poses (states) extracted from one complete gait cycle, and phases between these key poses are approximated by a polynomial function, for example, 3rd, 4th, or 5th order polynomials. These polynomial functions are adopted because they can insure that the joint trajectories are smoothly connected with first-order and second-order derivative continuity. First order derivative continuity guarantees the smoothness of velocity, while the second order guarantees the smoothness requirements of acceleration or the torque in the joints. As a result, the gait of robot will look natural.

If only the foot placement point of these key poses is specified, once the foot trajectories are generated, inverse kinematics should then be used to convert the locus of foot into the joint angles required to generate the foot placement curves for a particular gait.

To make the robot optimally move from its current position/stance to a goal position/stance, other parameters apart from those of the leg joint trajectories should also be considered, for example, parameters describing the position and orientation of the body, how the robot's weight shifts during walking, whether or how much the arms swing, and so forth.

The joint angles in these states, the coefficients of the polynomials, and some of the other parameters mentioned above are the design variables to be optimized by EC. These design variables, when treated as genes and arranged in an array, make up a chromosome of EC.

A variety of chromosome encoding methods, including the gray code representation, real number coding, mixed encoding of floating point number, and binary number, have been adopted, but the most often used encoding is the real coded method. This is due to difficulties associated with binary representation when dealing with a continuous search space with large dimension

1.3.6 Comparing EC with Other Global Optimization Approaches

Besides evolution-based optimization techniques, other global optimization approaches that adopt a non-evolutionary metaphor have also been employed in gait optimization. These also search for the global optimum of the cost function without using the differential information of a given cost function.

Particle Swarm Optimization (PSO) can be used to optimize the stable and straight movement patterns (gaits) of a humanoid robot with the control signals of the joint angles produced by a Truncated Fourier Series (TFS). It is reported that PSO optimized TFS significantly faster and better than GA to generate straighter and faster humanoid locomotion because PSO bypassed a local minimum that GA was caught in. The authors therefore concluded that PSO is better than GA as a learning method for the gait optimization problem in a non-deterministic environment.

We argue that GA may not necessarily be inferior to PSO in gait optimization, even in a non-deterministic environment such as the one in this experiment. This is because the PSO employed in this experiment was Adaptive PSO, which has a dynamically adjustable nonlinear parameter of inertia weight to control the balance between global and local exploration. A larger inertia weight facilitates a global search, while a smaller inertia weight facilitates a local search. The GA employed in this experiment is just a canonical paradigm with roulette wheel selection and a fixed rate of crossover and mutation. This may be the reason why PSO can speed up the search and perform better than GA in this experiment.

EC of course can employ the same mechanism to improve its efficiency. For example, Adaptive GA can adaptively change the probabilities of crossover and mutation during the process of evolution. In ES, the step size or mutation strength is often governed by self-adaptation (evolution window), and the individual step sizes for each coordinate or correlations between coordinates are either governed by self-adaptation or also by covariance matrix adaptation (CMA-ES).

Adaptive PSO is used to optimize the fastest forward gaits of the quadruped robot AIBO with the whole learning process running automatically on the physical robot. Starting with randomly generated parameters instead of hand-tuned parameters, several high-performance sets of gait parameters are obtained, and these gains were reported as being among the fastest forward gaits ever developed for the same robot platform.

Parallel PSO was applied to large-scale human movement problems, and experimental results show that PSO was outperformed by the gradient-based algorithm. It is reported that a single run with a gradient-based nonlinear least squares algorithm produced a significantly better solution than did 10 runs using global PSO. Thus the authors do not recommend using the PSO algorithm for solving large-scale human movement optimization problems possessing constraints or competing terms in the cost function.

The results of this experiment may be a fortunate exception. The objective functions of large-scale gait optimization problems with hundreds of design variables will no doubt be massively multimodal and the landscape must be very rugged. Therefore a gradient-based algorithm will certainly be trapped in a local minimum, and the global search ability of EC is absolutely necessary for decreasing this risk. We agree with the suggestion of the authors that a global local hybrid algorithm may be necessary for PSO and other global optimizers to solve large-scale human movement problems efficiently.

As far as we have seen from the literature, Ant Colony Optimization (ACO) has not yet been used in the field of gait optimization though this too is a famous metaheuristic of Swarm Intelligence (SI) similar to PSO and has been widely used to solve a lot of kinds of optimization problems.

The univariate dynamic encoding algorithm for searches (uDEAS) has also been applied to the gait optimization problem of a biped model walking up and down a staircase. The simulation results show that uDEAS outperforms adaptive GA with a 17 s versus 126 s run time on average and a slightly smaller minimum for best cost values. The authors attribute this result to the effectiveness of describing trajectories with the blending polynomial of uDEAS.

The problem representation method and the genotype encoding method directly determine the size and the characteristic of the search space and as a result directly affect the efficiency of EC optimization. For example, TFS is reported to be a good gait representation approach that can generate suitable angular trajectories for controlling bipedal locomotion. This is because it does not require inverse kinematics, and stable gaits with different step lengths and stride frequencies can be readily generated by changing the value of only one parameter in the TFS.

Though some comparison of performance between EC and other non-evolutionary global optimization approaches has been reported, no systematic comparative study has been carried out. Such a systematic comparative study may be not necessary or not

feasible because we often search for a set of satisfactory solutions instead of an absolutely global optimal solution. Both the robot platform and the objective functions of gait optimization will be different in each case, and thus it is difficult to find a benchmark robot and a set of benchmark objective functions to optimize.

Both EC and SI approaches are population-based iterative algorithms, even though they adopt different metaphors. Thus they share the same advantages and disadvantages in gait optimization, for example, a similar global and high-dimensional search capability, multi-objective optimization capability, as well as a lot of control parameters which require tuning. One thing that can be said for sure is that EC, and SI approaches are proven good tools for gait optimization for legged robots, and further research should be done to improve their performance in the field of gait optimization.

REFERENCES

1. F. A. von Hayek, *The Road to Serfdom: Text and Documents—The Definitive Edition*, University of Chicago Press, 2009.
2. J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.
3. J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 36, pp. 48–49, 1950.
4. K. J. Arrow, *Social Choice and Individual Values*, Wiley, 2nd edition, 1963.
5. L. Hurwicz, "The design of mechanisms for resource allocation," *The American Economic Review*, vol. 63, no. 2, pp. 1–30, 1973.
6. L. von Mises, *Socialism: An Economic and Sociological Analysis*, Laissez Faire Books, 1922.
7. M. D. Vose and G. E. Liepins, "Punctuated equilibria in genetic search," *Complex Systems*, vol. 5, no. 1, pp. 31–44, 1991.
8. O. Lange, "On the economic theory of socialism: part one," *The Review of Economic Studies*, vol. 4, no. 1, pp. 53–71, 1936.
9. R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
10. Y. Pei, "Chaotic evolution: fusion of chaotic ergodicity and evolutionary iteration for optimization," *Natural Computing*, vol. 13, no. 1, pp. 79–96, 2014.
11. L. Chen, P. Yang, Z. Liu, H. Chen, and X. Guo, "Gait optimization of biped robot based on mix-encoding genetic algorithm," in *Proceedings of the 2nd IEEE Conference on Industrial Electronics and Applications (ICIEA '07)*, pp. 1623–1626, May 2007.

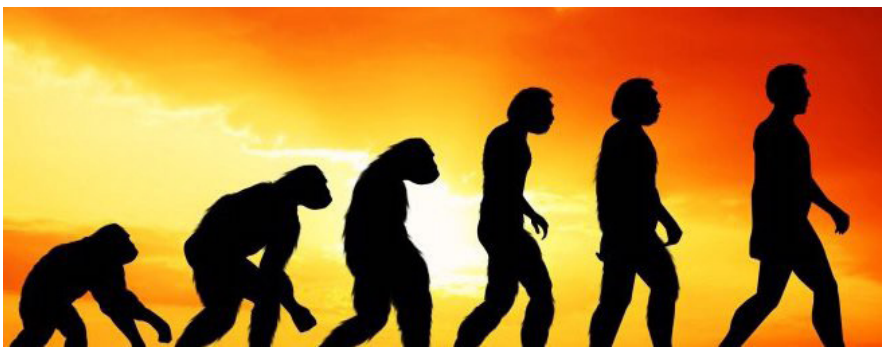
12. S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*, pp. 2562–2567, October 2004.
13. J. E. A. Bertram and A. Ruina, "Multiple walking speed-frequency relations are predicted by constrained optimization," *Journal of Theoretical Biology*, vol. 209, no. 4, pp. 445–453, 2001.

CHAPTER 2

EVOLUTIONARY ALGORITHM

INTRODUCTION

An evolutionary algorithm is an evolutionary AI-based computer application that solves problems by employing processes that mimic the behaviors of living things. As such, it uses mechanisms that are typically associated with biological evolution, such as reproduction, mutation and recombination.



Evolutionary algorithms function in a Darwinian-like natural selection process; the weakest solutions are eliminated while stronger, more viable options are retained and re-evaluated in the next evolution—with the goal being to arrive at optimal actions to achieve the desired outcomes.

2.1 EVOLUTIONARY ALGORITHM

As the history of the field suggests, there are many different variants of evolutionary algorithms. The common underlying idea behind all these techniques is the same: given a population of individuals within some environment that has limited resources, competition for those resources causes natural selection (survival of the fittest). This in turn causes a rise in the fitness of the population. Given a quality function to be maximized, we can randomly create a set of candidate solutions, i.e., elements of the function's domain. We then apply the quality function to these as an abstract fitness measure – the higher the better. On the basis of these fitness values some of the better candidates are chosen to seed the next generation. This is done by applying recombination and/or mutation to them. Recombination is an operator that is applied to two or more selected candidates (the so-called parents), producing one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Therefore executing the operations of recombination and mutation on the parents leads to the creation of a set of new candidates (the offspring). These have their fitness evaluated and then compete – based on their fitness (and possibly age) – with the old ones for a place in the next generation. This process can be iterated until a candidate with sufficient quality (a solution) is found or a previously set computational limit is reached.

There are two main forces that form the basis of evolutionary systems:

- Variation operators (recombination and mutation) create the necessary diversity within the population, and thereby facilitate novelty.

- Selection acts as a force increasing the mean quality of solutions in the population.

The combined application of variation and selection generally leads to improving fitness values in consecutive populations. It is easy to view this process as if evolution is optimizing (or at least 'approximising') the fitness function, by approaching the optimal values closer and closer over time. An alternative view is that evolution may be seen as a process of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements. Matching these requirements more closely implies an increased viability, which is reflected in a higher number of offspring. The evolutionary process results in a population which is increasingly better adapted to the environment.

It should be noted that many components of such an evolutionary process are stochastic. For example, during selection the best individuals are not chosen deterministically, and typically even the weak individuals have some chance of becoming a parent or of surviving. During the recombination process, the choice of which pieces from the parents will be recombined is made at random. Similarly for mutation, the choice of which pieces will be changed within a candidate solution, and of the new pieces to replace them, is made randomly. The general scheme of an evolutionary algorithm is given in pseudocode in Fig. 1, and is shown as a flowchart in Fig. 2.

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Figure 1. The general scheme of an evolutionary algorithm in pseudo-code.

It is easy to see that this scheme falls into the category of generate-and-test algorithms. The evaluation (fitness) function provides a heuristic estimate of solution quality, and the search process is driven by the variation and selection operators. Evolutionary algorithms possess a number of features that can help position them within the family of generate-and-test methods:

- EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously.
- Most EAs use recombination, mixing information from two or more candidate solutions to create a new one.
- EAs are stochastic.

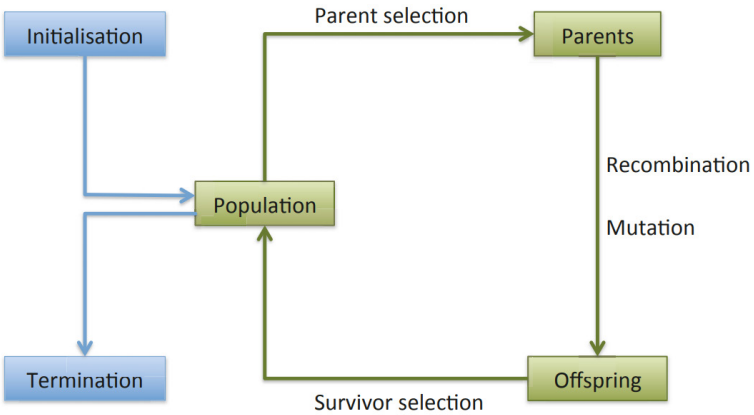


Figure 2. The general scheme of an evolutionary algorithm as a flow-chart.

In particular, different streams are often characterized by the representation of a candidate solution – that is to say the data structures used to encode candidates. Typically this has the form of strings over a finite alphabet in genetic algorithms (GAs), real-valued vectors in evolution strategies (ESs), finite state machines in classical evolutionary programming (EP), and trees in genetic programming (GP). The origin of these differences is mainly historical. Technically, one representation might be preferable to others if it matches the given problem better; that is, it makes the encoding of candidate solutions easier or more natural. For

instance, when solving a satisfiability problem with n logical variables, the straightforward choice is to use bit-strings of length n so that the contents of the ith bit would denote that variable i took the value true (1) or false (0). Hence, the appropriate EA would be a GA. To evolve a computer program that can play checkers, the parse trees of the syntactic expressions forming the programs are a natural choice to represent candidate solutions, thus a GP approach is likely. It is important to note two points. First, the recombination and mutation operators working on candidates must match the given representation. Thus, for instance, in GP the recombination operator works on trees, while in GAs it operates on strings. Second, in contrast to variation operators, the selection process only takes fitness information into account, and so it works independently from the choice of representation. Therefore differences between the selection mechanisms commonly applied in each stream are a matter of tradition rather than of technical necessity.

2.2 COMPONENTS OF EVOLUTIONARY ALGORITHMS

There are a number of components, procedures, or operators that must be specified in order to define a particular EA. The most important components are:

- representation (definition of individuals)
- evaluation function (or fitness function)
- population
- parent selection mechanism
- variation operators, recombination and mutation
- survivor selection mechanism (replacement)

To create a complete, runnable algorithm, it is necessary to specify each component and to define the initialization procedure. If we wish the algorithm to stop at some stage, we must also provide a termination condition.

2.2.1 Representation (Definition of Individuals)

The first step in defining an EA is to link the 'real world' to the 'EA world', that is, to set up a bridge between the original problem context and the problem-solving space where evolution takes place. This often involves simplifying or abstracting some aspects of the real world to create a well-defined and tangible problem context within which possible solutions can exist and be evaluated, and this work is often undertaken by domain experts. The first step from the point of view of automated problem-solving is to decide how possible solutions should be specified and stored in a way that can be manipulated by a computer. We say that objects forming possible solutions within the original problem context are referred to as phenotypes, while their encoding, that is, the individuals within the EA, are called genotypes. This first design step is commonly called representation, as it amounts to specifying a mapping from the phenotypes onto a set of genotypes that are said to represent them. For instance, given an optimization problem where the possible solutions are integers, the given set of integers would form the set of phenotypes. In this case one could decide to represent them by their binary code, so, for example, the value 18 would be seen as a phenotype, and 10010 as a genotype representing it. It is important to understand that the phenotype space can be very different from the genotype space, and that the whole evolutionary search takes place in the genotype space. A solution – a good phenotype – is obtained by decoding the best genotype after termination. Therefore it is desirable that the (optimal) solution to the problem at hand – a phenotype – is represented in the given genotype space. In fact, since in general we will not know in advance what that solution looks like, it is usually desirable that all possible feasible solutions can be represented.

The evolutionary computation literature contains many synonyms:

- On the side of the original problem context the terms candidate solution, phenotype, and individual are all used to denote possible solutions. The space of all possible candidate solutions is commonly called the phenotype space.

- On the side of the EA, the terms genotype, chromosome, and again individual are used to denote points in the space where the evolutionary search actually takes place. This space is often termed the genotype space.
- There are also many synonymous terms for the elements of individuals. A placeholder is commonly called a variable, a locus (plural: loci), a position, or – in a biology-oriented terminology – a gene. An object in such a place can be called a value or an allele.

It should be noted that the word 'representation' is used in two slightly different ways. Sometimes it stands for the mapping from the phenotype to the genotype space. In this sense it is synonymous with encoding, e.g., one could mention binary representation or binary encoding of candidate solutions. The inverse mapping from genotypes to phenotypes is usually called decoding, and it is necessary that the representation should be invertible so that for each genotype there is at most one corresponding phenotype. The word representation can also be used in a slightly different sense, where the emphasis is not on the mapping itself, but on the data structure of the genotype space. This interpretation is the one we use when, for example, we speak about mutation operators for binary representation.

2.2.2 Evaluation Function (Fitness Function)

The role of the evaluation function is to represent the requirements the population should adapt to meet. It forms the basis for selection, and so it facilitates improvements. More accurately, it defines what improvement means. From the problem-solving perspective, it represents the task to be solved in the evolutionary context. Technically, it is a function or procedure that assigns a quality measure to genotypes. Typically, this function is composed from the inverse representation (to create the corresponding phenotype) followed by a quality measure in the phenotype space. To stick with the example above, if the task is to find an integer x that maximizes x^2, the fitness of the genotype 10010 could be

defined by decoding its corresponding phenotype (10010 → 18) and then taking its square: $18^2 = 324$.

The evaluation function is commonly called the fitness function in EC. This might cause a counterintuitive terminology if the original problem requires minimization, because the term fitness is usually associated with maximization. Mathematically, however, it is trivial to change minimization into maximization, and vice versa. Quite often, the original problem to be solved by an EA is an optimization problem. In this case the name objective function is often used in the original problem context, and the evaluation (fitness) function can be identical to, or a simple transformation of, the given objective function.

2.2.3 Population

The role of the population is to hold (the representation of) possible solutions. A population is a multiset of genotypes. The population forms the unit of evolution. Individuals are static objects that do not change or adapt; it is the population that does. Given a representation, defining a population may be as simple as specifying how many individuals are in it, that is, setting the population size. In some sophisticated EAs a population has an additional spatial structure, defined via a distance measure or a neighborhood relation. This corresponds loosely to the way that real populations evolve within the context of a spatial structure given by individuals' geographical locations. In such cases the additional structure must also be defined in order to fully specify a population.

In almost all EA applications the population size is constant and does not change during the evolutionary search – this produces the limited resources need to create competition. The selection operators (parent selection and survivor selection) work at the population level. In general, they take the whole current population into account, and choices are always made relative to what is currently present. For instance, the best individual of a given population is chosen to seed the next generation, or the

worst individual of a given population is chosen to be replaced by a new one. This population level activity is in contrast to variation operators, which act on one or more parent individuals.

The diversity of a population is a measure of the number of different solutions present. No single measure for diversity exists. Typically people might refer to the number of different fitness values present, the number of different phenotypes present, or the number of different genotypes. Other statistical measures such as entropy are also used. Note that the presence of only one fitness value in a population does not necessarily imply that only one phenotype is present, since many phenotypes may have the same fitness. Equally, the presence of only one phenotype does not necessarily imply only one genotype. However, if only one genotype is present then this implies only one phenotype and fitness value are present.

2.2.4 Parent Selection Mechanism

The role of parent selection or mate selection is to distinguish among individuals based on their quality, and, in particular, to allow the better individuals to become parents of the next generation. An individual is a parent if it has been selected to undergo variation in order to create offspring. Together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements. In EC, parent selection is typically probabilistic. Thus, high-quality individuals have more chance of becoming parents than those with low quality. Nevertheless, low-quality individuals are often given a small, but positive chance; otherwise the whole search could become too greedy and the population could get stuck in a local optimum.

2.2.5 Variation Operators (Mutation and Recombination)

The role of variation operators is to create new individuals from old ones. In the corresponding phenotype space this amounts to generating new candidate solutions. From the generate-and-test

search perspective, variation operators perform the generate step. Variation operators in EC are divided into two types based on their arity, distinguishing unary (mutation) and n-ary versions (recombination).

Mutation

A unary variation operator is commonly called mutation. It is applied to one genotype and delivers a (slightly) modified mutant, the child or offspring.

A mutation operator is always stochastic: its output – the child – depends on the outcomes of a series of random choices. It should be noted that not all unary operators are seen as mutation. For example, it might be tempting to use the term mutation to describe a problem-specific heuristic operator which acts systematically on one individual trying to find its weak spot and improve it by performing a small change. However, in general mutation is supposed to cause a random, unbiased change. For this reason it might be more appropriate not to call heuristic unary operators mutation. Historically, mutation has played a different role in various EC dialects. Thus, for example, in genetic programming it is often not used at all, whereas in genetic algorithms it has traditionally been seen as a background operator, providing the gene pool with 'fresh blood', and in evolutionary programming it is the only variation operator, solely responsible for the generation of new individuals.

Variation operators form the evolutionary implementation of elementary (search) steps, giving the search space its topological structure. Generating a child amounts to stepping to a new point in this space. From this perspective, mutation has a theoretical role as well: it can guarantee that the space is connected. There are theorems which state that an EA will (given sufficient time) discover the global optimum of a given problem. These often rely on this connectedness property that each genotype representing a possible solution can be reached by the variation operators. The simplest way to satisfy this condition is to allow the mutation

operator to jump everywhere: for example, by allowing any allele to be mutated into any other with a nonzero probability. However, many researchers feel these proofs have limited practical importance, and EA implementations often don't possess this property.

Recombination

A binary variation operator is called recombination or crossover. As the names indicate, such an operator merges information from two parent genotypes into one or two offspring genotypes. Like mutation, recombination is a stochastic operator: the choices of what parts of each parent are combined, and how this is done, depend on random drawings. Again, the role of recombination differs between EC dialects: in genetic programming it is often the only variation operator, and in genetic algorithms it is seen as the main search operator, whereas in evolutionary programming it is never used. Recombination operators with a higher arity (using more than two parents) are mathematically possible and easy to implement, but have no biological equivalent. Perhaps this is why they are not commonly used, although several studies indicate that they have positive effects on the evolution.

The principle behind recombination is simple – by mating two individuals with different but desirable features, we can produce an offspring that combines both of those features. This principle has a strong supporting case – for millennia it has been successfully applied by plant and livestock breeders to produce species that give higher yields or have other desirable features. Evolutionary algorithms create a number of offspring by random recombination, and we hope that while some will have undesirable combinations of traits, and most may be no better or worse than their parents, some will have improved characteristics. The biology of the planet Earth, where, with very few exceptions, lower organisms reproduce asexually and higher organisms reproduce sexually, suggests that recombination is the superior form of reproduction. However recombination operators in EAs are usually applied probabilistically, that is, with a nonzero chance of not being

performed. It is important to remember that variation operators are representation dependent. Thus for different representations different variation operators have to be defined. For example, if genotypes are bit-strings, then inverting a bit can be used as a mutation operator. However, if we represent possible solutions by tree-like structures another mutation operator is required.

2.2.6 Survivor Selection Mechanism (Replacement)

Similar to parent selection, the role of survivor selection or environmental selection is to distinguish among individuals based on their quality. However, it is used in a different stage of the evolutionary cycle – the survivor selection mechanism is called after the creation of the offspring from the selected parents. In EC the population size is almost always constant. This requires a choice to be made about which individuals will be allowed in to the next generation. This decision is often based on their fitness values, favoring those with higher quality, although the concept of age is also frequently used. In contrast to parent selection, which is typically stochastic, survivor selection is often deterministic. Thus, for example, two common methods are the fitness-based method of ranking the unified multi-set of parents and offspring and selecting the top segment, or the age-biased approach of selecting only from the offspring.

Survivor selection is also often called the replacement strategy. In many cases the two terms can be used interchangeably, but we use the name survivor selection to keep terminology consistent: steps 1 and 5 in Fig. 1 are both named selection, distinguished by a qualifier. Equally, if the algorithm creates surplus children (e.g., 500 offspring from a population of 100), then using the term survivor selection is clearly appropriate. On the other hand, the term "replacement" might be preferred if the number of newly-created children is small compared to the number of individuals in the population. For example, a "steady-state" algorithm might generate two children per iteration from a population of 100. In this case, survivor selection means choosing the two old individuals that are to be deleted to make space for the new ones, so it is

more efficient to declare that everybody survives unless deleted and to choose whom to replace. Both strategies can of course be seen in nature, and have their proponents in EC, so in the rest of this book we will be pragmatic about this issue. We will use survivor selection in the section headers for reasons of generality and uniformity, while using replacement if it is commonly used in the literature for the given procedure we are discussing.

2.2.7 Initialization

Initialization is kept simple in most EA applications; the first population is seeded by randomly generated individuals. In principle, problem-specific heuristics can be used in this step, to create an initial population with higher fitness. Whether this is worth the extra computational effort, or not, very much depends on the application at hand.

2.2.8 Termination Condition

We can distinguish two cases of a suitable termination condition. If the problem has a known optimal fitness level, probably coming from a known optimum of the given objective function, then in an ideal world our stopping condition would be the discovery of a solution with this fitness. If we know that our model of the real-world problem contains necessary simplifications, or may contain noise, we may accept a solution that reaches the optimal fitness to within a given precision $\epsilon > 0$. However, EAs are stochastic and mostly there are no guarantees of reaching such an optimum, so this condition might never get satisfied, and the algorithm may never stop. Therefore we must extend this condition with one that certainly stops the algorithm. The following options are commonly used for this purpose:

1. The maximally allowed CPU time elapses.
2. The total number of fitness evaluations reaches a given limit.

3. The fitness improvement remains under a threshold value for a given period of time (i.e., for a number of generations or fitness evaluations).
4. The population diversity drops under a given threshold.

Technically, the actual termination criterion in such cases is a disjunction: optimum value hit or condition X satisfied. If the problem does not have a known optimum, then we need no disjunction. We simply need a condition from the above list, or a similar one that is guaranteed to stop the algorithm.

2.3 TYPES OF EVOLUTIONARY ALGORITHM (EA)

Evolutionary algorithms (EAs) are population-based metaheuristics. Historically, the design of EAs was motivated by observations about natural evolution in biological populations. Recent varieties of EA tend to include a broad mixture of influences in their design, although biological terminology is still in common use. The term 'EA' is also sometimes extended to algorithms that are motivated by population-based aspects of EAs, but which are not directly descended from traditional EAs, such as scatter search. The term evolutionary computation is also used to refer to EAs, but usually as a generic term that includes optimization algorithms motivated by other natural processes, such as particle swarm optimization and artificial immune systems.

The main classes of EA in contemporary usage are (in order of popularity) genetic algorithms (GAs), evolution strategies (ESs), differential evolution (DE) and estimation of distribution algorithms (EDAs). Multi-objective evolutionary algorithms (MOEAs), which generalize EAs to the multiple objective case, and memetic algorithms (MAs), which hybridize EAs with local search, are also popular, particularly within applied work. Special-purpose EAs, such as genetic programming (GP) and learning classifier systems (LCS) are also widely used.

Similar techniques differ in genetic representation and other implementation details, and the nature of the particular applied problem.

2.3.1 Genetic Algorithm

Genetic algorithms, or GAs, are one of the earliest forms of EA, and remain widely used. Candidate solutions, often referred to as chromosomes in the GA literature, comprise a vector of decision variables. Nowadays, these variables tend to have a direct mapping to an optimization domain, with each decision variable (or gene) in the GA chromosome representing a value (or allele) that is to be optimized. However, it should be noted that historically GAs worked with binary strings, with real values encoded by multiple binary symbols, and that this practice is still sometimes used. GA solution vectors are either fixed-length or variable-length, with the former the more common of the two.

Given their long history, genetic algorithm implementations vary considerably. However, it is fairly common to use a mutation operator that changes each decision variable with a certain probability (values of 4-8% are typical, depending upon the problem domain). When the solution vector is a binary string, the effect of the mutation operator is simply to flip the value. More generally, if the solution vector is a 'k-ary' string, in which each position can take any of a discrete set of k possible values, then the mutation operator is usually designed to choose a random new value from the available alphabet. If the solution vector is a string of real-valued parameters within a set range, the new value may be sampled from a uniform distribution in that range, or it may be sampled from a non-uniform (e.g. Gaussian) probability distribution centered around the current value. The latter is generally the preferred approach, since it leads to less disruptive change on average. Recombination is typically implemented using two-point or uniform crossover. Two-point crossover chooses two parent solutions and two crossover points within the solutions. The values of the decision variables lying between these two points are then swapped to form two child solutions. Uniform crossover is similar, except that crossover points are created at each decision variable with a given probability. Other forms of crossover have also been used in GAs. Examples include line crossover and multi-parent crossover. Other variation operators,

such as inversion, have been found useful for some problems. Various forms of selection are used with GAs. Rank-based or tournament selection are generally preferred, since they maintain exploration better than the more traditional fitness-proportionate selection (e.g. roulette-wheel selection). Note, however, that the latter is still widely used. Rank-based selection involves ranking the population in terms of objective value. Population members are then chosen to become parents with a probability proportional to their rank. In tournament selection, a small group of solutions (typically 3 or 4) are uniformly sampled from the population, and those with the highest objective value(s) become the parent(s) of the next child solution that is created. Tournament selection allows selective pressure to be easily varied by adjusting the tournament size.

2.3.2 Genetic Programming

Genetic programming (GP) is relatively new; it is a specialized form of a GA which operates on very specific types of solution, using modified genetic operators. The GP was developed by Koza as an attempt to find the way for the automatic generation of the program codes when the evaluation criteria for their proper operation is known. Because the searched solution is a program, the evolved potential solutions are coded in the form of trees instead of linear chromosomes (of bits or numbers) widespread in GAs. Of course, the genetic operators are specialized for working on trees, e.g., crossover as exchanging the subtrees, mutation as a change of node or leaf.

2.3.3 Evolutionary Programming

Evolutionary programming (EP) was developed as a tool for discovering the grammar of the unknown language. However, EP became more popular when it was proposed as the numerical optimization technique. The EP is similar to the ES ($\mu + \lambda$), but with one essential difference. In EP, the new population of individuals is created by mutating every individual from the

parental population, while in the ES($\mu + \lambda$), every individual has the same probability to be selected to the temporary population on which the genetic operations are performed. In the EP, the mutation is based on the random perturbation of the values of the particular genes of the mutated individual. The newly created and the parental populations are the same sizes ($\mu = \lambda$). Finally, the new generation of the population is created using the ranking selection of the individuals from both, the parental and the mutated populations.

2.3.4 Gene Expression Programming

Gene expression programming (GEP) is an evolutionary algorithm that creates computer programs or models. These computer programs are complex tree structures that learn and adapt by changing their sizes, shapes, and composition, much like a living organism. And like living organisms, the computer programs of GEP are also encoded in simple linear chromosomes of fixed length. Thus, GEP is a genotype–phenotype system, benefiting from a simple genome to keep and transmit the genetic information and a complex phenotype to explore the environment and adapt to it.

2.3.5 Evolution Strategy

Evolution strategies, or ESs, also have a long history, and this parallels the development of GAs. Whilst early ESs were restricted to a single search point and used no recombination operator, modern formulations have converged towards the GA norms, and tend to use both a population of search points and recombination. A lasting difference, however, is how they carry out mutation, with ESs using strategies that guide how the mutation operator is applied to each decision variable. Unlike GAs, ESs mutate every decision variable at each application of the operator, and do so according to a set of strategy parameters that determine the magnitude of these changes. Strategy parameters usually control characteristics of probability distributions from which the new values of decision variables are drawn.

It is standard practice to adapt strategy parameters over the course of an ES run, the basic idea being that different types of move will be beneficial at different stages of search. Various techniques have been used to achieve this adaptation. Some of these involve applying a simple formula, e.g. the 1/5th rule, which involves increasing or decreasing the magnitude of changes based on the number of successful mutations that have recently been observed. Others are based around the idea of self-adaptation, which involves encoding the strategy parameters as additional decision variables, and hence allowing evolution to come up with appropriate values. However, the most widely used contemporary approach is covariance matrix adaptation (CMA-ES), which uses a mechanism for estimating the directions of productive gradients within the search space, and then applying moves in those directions. In this respect, CMA-ES has similarities with gradient-based optimization methods.

ESs use different recombination operators to GAs, and often use more than two parents to create each child solution. For example, intermediate recombination gives a child solution the average values of each decision variable in each of the parent solutions. Weighted multi-recombination is similar, but uses a weighted average, based on the fitness of each parent. Also unlike GAs, ESs tend to use deterministic rather than probabilistic selection mechanisms, whereby the best solutions in the population are always used as parents of the next generation.

2.3.6 Differential Evolution

Differential evolution (DE) is a relatively recent EA formulation which uses a mechanism for adaptive search that does not make use of probability distributions. Whilst its basic mechanism is similar to a GA, its mutation operator is quite different, using a geometric approach that is motivated by the moves performed in the Nelder Mead simplex search method. This involves selecting two existing search points from the population, taking their vector difference, scaling this by a constant F, and then adding this to a third search point, again sampled randomly from the population.

Following mutation, DE's crossover operator recombines the mutated search point (the mutant vector) with another existing search point (the target vector), replacing it if the child solution (known as a trial vector) is of equal or greater objective value. There are two standard forms of crossover: exponential crossover and binomial crossover, which closely resemble GA two-point crossover and uniform crossover, respectively. The comparisons between target vector and trial vector play the same role as the selection mechanism in a GA or ES. Since DE requires each existing solution to be used once as a target vector, the whole population is replaced in the course of applying crossover.

An advantage of using simplex-like mutations in DE is that the algorithm is largely self-adapting, with moves automatically becoming smaller in each dimension as the population converges. More generally, the authors of the method have claimed that this sort of self-adaptation means that the size and direction of moves are automatically matched to the search landscape, a phenomenon they term contour matching. When compared to CMA-ES, for example, this means that the algorithm has few parameters and is relatively easy to implement.

2.3.7 Neuroevolution

Neuroevolution is a form of artificial intelligence that uses evolutionary algorithms to generate artificial neural networks (ANN), parameters, topology and rules. It is most commonly applied in artificial life, general game playing and evolutionary robotics. The main benefit is that neuroevolution can be applied more widely than supervised learning algorithms, which require a syllabus of correct input-output pairs. In contrast, neuroevolution requires only a measure of a network's performance at a task. For example, the outcome of a game (i.e. whether one player won or lost) can be easily measured without providing labeled examples of desired strategies. Neuroevolution is commonly used as part of the reinforcement learning paradigm, and it can be contrasted with conventional deep learning techniques that use gradient descent on a neural network with a fixed topology.

2.3.8 Learning Classifier System

Learning classifier systems (LCS) are a paradigm of rule-based machine learning methods that combine a discovery component (e.g. typically a genetic algorithm) with a learning component (performing either supervised learning, reinforcement learning, or unsupervised learning). Learning classifier systems seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions (e.g. behavior modeling, classification, data mining, regression, function approximation, or game strategy). This approach allows complex solution spaces to be broken up into smaller, simpler parts.

The founding concepts behind learning classifier systems came from attempts to model complex adaptive systems, using rule-based agents to form an artificial cognitive system (i.e. artificial intelligence).

Here the solution is a set of classifiers (rules or conditions). A Michigan-LCS evolves at the level of individual classifiers whereas a Pittsburgh-LCS uses populations of classifier-sets. Initially, classifiers were only binary, but now include real, neural net, or S-expression types. Fitness is typically determined with either a strength or accuracy based reinforcement learning or supervised learning approach.

2.4 AN EVOLUTIONARY CYCLE BY HAND

To illustrate the working of an EA, reproduction cycle on a simple problem after Goldberg, that of maximizing the values of x^2 for integers in the range 0–31. To execute a full evolutionary cycle, we must make design decisions regarding the EA components representation, parent selection, recombination, mutation, and survivor selection.

For the representation we use a simple five-bit binary encoding mapping integers (phenotypes) to bit-strings (genotypes). For parent selection we use a fitness proportional mechanism, where the probability pi that an individual i in population P is chosen to be a parent is $p_i = f(i)/\sum_{j \in P} f(j)$. Furthermore, we can decide to replace the entire population in one go by the offspring created from the selected parents. This means that our survivor selection operator is very simple: all existing individuals are removed from the population and all new individuals are added to it without comparing fitness values. This implies that we will create as many offspring as there are members in the population. Given our chosen representation, the mutation and recombination operators can be kept simple. Mutation is executed by generating a random number (from a uniform distribution over the range [0, 1]) in each bit position, and comparing it to a fixed threshold, usually called the mutation rate. If the random number is below that rate, the value of the gene in the corresponding position is flipped. Recombination is implemented by the classic one-point crossover. This operator is applied to two parents and produces two children by choosing a random crossover-point along the strings and swapping the bits of the parents after this point.

After having made the essential design decisions, we can execute a full selection–reproduction cycle. Table 1 shows a random initial population of four genotypes, the corresponding phenotypes, and their fitness values. The cycle then starts with selecting the parents to seed the next generation. The fourth column of Table 1 shows the expected number of copies of each individual after parent selection, being $f_i/\bar{f}$, where $\bar{f}$ denotes the average fitness (displayed values are rounded up). As can be seen, these numbers are not integers; rather they represent a probability distribution, and the mating pool is created by making random choices to sample from this distribution. The column “Actual count” stands for the number of copies in the mating pool, i.e., it shows one possible outcome.

Table 1. The x^2 example, 1: initialization, evaluation, and parent selection.

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Next the selected individuals are paired at random, and for each pair a random point along the string is chosen. Table 2 shows the results of crossover on the given mating pool for crossover points after the fourth and second genes, respectively, together with the corresponding fitness values. Mutation is applied to the offspring delivered by crossover. Once again, we show one possible outcome of the random drawings, and Table 3 shows the hand-made 'mutants'. In this case, the mutations shown happen to have caused positive changes in fitness, but we should emphasize that in later generations, as the number of 1's in the population rises, mutation will be on average (but not always) deleterious. Although manually engineered, this example shows a typical progress: the average fitness grows from 293 to 588.5, and the best fitness in the population from 576 to 729 after crossover and mutation.

Table 2. The x^2 example, 2: crossover and offspring evaluation.

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3. The x^2 example, 3: mutation and offspring evaluation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

2.5 EXAMPLE APPLICATIONS

2.5.1 The Eight-Queens Problem

This is the problem of placing eight queens on a regular 8 × 8 chessboard so that no two of them can check each other. There are many classical artificial intelligence approaches to this problem, which work in a constructive, or incremental, fashion. They start by placing one queen, and after having placed n queens, they attempt to place the (n + 1)th in a feasible position where the new queen does not check any others. Typically some sort of backtracking mechanism is applied; if there is no feasible position for the (n+1) th queen, the nth is moved to another position.

An evolutionary approach to this problem is drastically different in that it is not incremental. Our candidate solutions are complete (rather than partial) board configurations, which specify the positions of all eight queens. The phenotype space P is the set of all such configurations. Clearly, most elements of this space are infeasible, violating the condition of non-checking queens. The quality q(p) of any phenotype p ◎ P can be simply quantified by the number of checking queen pairs. The lower this measure, the better a phenotype (board configuration), and a zero value, q(p) = 0, indicates a good solution. From this observation we can formulate a suitable objective function to be minimized, with a known optimal value. Even though we have not defined genotypes at this point,

we can state that the fitness (to be maximized) of a genotype g that represents phenotype p is some inverse of q(p). There are many possible ways of specifying what kind of inverse we wish to use here. For instance, 1/q(p) is an easy option, but has the disadvantage that attempting division by zero is a problem for many computing systems. We could circumvent this by watching for q(p) = 0 and saying that when this occurs we have a solution, or by adding a small value ϵ, i.e., $1/(q(p) + \epsilon)$. Other options are to use $-q(p)$ or $M - q(p)$, where M is a sufficiently large number to make all fitness values positive, e.g., $M \geq \max\{q(p) \mid p \in P\}$. This fitness function inherits the property of q that it has a known optimum M.

To design an EA to search the space P we need to define a representation of phenotypes from P. The most straightforward idea is to use a matrix representation of elements of P directly as genotypes, meaning that we must design variation operators for these matrices. In this example, however, we define a more clever representation as follows. A genotype, or chromosome, is a permutation of the numbers 1,..., 8, and a given $g = \langle i_1,...,i_8 \rangle$ denotes the (unique) board configuration, where the nth column contains exactly one queen placed on the inth row. For instance, the permutation $g = \langle 1,..., 8 \rangle$ represents a board where the queens are placed along the main diagonal. The genotype space G is now the set of all permutations of 1,..., 8 and we also have defined a mapping $F : G \rightarrow P$.

It is easy to see that by using such chromosomes we restrict the search to board configurations where horizontal constraint violations (two queens on the same row) and vertical constraint violations (two queens on the same column) do not occur. In other words, the representation guarantees half of the requirements of a solution – what remains to be minimized is the number of diagonal constraint violations. From a formal perspective we have chosen a representation that is not surjective since only part of P can be obtained by decoding elements of G. While in general this could carry the danger of missing solutions in P, in our present example this is not the case, since we know a priori that those phenotypes from $P \setminus F(G)$ can never be solutions.

The next step is to define suitable variation operators (mutation and crossover) for our representation, i.e., to work on genotypes that are permutations. The crucial feature of a suitable operator is that it does not lead out of the space G. In common parlance, the offspring of permutations must themselves be permutations. For mutation we can use an operator that randomly selects two positions in a given chromosome, and swaps the values found in those positions. A good crossover for permutations is less obvious, but the mechanism outlined in Fig. 3 will create two child permutations from two parents.

1. Select a random position, the crossover point, $i \in \{1, \ldots, 7\}$
2. Cut both parents into two segments at this position
3. Copy the first segment of parent 1 into child 1 and the first segment of parent 2 into child 2
4. Scan parent 2 from left to right and fill the second segment of child 1 with values from parent 2, skipping those that it already contains
5. Do the same for parent 1 and child 2

Figure 3. 'Cut-and-crossfill' crossover.

The important thing about these variation operators is that mutation causes a small undirected change, and crossover creates children that inherit genetic material from both parents. It should be noted though that there can be large performance differences between operators, e.g., an EA using mutation A might find a solution quickly, whereas one using mutation B might never find a solution. The operators we sketch here are not necessarily efficient; they merely serve as examples of operators that are applicable to the given representation.

The next step in setting up an EA is to decide upon the selection and population update mechanisms. We will choose a simple scheme for managing the population. In each evolutionary cycle we will select two parents, producing two children, and the new population of size n will contain the best n of the resulting n + 2 individuals (the old population plus the two new ones).

Parent selection will be done by choosing five individuals randomly from the population and taking the best two as parents. This ensures a bias towards using parents with relatively high

fitness. Survivor selection checks which old individuals should be deleted to make place for the new ones – provided the new ones are better. The strategy we will use merges the population and offspring, then ranks them according to fitness, and deletes the worst two.

To obtain a full specification we can decide to fill the initial population with randomly generated permutations, and to terminate the search when we find a solution, or when 10,000 fitness evaluations have elapsed, whichever happens sooner. Furthermore we can decide to use a population size of 100, and to use the variation operators with a certain frequency. For instance, we always apply crossover to the two selected parents and in 80% of the cases apply mutation to the offspring. Putting this all together, we obtain an EA as summarized in Table 4.

Table 4. Description of the EA for the eight-queens problem.

Representation	Permutations
Recombination	'Cut-and-crossfill' crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluations

2.5.2 The Knapsack Problem

The 0–1 knapsack problem, a generalization of many industrial problems. We are given a set of n items, each of which has attached to it some value v_i, and some cost c_i. The task is to select a subset of those items that maximizes the sum of the values, while keeping the summed cost within some capacity C_{max}. Thus, for example, when packing a backpack for a round-the-world trip, we must balance likely utility of the items against the fact that we have a limited volume (the items chosen must fit in one bag), and weight (airlines

impose fees for luggage over a given weight). It is a natural idea to represent candidate solutions for this problem as binary strings of length n, where a 1 in a given position indicates that an item is included and a 0 that it is omitted. The corresponding genotype space G is the set of all such strings with size 2^n, which increases exponentially with the number of items considered. Using this G, we fix the representation in the sense of data structure, and next we need to define the mapping from genotypes to phenotypes.

The first representation (in the sense of a mapping) that we consider takes the phenotype space P and the genotype space to be identical. The quality of a given solution p, represented by a binary genotype g, is thus determined by summing the values of the included items, i.e., $q(p) = \sum_{i=1}^{n} v_i \cdot g_i$. However, this simple representation leads us to some immediate problems. By using a one-to-one mapping between the genotype space G and the phenotype space P, individual genotypes may correspond to invalid solutions that have an associated cost greater than the capacity, i.e., $\sum_{i=1}^{n} c_i \cdot g_i > C_{max}$.

The second representation that we outline here solves this problem by employing a decoder function that breaks the one-to-one correspondence between the genotype space G and the solution space P. In essence, our genotype representation remains the same, but when creating a solution we read from left to right along the binary string, and keep a running tally of the cost of included items. When we encounter a value 1, we first check to see whether including the item would break our capacity constraint. In other words, rather than interpreting a value 1 as meaning include this item, we interpret it as meaning include this item IF it does not take us over the cost constraint. The effect of this scheme is to make the mapping from genotype to phenotype space many-to-one, since once the capacity has been reached, the values of all bits to the right of the current position are irrelevant, as no more items will be added to the solution. Furthermore, this mapping ensures that all binary strings represent valid solutions with a unique fitness (to be maximized).

Having decided on a fixed-length binary representation, we

can now choose off-the-shelf variation operators from the GA literature, because the bit-string representation is 'standard' there. A suitable (but not necessarily optimal) recombination operator is the so-called one-point crossover, where we align two parents and pick a random point along their length.

The two offspring are created by exchanging the tails of the parents at that point. We will apply this with 70% probability, i.e., for each pair of parents there is a 70% chance that we will create two offspring by crossover and 30% that the children will be just copies of the parents.

A suitable mutation operator is so-called bit-flipping: in each position we invert the value with a small probability pm $\in [0, 1)$.

In this case we will create the same number of offspring as we have members in our initial population. As noted above, we create two offspring from each two parents, so we will select that many parents and pair them randomly.

We will use a tournament for selecting the parents, where each time we pick two members of the population at random (with replacement), and the one with the highest value q(p) wins the tournament and becomes a parent. We will institute a generational scheme for survivor selection, i.e., all of the population in each iteration are discarded and replaced by their offspring.

Finally, we should consider initialization (which we will do by random choice of 0 and 1 in each position of our initial population), and termination. In this case, we do not know the maximum value that we can achieve, so we will run our algorithm until no improvement in the fitness of the best member of the population has been observed for 25 generations.

We have already defined our crossover probability as 0.7; we will work with a population size of 500 and a mutation rate of pm = 1/n, i.e., that will on average change one value in every offspring. Our evolutionary algorithm to tackle this problem can be specified as below in Table 5.

Table 5. Description of the EA for the knapsack problem.

Representation	Binary strings of length n
Recombination	One-point crossover
Recombination probability	70%
Mutation	Each value inverted with independent probability p_m
Mutation probability p_m	$1/n$
Parent selection	Best out of random 2
Survival selection	Generational
Population size	500
Number of offspring	500
Initialisation	Random
Termination condition	No improvement in last 25 generations

2.6 THE OPERATION OF AN EVOLUTIONARY ALGORITHM

Evolutionary algorithms have some rather general properties concerning how they work. To illustrate how an EA typically works, we will assume a one-dimensional objective function to be maximized. Figure 4 shows three stages of the evolutionary search, showing how the individuals might typically be distributed in the beginning, somewhere halfway, and at the end of the evolution. In the first stage directly after initialization, the individuals are randomly spread over the whole search space (Fig. 4, left). After only a few generations this distribution changes: because of selection and variation operators the population abandons low-fitness regions and starts to climb the hills (Fig. 4, middle). Yet later (close to the end of the search, if the termination condition is set appropriately), the whole population is concentrated around a few peaks, some of which may be suboptimal. In principle it is possible that the population might climb the wrong hill, leaving all of the individuals positioned around a local but not global optimum. Although there is no universally accepted rigorous definition of the terms exploration and exploitation, these notions are often used to categorize distinct phases of the search process. Roughly speaking, exploration is the generation of new individuals in as-yet untested regions of the search space, while exploitation means the concentration of the search in the vicinity of known good solutions. Evolutionary search processes are often referred

to in terms of a trade-off between exploration and exploitation. Too much of the former can lead to inefficient search, and too much of the latter can lead to a propensity to focus the search too quickly. Premature convergence is the well-known effect of losing population diversity too quickly, and getting trapped in a local optimum.

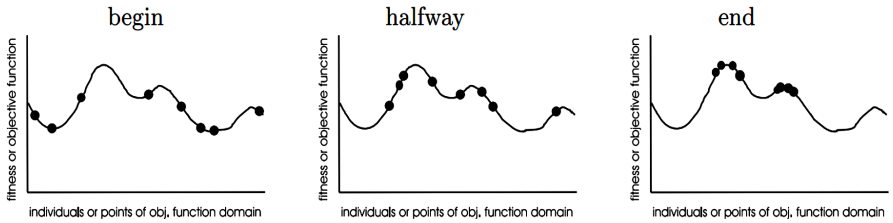


Figure 4. Typical progress of an EA illustrated in terms of population distribution. For each point x in the search space y shows the corresponding fitness value.

The other effect we want to illustrate is the anytime behavior of EAs by plotting the development of the population's best fitness value over time (Fig. 5). This curve shows rapid progress in the beginning and flattening out later on. This is typical for many algorithms that work by iterative improvements to the initial solution(s). The name 'anytime' comes from the property that the search can be stopped at any time, and the algorithm will have some solution, even if it is suboptimal. Based on this anytime curve we can make some general observations concerning initialization and the termination condition for EAs. We questioned whether it is worth putting extra computational effort into applying intelligent heuristics to seed the initial population with better-than-random individuals. In general, it could be said that that the typical progress curve of an evolutionary process makes it unnecessary. This is illustrated in Fig. 6. As the figure indicates, using heuristic initialization can start the evolutionary search with a better population. However, typically a few (k in the figure) generations are enough to reach this level, making the extra effort questionable.

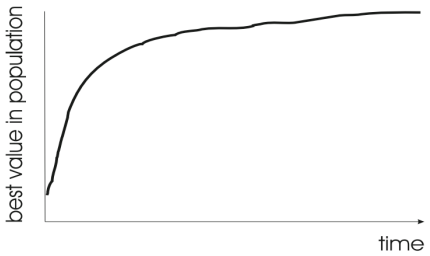


Figure 5. Typical progress of an EA illustrated in terms of development over time of the highest fitness in the population.

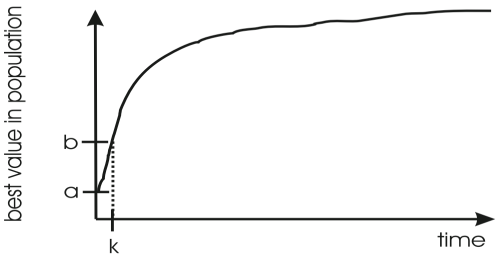


Figure 6. Illustration of why heuristic initialization might not be worth additional effort. Level a shows the best fitness in a randomly initialized population; level b belongs to heuristic initialization.

The anytime behavior also gives some general indications regarding the choice of termination conditions for EAs. In Fig. 7 we divide the run into two equally long sections. As the figure indicates, the progress in terms of fitness increase in the first half of the run (X) is significantly greater than in the second half (Y). This suggests that it might not be worth allowing very long runs. In other words, because of frequently observed anytime behavior of EAs, we might surmise that effort spent after a certain time (number of fitness evaluations) is unlikely to result in better solution quality.

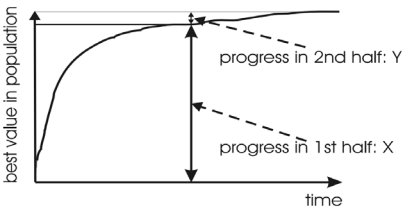


Figure 7. Why long runs might not be worth performing. X shows the fitness increase in the first half of the run, while Y belongs to the second half.

We close this review of EA behavior by looking at EA performance from a global perspective. That is, rather than observing one run of the algorithm, we consider the performance of EAs for a wide range of problems. Fig. 8 shows the 1980s view after Goldberg.

What the figure indicates is that EAs show a roughly evenly good performance over a wide range of problems. This performance pattern can be compared to random search and to algorithms tailored to a specific problem type.

EAs are suggested to clearly outperform random search. In contrast, a problem-tailored algorithm performs much better than an EA, but only on the type of problem for which it was designed. As we move away from this problem type to different problems, the problem-specific algorithm quickly loses performance. In this sense, EAs and problem-specific algorithms form two opposing extremes. This perception played an important role in positioning EAs and stressing the difference between evolutionary and random search, but it gradually changed in the 1990s based on new insights from practice as well as from theory. The contemporary view acknowledges the possibility of combining the two extremes into a hybrid algorithm. As for theoretical considerations, the No Free Lunch theorem has shown that (under some conditions) no black-box algorithm can outperform random walk when averaged over ‘all’ problems. That is, showing the EA line always above that of random search is fundamentally incorrect.

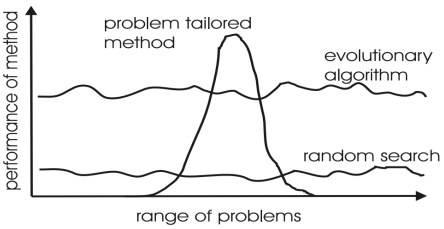


Figure 8. 1980s view of EA performance after Goldberg.

2.7 NATURAL VERSUS ARTIFICIAL EVOLUTION

From the perspective of the underlying substrate, the emergence of evolutionary computation can be considered as a major transition of the evolutionary principles from wetware, the realm of biology, to software, the realm of computers.

This was made possible by using computers as instruments for creating digital worlds that are very flexible and much more controllable than the physical reality we live in.

Together with the increased understanding of the genetic mechanisms behind evolution this brought about the opportunity to become active masters of evolutionary processes that are fully designed and executed by human experimenters from above.

It could be argued that evolutionary algorithms are not faithful models of natural evolution. However, they certainly are a form of evolution.

As phrased by Dennett: If you have variation, heredity, and selection, then you must get evolution. In Table 6 we compare natural evolution and artificial evolution as used in contemporary evolutionary algorithms.

Table 6. Differences between natural and artificial evolution.

	Natural evolution	Artificial evolution
Fitness	Observed quantity: *a posteriori* effect of selection ('in the eye of the observer').	Predefined *a priori* quantity that drives selection.
Selection	Complex multifactor force based on environmental conditions, other individuals of the same species and other species (e.g., predators). Viability is tested continually; reproducibility is tested at discrete times.	Randomized operator with selection probabilities based on given fitness values. Parent selection and survivor selection both happen at discrete times.
Genotype-phenotype mapping	Highly complex biochemical process influenced by the environment.	Relatively simple mathematical transformation or parameterised procedure.
Variation	Offspring created from one (asexual reproduction) or two parents (sexual reproduction).	Offspring may be generated from one, two, or many parents.
Execution	Parallel, decentralized execution; birth and death events are not synchronised.	Typically centralized with synchronised birth and death.
Population	Spatial embedding implies structured populations. Population size varies according to the relative number of death and birth events.	Typically unstructured and panmictic (all individuals are potential partners). Population size is kept constant by synchronising time and number of birth and death events.

2.8 EVOLUTIONARY COMPUTING, GLOBAL OPTIMIZATION, AND OTHER SEARCH ALGORITHMS

Evolutionary algorithms are often used for problem optimization. Of course EAs are not the only optimization technique known, so in this section we explain where EAs fall into the general class of optimization methods, and why they are of increasing interest.

In an ideal world, we would possess the technology and algorithms that could provide a provably optimal solution to any problem that we could suitably pose to the system. In fact such algorithms do exist: an exhaustive enumeration of all of

the possible solutions to a problem is clearly such an algorithm. Moreover, for many problems that can be expressed in a suitably mathematical formulation, much faster, exact techniques such as branch and bound search are well known. However, despite the rapid progress in computing technology, and even if there is no halt to Moore's Law, all too often the types of problems posed by users exceed in their demands the capacity of technology to answer them.

Decades of computer science research have taught us that many real-world problems can be reduced in their essence to well-known abstract forms, for which the number of potential solutions grows very quickly with the number of variables considered. For example, many problems in transportation can be reduced to the well-known travelling salesperson problem (TSP): given a list of destinations, construct the shortest tour that visits each destination exactly once. If we have n destinations, with symmetric distances between them, the number of possible tours is $n!/2 = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3$, which is exponential in n. For some of these abstract problems exact methods are known whose time complexity scales linearly (or at least polynomially) with the number of variables. However, it is widely accepted that for many types of problems encountered, no such algorithms exist. Thus, despite the increase in computing power, beyond a certain size of problem we must abandon the search for provably optimal solutions, and look to other methods for finding good solutions.

The term global optimization refers to the process of attempting to find the solution with the optimal value for some fitness function. In mathematical terminology, we are trying to find the solution x^* out of a set of possible solutions S, such that $x = x^* \Rightarrow f(x^*) \geq f(x) \; \forall x \in S$. Here we have assumed a maximization problem – the inequality is simply reversed for minimization.

As noted above, a number of deterministic algorithms exist that, if allowed to run to completion, are guaranteed to find x^*. The simplest example is, of course, complete enumeration of all the solutions in S, which can take an exponentially long time as the number of variables increases. A variety of other techniques,

collectively known as box decomposition, are based on ordering the elements of S into some kind of tree, and then reasoning about the quality of solutions in each branch in order to decide whether to investigate its elements. Although methods such as branch and bound can sometimes make very fast progress, in the worst case (caused by searching in a suboptimal order) the time complexity of the algorithms is still the same as complete enumeration.

Another class of search methods is known as heuristics. These may be thought of as sets of rules for deciding which potential solution out of S should next be generated and tested. For some randomized heuristics, such as simulated annealing and certain variants of EAs, convergence proofs do in fact exist, i.e., they are guaranteed to find x^*. Unfortunately these algorithms are fairly weak, in the sense that they will not identify x^c as being globally optimal, rather as simply the best solution seen so far.

An important class of heuristics is based on the idea of using operators that impose some kind of structure onto the elements of S, such that each point x has associated with it a set of neighbors N(x). In Fig. 2 the variables (traits) x and y were taken to be real-valued, which imposes a natural structure on S. The reader should note that for those types of problem where each variable takes one of a finite set of values (so-called combinatorial optimization), there are many possible neighborhood structures. As an example of how the landscape 'seen' by a local search algorithm depends on its neighborhood structure, the reader might wish to consider what a chessboard would look like if we reordered it, so that squares that are possible next moves for the knight piece were adjacent to each other. Thus points which are locally optimal (fitter than all their neighbors) in the landscape induced by one neighborhood structure may not be for another. However, by its definition, the global optimum x^* will always be fitter than all of its neighbors under any neighborhood structure.

So-called local search algorithms and their many variants work by taking a starting solution x, and then searching the candidate solutions in N(x) for one x' that performs better than x. If such a solution exists, then this is accepted as the new incumbent

solution, and the search proceeds by examining the candidate solutions in N(x ‘). This process will eventually lead to the identification of a local optimum: a solution that is superior to all those in its neighborhood. Such algorithms (often referred to as hill climbers for maximization problems) have been well studied over the decades. They have the advantage that they are often quick to identify a good solution to the problem, which is sometimes all that is required in practical applications. However, the downside is that problems will frequently exhibit numerous local optima, some of which may be significantly worse than the global optimum, and no guarantees can be offered for the quality of solution found.

A number of methods have been proposed to get around this problem by changing the search landscape, either by changing the neighborhood structure, or by temporarily assigning low fitness to already-seen good solutions. However the theoretical basis behind these algorithms is still very much in gestation.There are a number of features of EAs that distinguish them from local search algorithms, relating principally to their use of a population. The population provides the algorithm with a means of defining a non-uniform probability distribution function (p.d.f.) governing the generation of new points from S. This p.d.f. reflects possible interactions between points in S which are currently represented in the population. The interactions arise from the recombination of partial solutions from two or more members of the population (parents). This potentially complex p.d.f. contrasts with the globally uniform distribution of blind random search, and the locally uniform distribution used by many other stochastic algorithms such as simulated annealing and various hill-climbing algorithms. The ability of EAs to maintain a diverse set of points provides not only a means of escaping from local optima, but also a means of coping with large and discontinuous search spaces. In addition, if several copies of a solution can be generated, evaluated, and maintained in the population, this provides a natural and robust way of dealing with problems where there is noise or uncertainty associated with the assignment of a fitness score to a candidate solution.

REFERENCES

1. Ashlock, D. (2006), Evolutionary Computation for Modeling and Optimization, Springer, ISBN 0-387-22196-4.
2. Benko, Attila; Dosa, Gyorgy; Tuza, Zsolt (2010). "Bin Packing/Covering with Delivery, solved with the evolution of algorithms". 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA). pp. 298–302. doi:10.1109/BICTA.2010.5645312. ISBN 978-1-4244-6437-1. S2CID 16875144.
3. Claudio Comis Da Ronco, Ernesto Benini, A Simplex-Crossover-Based Multi-Objective Evolutionary Algorithm, IAENG Transactions on Engineering Technologies, Volume 247 of the series Lecture Notes in Electrical Engineering pp 583-598, 2013 https://link.springer.com/chapter/10.1007%2F978-94-007-6818-5_41
4. Eiben, A.E., Smith, J.E. (2003), Introduction to Evolutionary Computing, Springer.
5. Michalewicz Z., Fogel D.B. (2004). How To Solve It: Modern Heuristics, Springer.
6. Poli, R.; Langdon, W. B.; McPhee, N. F. (2008). A Field Guide to Genetic Programming. Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4. Archived from the original on 2016-05-27. Retrieved 2011-03-05.
7. Price, K., Storn, R.M., Lampinen, J.A., (2005). "Differential Evolution: A Practical Approach to Global Optimization", Springer.
8. Rahman, Rosshairy Abd.; Kendall, Graham; Ramli, Razamin; Jamari, Zainoddin; Ku-Mahamud, Ku Ruhana (2017). "Shrimp Feed Formulation via Evolutionary Algorithm with Power Heuristics for Handling Constraints". Complexity. 2017: 1–12. doi:10.1155/2017/7053710.
9. Simon, D. (2013): Evolutionary Optimization Algorithms, Wiley.

CHAPTER 3

GENETIC ALGORITHM

INTRODUCTION

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation. GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest". In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

3.1 REPRESENTATION OF INDIVIDUALS

Genetic representation is a way of representing solutions/ individuals in evolutionary computation methods. Genetic representation can encode appearance, behaviour, physical qualities of individuals. Designing a good genetic representation that is expressive and evolvable is a hard problem in evolutionary computation. Difference in genetic representations is one of the major criteria drawing a line between known classes of evolutionary computation.

Terminology is often analogous with natural genetics. The block of computer memory that represents one candidate solution is called an individual. The data in that block is called a chromosome. Each chromosome consists of genes. The possible values of a particular gene are called alleles. A programmer may represent all the individuals of a population using binary encoding, permutational encoding, encoding by tree, or any one of several other representations.

Genetic algorithms use linear binary representations. The most standard one is an array of bits. Arrays of other types and structures

can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size. This facilitates simple crossover operation. Variable length representations were also explored in Genetic algorithms, but crossover implementation is more complex in this case.

Evolution strategy uses linear real-valued representations, e.g. an array of real values. It uses mostly gaussian mutation and blending/averaging crossover.

Genetic programming (GP) pioneered tree-like representations and developed genetic operators suitable for such representations. Tree-like representations are used in GP to represent and evolve functional programs with desired properties. Human-based genetic algorithm (HBGA) offers a way to avoid solving hard representation problems by outsourcing all genetic operators to outside agents, in this case, humans. The algorithm has no need for knowledge of a particular fixed genetic representation as long as there are enough external agents capable of handling those representations, allowing for free-form and evolving genetic representations.

3.1.1 Data Representation

The organization of any computer depends considerably on how it represents numbers, characters, and control information. The converse is also true: Standards and conventions established over the years have determined certain aspects of computer organization.

Data Type

A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error. A string, for example, is a data type that is used to classify text and an integer is a data type used to classify whole numbers.

Data Type	Used for	Example
String	Alphanumeric characters	hello world, Alice, Bob123
Integer	Whole numbers	7, 12, 999
Float (floating point)	Number with a decimal point	3.15, 9.06, 00.13
Character	Encoding text numerically	97 (in ASCII, 97 is a lower case 'a')
Boolean	Representing logical values	TRUE, FALSE

The data type defines which operations can safely be performed to create, transform and use the variable in another computation. When a program language requires a variable to only be used in ways that respect its data type, that language is said to be strongly typed. This prevents errors, because while it is logical to ask the computer to multiply a float by an integer (1.5 x 5), it is illogical to ask the computer to multiply a float by a string (1.5 x Alice). When a programming language allows a variable of one data type to be used as if it were a value of another data type, the language is said to be weakly typed.

Technically, the concept of a strongly typed or weakly typed programming language is a fallacy. In every programming language, all values of a variable have a static type but the type might be one whose values are classified into one or more classes. And while some classes specify how the data type's value will be compiled or interpreted, there are other classes whose values are not marked with their class until run-time. The extent to which a programming language discourages or prevents type error is known as type safety.

Fixed-Point Representation

- Positive integers and zero can be represented by unsigned numbers

- Negative numbers must be represented by signed numbers since + and – signs are not available, only 1's and 0's are
- Signed numbers have msb as 0 for positive and 1 for negative – msb is the sign bit
- Two ways to designate binary point position in a register
  - Fixed point position
  - Floating-point representation
- Fixed point position usually uses one of the two following positions
  - A binary point in the extreme left of the register to make it a fraction
  - A binary point in the extreme right of the register to make it an integer
  - In both cases, a binary point is not actually present
- The floating-point representations uses a second register to designate the position of the binary point in the first register
- When an integer is positive, the msb, or sign bit, is 0 and the remaining bits represent the magnitude
- When an integer is negative, the msb, or sign bit, is 1, but the rest of the number can be represented in one of three ways
  - Signed-magnitude representation
  - Signed-1's complement representation
  - Signed-2's complement representation
- Consider an 8-bit register and the number +14
  - The only way to represent it is 00001110
- Consider an 8-bit register and the number –14
  - Signed magnitude: 1 0001110
  - Signed 1's complement: 1 1110001
  - Signed 2's complement: 1 1110010

- Typically use signed 2′s complement
- Addition of two signed-magnitude numbers follow the normal rules
  - If same signs, add the two magnitudes and use the common sign
  - Differing signs, subtract the smaller from the larger and use the sign of the larger magnitude
  - Must compare the signs and magnitudes and then either add or subtract
- Addition of two signed 2′s complement numbers does not require a comparison or subtraction – only addition and complementation
  - Add the two numbers, including their sign bits
  - Discard any carry out of the sign bit position
  - All negative numbers must be in the 2's complement form
  - If the sum obtained is negative, then it is in 2's complement form

+6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

- Subtraction of two signed 2′s complement numbers is as follows
- Take the 2′s complement form of the subtrahend (including sign bit)
- Add it to the minuend (including the sign bit)
- A carry out of the sign bit position is discarded
- An overflow occurs when two numbers of n digits each are added and the sum occupies n + 1 digits
- Overflows are problems since the width of a register is finite
- Therefore, a flag is set if this occurs and can be checked

by the user

- Detection of an overflow depends on if the numbers are signed or unsigned
- For unsigned numbers, an overflow is detected from the end carry out of the msb
- For addition of signed numbers, an overflow cannot occur if one is positive and one is negative – both have to have the same sign
- An overflow can be detected if the carry into the sign bit position and the carry out of the sign bit position are not equal

+70	0	1000110	-70	1	0111010
+80	0	1010000	-80	1	0110000
+150	1	0010110	-150	0	1101010

- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit
- A 4-bit decimal code requires four flip-flops for each decimal digit
- This takes much more space than the equivalent binary representation and the circuits required to perform decimal arithmetic are more complex
- Representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary
- Either signed magnitude or signed complement systems
- The sign of a number is represented with four bits
  - 0000 for +
  - 1001 for –
- To obtain the 10's complement of a BCD number, first take the 9's complement and then add one to the least significant digit
- Example: (+375) + (-240) = +135

0	375	$(0000\ 0011\ 0111\ 1010)_{BCD}$
+9	760	$(1001\ 0111\ 0110\ 0000)_{BCD}$
0	135	$(0000\ 0001\ 0011\ 0101)_{BCD}$

3.1.2 Floating-Point Representation

If we wanted to build a real computer, we could use any of the integer representations that we just studied. We would pick one of them and proceed with our design tasks. Our next step would be to decide the word size of our system. If we want our system to be really inexpensive, we would pick a small word size, say 16 bits. Allowing for the sign bit, the largest integer that this system can store is 32,767. So now what do we do to accommodate a potential customer who wants to keep a tally of the number of spectators paying admission to professional sports events in a given year? Certainly, the number is larger than 32,767. No problem. Let's just make the word size larger. Thirty-two bits ought to do it. Our word is now big enough for just about anything that anyone wants to count. But what if this customer also needs to know the amount of money each spectator spends per minute of playing time? This number is likely to be a decimal fraction. Now we're really stuck.

The easiest and cheapest approach to this problem is to keep our 16-bit system and say, "Hey, we're building a cheap system here. If you want to do fancy things with it, get yourself a good programmer." Although this position sounds outrageously flippant in the context of today's technology, it was a reality in the earliest days of each generation of computers. There simply was no such thing as a floating-point unit in many of the first mainframes or microcomputers. For many years, clever programming enabled these integer systems to act as if they were, in fact, floating-point systems.

If you are familiar with scientific notation, you may already be thinking of how you could handle floating-point operations—how you could provide floating-point emulation—in an integer system. In scientific notation, numbers are expressed in two parts: a fractional part, called a mantissa, and an exponential part that indicates the power of ten to which the mantissa should be raised to obtain the value we need. So to express 32,767 in scientific notation, we could write 3.2767×10^4. Scientific notation simplifies pencil and paper calculations that involve very large or very small

numbers. It is also the basis for floating-point computation in today's digital computers.

A Simple Model

In digital computers, floating-point numbers consist of three parts: a sign bit, an exponent part (representing the exponent on a power of 2), and a fractional part called a significand (which is a fancy word for a mantissa). The number of bits used for the exponent and significand depends on whether we would like to optimize for range (more bits in the exponent) or precision (more bits in the significand). We will use a 14-bit model with a 5-bit exponent, an 8-bit significand, and a sign bit (see Figure 1).

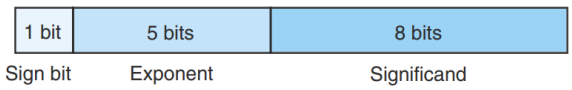
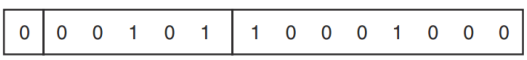


Figure 1: Floating-Point Representation.

Let's say that we wish to store the decimal number 17 in our model. We know that $17 = 17.0 \times 10^0 = 1.7 \times 10^1 = 0.17 \times 10^2$. Analogously, in binary, $17_{10} = 10001_2 \times 2^0 = 1000.1_2 \times 2^1 = 100.01_2 \times 2^2 = 10.001_2 \times 2^3 = 1.0001_2 \times 2^4 = 0.10001_2 \times 2^5$. If we use this last form, our fractional part will be 10001000 and our exponent will be 00101, as shown here:



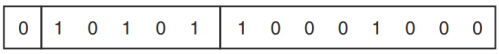
Using this form, we can store numbers of much greater magnitude than we could using a fixed-point representation of 14 bits (which uses a total of 14 binary digits plus a binary, or radix, point). If we want to represent $65536 = 0.1_2 \times 21^7$ in this model, we have:

One obvious problem with this model is that we haven't provided for negative exponents. If we wanted to store 0.25 we would have no way of doing so because 0.25 is 2^{-2} and the exponent –2 cannot be represented. We could fix the problem by adding a sign bit to the exponent, but it turns out that it is more efficient to use

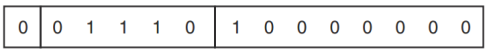
a biased exponent, because we can use simpler integer circuits when comparing the values of two floating-point numbers.

The idea behind using a bias value is to convert every integer in the range into a non-negative integer, which is then stored as a binary numeral. The integers in the desired range of exponents are first adjusted by adding this fixed bias value to each exponent. The bias value is a number near the middle of the range of possible values that we select to represent zero. In this case, we could select 16 because it is midway between 0 and 31 (our exponent has 5 bits, thus allowing for 2^5 or 32 values). Any number larger than 16 in the exponent field will represent a positive value. Values less than 16 will indicate negative values. This is called an excess-16 representation because we have to subtract 16 to get the true value of the exponent. Note that exponents of all zeros or all ones are typically reserved for special numbers (such as zero or infinity).

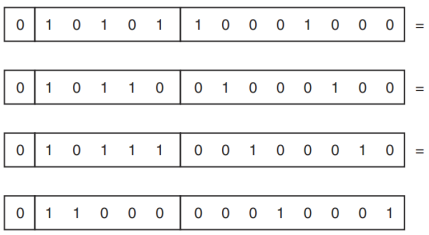
Returning to our example of storing 17, we calculated $17_{10} = 0.10001_2 \times 2^5$. The biased exponent is now 16 + 5 = 21:



If we wanted to store $0.25 = 1.0 \times 2^{-2}$ we would have:



There is still one rather large problem with this system: We do not have a unique representation for each number. All of the following are equivalent:



Because synonymous forms such as these are not well-suited for digital computers, a convention has been established where the leftmost bit of the significand will always be a 1. This is called

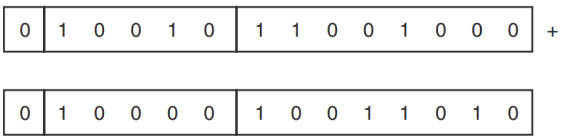
normalization. This convention has the additional advantage in that the 1 can be implied, effectively giving an extra bit of precision in the significand.

Floating-Point Arithmetic

If we wanted to add two decimal numbers that are expressed in scientific notation, such as $1.5 \times 10^2 + 3.5 \times 10^3$, we would change one of the numbers so that both of them are expressed in the same power of the base. In our example, $1.5 \times 10^2 + 3.5 \times 10^3 = 0.15 \times 10^3 + 3.5 \times 10^3 = 3.65 \times 10^3$. Floating-point addition and subtraction work the same way.

Example

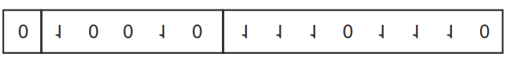
Add the following binary numbers as represented in a normalized 14-bit format with a bias of 16.



We see that the addend is raised to the second power and that the augend is to the zero power. Alignment of these two operands on the binary point gives us:

$$\begin{array}{r} 11.001000 \\ +\ 0.10011010 \\ \hline 11.10111010 \end{array}$$

Renormalizing, we retain the larger exponent and truncate the low-order bit. Thus, we have:



Multiplication and division are carried out using the same rules of exponents applied to decimal arithmetic, such as $2^{-3} \times 2^4 = 2^1$.

Floating-Point Errors

When we use pencil and paper to solve a trigonometry problem or compute the interest on an investment, we intuitively understand that we are working in the system of real numbers. We know that this system is infinite, because given any pair of real numbers, we can always find another real number that is smaller than one and greater than the other.

Unlike the mathematics in our imaginations, computers are finite systems, with finite storage. When we call upon our computers to carry out floating-point calculations, we are modeling the infinite system of real numbers in a finite system of integers. What we have, in truth, is an approximation of the real number system. The more bits we use, the better the approximation. However, there is always some element of error, no matter how many bits we use. Floating-point errors can be blatant, subtle, or unnoticed. The blatant errors, such as numeric overflow or underflow, are the ones that cause programs to crash. Subtle errors can lead to wildly erroneous results that are often hard to detect before they cause real problems. For example, in our simple model, we can express normalized numbers in the range of $-.11111111_2 \times 2^{15}$ through $+.11111111 \times 2^{15}$. Obviously, we cannot store 2^{-19} or 2^{128}; they simply don't fit. It is not quite so obvious that we cannot accurately store 128.5, which is well within our range. Converting 128.5 to binary, we have 10000000.1, which is 9 bits wide. Our significand can hold only eight. Typically, the low-order bit is dropped or rounded into the next bit. No matter how we handle it, however, we have introduced an error into our system.

We can compute the relative error in our representation by taking the ratio of the absolute value of the error to the true value of the number. Using our example of 128.5, we find:

$$\frac{128.5 - 128}{128} = 0.003906 \approx 0.39\%.$$

If we are not careful, such errors can propagate through a lengthy calculation, causing substantial loss of precision. Figure

2 illustrates the error propagation as we iteratively multiply 16.24 by 0.91 using our 14-bit model. Upon converting these numbers to 8-bit binary, we see that we have a substantial error from the outset.

As you can see, in six iterations, we have more than tripled the error in the product. Continued iterations will produce an error of 100% because the product eventually goes to zero. Although this 14-bit model is so small that it exaggerates the error, all floating-point systems behave the same way. There is always some degree of error involved when representing real numbers in a finite system, no matter how large we make that system. Even the smallest error can have catastrophic results, particularly when computers are used to control physical events such as in military and medical applications. The challenge to computer scientists is to find efficient algorithms for controlling such errors within the bounds of performance and economics.

Multiplier		Multiplicand	14-Bit Product	Real Product	Error
1000.001 (16.125)	×	0.11101000 = (0.90625)	1110.1001 (14.5625)	14.7784	1.46%
1110.1001 (14.5625)	×	0.11101000 =	1101.0011 (13.1885)	13.4483	1.94%
1101.0011 (13.1885)	×	0.11101000 =	1011.1111 (11.9375)	12.2380	2.46%
1011.1111 (11.9375)	×	0.11101000 =	1010.1101 (10.8125)	11.1366	2.91%
1010.1101 (10.8125)	×	0.11101000 =	1001.1100 (9.75)	10.1343	3.79%
1001.1100 (9.75)	×	0.11101000 =	1000.1101 (8.8125)	8.3922	4.44%

Figure 2 Error Propagation in a 14-Bit Floating-Point Number.

3.1.3 Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is parity check.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

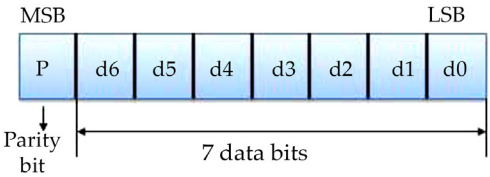
How to Detect and Correct Errors?

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

- The additional bits are called parity bits. They allow detection or correction of the errors.
- The data bits along with the parity bits form a code word.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).

Use of Parity Bit

The parity bit can be set to 0 and 1 depending on the type of the parity required.

- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in figure (3).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in figure (4).

Figure 3:

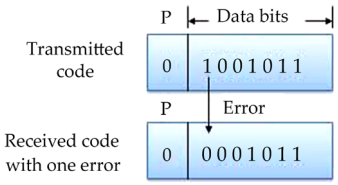
Figure 4:

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected

parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct.

If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



3.2 NUMBER REPRESENTATION AND BINARY CODE

Binary is a number system which builds numbers from elements called bits. Each bit can be represented by any two mutually exclusive states.

Generally, when one write it down or code bits, and represent them with 1 and 0 to build binary numbers the same way and build numbers in the traditional base 10 system. However, instead of a one's column, a 10's column, and a 100 column (and so on) and have a one's column, a two's columns, a four's column, an eight's column, and so on, as illustrated:

Table 1. Binary

$2^{\cdots}$	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	64	32	16	8	4	2	1

For example, to represent the number 203 in base 10, place a 3 in the 1's column, a 0 in the 10's column and a 2 in the 100's column.

This is expressed with exponents in the Table 2.

Table 2. 203 in base 10

10^2	10^1	10^0
2	0	3

Or, in other words, 2 Ã 10^2 + 3 Ã 10^0 = 200 + 3 = 203.

To represent the same thing in binary, and would have in the Table 3.

Table 3. 203 in base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	1	0	1	1

That equates to 2^7 + 2^6 + 2^3+2^1 + 2^0 = 128 + 64 + 8 + 2 + 1 = 203.

Conversion

The easiest way to convert between bases is to use a computer; after all, that's what they are good at! However, it is often useful to know how to do conversions by hand.

The easiest method to convert between bases is repeated division. To convert, repeatedly divide the quotient by the base, until the quotient is zero, making note of the remainders at each step. Then, write the remainders in reverse, starting at the bottom and appending to the right each time. An *example* should illustrate; since you are converting to binary and use a base of 2.

Table 4. Convert 203 to binary

Quotient	**Â**	**Remainder**	**Â**
203_{10} Ã· 2 =	101	1	Â
101_{10} Ã· 2 =	50	1	â†‘
50_{10} Ã· 2 =	25	0	â†‑

25_{10} Ã· 2 =	12	1	â†‘
12_{10} Ã· 2 =	6	0	â†‘
6_{10} Ã· 2 =	3	0	â†‘
3_{10} Ã· 2 =	1	1	â†‘
1_{10} Ã· 2 =	0	1	â†‘

Reading from the bottom and appending to the right each time gives 11001011, which anyone saw from the example was 203.ss

Bits and Bytes

To represent all the letters of the alphabet you would need at least enough different combinations to represent all the lower case letters, the upper case letters, numbers and punctuation, plus a few extras. Adding this up means need probably around 80 different combinations.

- If you have two bits and can represent 4 unique combinations (00 01 10 11).
- If you have three bits and can represent 8 different combinations.
- With n bits you can represent 2^n unique combinations.

8 bits gives us $2^8 = 256$ unique representations, more than enough for the alphabet combinations. And call a group of 8 bits a *byte*. Guess how bit a C char variable is?

ASCII

Given that a byte can represent any of the values 0 through 256, anyone could arbitrarily make up a mapping between characters and numbers. *For example*, a video card manufacturer could decide that the value 10 represents A, so when value 10 is sent to the video card it displays a capital 'A' on the screen.

To avoid this happening, the *American Standard Code for Information Interchange* or ASCII was invented. This is a *7-bit* code, meaning there are 2^7 or 128 available codes.

The range of codes is divided up into two major parts; the non-printable and the printable. Printable characters are things like characters (supper and lower case), numbers and punctuation. Nonprintable codes are for control, and do things like make a carriage-return, ring the terminal bell or the special `NULL` code which represents nothing at all.

127 unique characters is sufficient for American English, but becomes very restrictive when someone wants to represent characters common in other languages, especially Asian languages which can have many thousands of unique characters.

To alleviate this, modern systems are moving away from ASCII to *Unicode*, which can use up to 4 bytes to represent a character, giving *much* more room!

Parity

ASCII, being only a 7-bit code, leaves one bit of the byte spare. This can be used to implement *parity* which is a simple form of error checking. Consider a computer using punch-cards for input, where a hole represents 1 and no hole represents 0. Any inadvertent covering of a hole will cause an incorrect value to be read, causing undefined behavior.

Parity allows a simple check of the bits of a byte to ensure they were read correctly, and can implement either *odd* or *even* parity by using the extra bit as a *parity bit*.

In odd parity, if the number of 1's in the 7 bits of information is odd, the parity bit is set, otherwise it is not set. Even parity is the opposite; if the number of 1's is even the parity bit is set to 1. In this way, the flipping of one bit will case a parity error, which can be detected.

16, 32 and 64 bit Computers

Numbers do not fit into bytes; hopefully bank balance in dollars will need more range than can fit into one byte! Most modern architectures are *32 bit* computers. This means they work with 4 bytes at a time when processing and reading or writing to memory; and refers to 4 bytes as a *word*; this is analogous to language where letters (bits) make up words in a sentence, except in computing every word has the same size! The size of a C `it` variable is 32 bits. Newer architectures are 64 bits, which doubles the size the processor works with (8 bytes).

Kilo, Mega and Giga Bytes

Computers deal with a lot of bytes; that's what makes them so powerful!

One need a way to talk about large numbers of bytes, and a natural way is to use the "International System of Units" (SI) prefixes as used in most other scientific areas. So *for example*, kilo refers to 10^3 or 1,000 units, as in a kilogram has 1,000 grams.

1,000 is a nice round number in base 10, but in binary it is `1111101000` which is not a particularly "round" number. However, 1024 (or 2^{10}) is (`10000000000`), and happens to be quite close to the base ten meaning of kilo (1000 as opposed to 1024).

Hence 1024 bytes became known as a *kilobyte*. The first mass market computer was the Commodore 64, so named because it had 64 kilobytes of storage.

Today, kilobytes of memory would be small for a wrist watch, let alone a personal computer. The next SI unit is "mega" for 10^6. As it happens, 2^{20} is again close to the SI base 10 definition; 1048576 as opposed to 1000000.

The units keep increasing by powers of 10; each time it diverges from the base SI meaning.

Table 5 Bytes

2^{10}	Kilobyte
2^{20}	Megabyte
2^{30}	Gigabyte
2^{40}	Terabyte
2^{50}	Petabyte
2^{60}	Exabyte

Therefore a 32 bit computer can address up to four gigabytes of memory; the extra two bits can represent four groups of 2^{30} `bytes`. A 64 bit computer can address up to 8 Exabyte's; you might be interested in working out just how big a number this is. To get a feel for how bit that number is calculate how long it would take to count to 2^{64}; if you incremented once per second.

Boolean Operations

George Boole was a mathematician who discovered a whole area of mathematics called *Boolean algebra*. Whilst he made his discoveries in the mid 1800's, his mathematics is the fundamentals of all computer science.

Boolean operations simply take a particular input and produce a particular output follows a rule.

For example, the simplest Boolean operation, `not` simply inverts the value of the input operand. Other operands usually take two inputs, and produce a single output.

The fundamental Boolean operations used in computer science are easy to remember and listed. These represent them with *truth tables*; they simply show all possible inputs and outputs. The term *true* simply reflects `1` in binary.

NOT

Usually represented by (`!`), *NOT* simply inverts the value, so `0` becomes `1` and `1` becomes `0`

Table 6: Truth table for not

Input	Output
1	0
0	1

AND

To remember how the AND operation works think of it as "if one input true and the other are true, result is true.

Table 7: Truth table for and

Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR

To remember how the OR operation works think of it as "if one input or the other input is true, the result is true.

Table 8: Truth table for OR

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1

Exclusive OR (XOR)

Exclusive OR, written as XOR is a special case of or where the output is true if one, and only one, of the inputs is true. This operation can surprisingly do many interesting tricks, but one will not see a lot of it in the kernel.

Table 9: Truth table for XOR

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	0

3.2.1 How Computers use Boolean Operations

Believe it or not, essentially everything computer does comes back to the operations.

For example, the half adder is a type of circuit made up from Boolean operations that can add bits together (it is called a half adder because it does not handle carry bits). Put more half adders together, and will start to build something that can add together long binary numbers. Add some external memory, and has a computer.

Electronically, the Boolean operations are implemented in *gates* made by *transistors*.

This is why you might have heard about transistor counts and things like Moores Law. The more transistors, the more gates, the more things and can add together. To create the modern computer, there are an awful lot of gates, and an awful lot of transistors. Some of the latest Itanium processors have around 460 million transistors.

Hexadecimal

Hexadecimal refers to a base 16 number system. Uses this in computer science for only one reason; it makes it easy for humans to think about binary numbers. Computers only ever deal in binary and hexadecimal is simply a shortcut for us humans trying to work with the computer.

So why base 16? Well, the most natural choice is base 10, since anyone is used to thinking in base 10 from everyday number system. But base 10 does not work well with binary to represent 10 different elements in binary, and need four bits. Four bits, however, gives us sixteen possible combinations. So they can either take the very tricky road of trying to convert between base 10 and binary, or take the easy road and make up a base 16 number system - hexadecimal!

Hexadecimal uses the standard base 10 numerals, but adds `A B C D E F` which refer to `10 11 12 13 14 15` (one start from zero).

Traditionally, any time you sees a number prefixed by `0x` this will denote a hexadecimal number.

As mentioned, to represent 16 different patterns in binary, and would need exactly four bits. Therefore, each hexadecimal numeral represents exactly four bits.

One should consider it an exercise to learn the given Table off by heart.

Table 10: Hexadecimal, binary and decimal

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2

3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Of course there is no reason not to continue the pattern (say, assign G to the value 16), but 16 values is an excellent tradeoff between the vagaries of human memory and the number of bits used by a computer. Anyone simply represents larger numbers of bits with more numerals. *For example*, a sixteen bit variable can be represented by 0xAB12, and to find it in binary simply take each individual numeral, convert it as per the table and join them all together (so 0xAB12 ends up as the 16-bit binary number 1010101100010010). And can use the reverse to convert from binary back to hexadecimal.

One can also use the same repeated division scheme to change the base of a number.

For example, to find 203 in hexadecimal.

Table 11: Convert 203 to hexadecimal

Quotient	**Â**	**Remainder**	**Â**
203_{10} Ã· 16 =	12	11 (0xB)	Â
12_{10} Ã· 16 =	0	12 (0xC)	â†‘

Hence 203 in hexadecimal is 0xCB.

Practical Implications

Use of Binary in Code

Whilst binary is the underlying language of every computer, it is entirely practical to program a computer in high level languages without knowing the first thing about it. However, for the low level codes and they are interested in a few fundamental binary principles are used repeatedly.

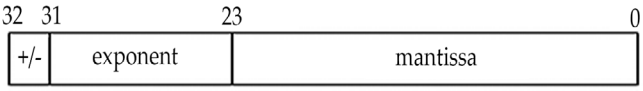
3.2.2 Fixed- and Floating-Point Number Representation

The Institute of Electrical and Electronics Engineers (IEEE) standardizes floating-point representation in IEEE 754. Floating-point representation is similar to scientific notation in that there is a number multiplied by a base number raised to some power. *For example*, 118.625 are represented in scientific notation as 1.18625×10^2. The main benefit of this representation is that it provides varying degrees of precision based on the scale of the numbers that anyone is using. *For example*, it is beneficial to talk in terms of angstroms (10^{-10} m) when one is working with the distance between atoms. However, if they are dealing with the distance between cities, this level of precision is no longer practical or necessary.

IEEE 754 defines binary representations for 32-bit single-precision and 64-bit double-precision (64-bit) numbers as well as extended

single-precision and extended double-precision numbers. Examine the specification for single-precision, floating-point numbers, also called floats.

A float consists of three parts: the sign bit, the exponent, and the mantissa. The division of the three parts is as follows:



The sign bit is 0 if the number is positive and 1 if the number is negative.

The exponent is an 8-bit number that ranges in value from -126 to 127. The exponent is actually not the typical two's complement representation because this makes comparisons more difficult. Instead, the value is biased by adding 127 to the desired exponent and representation, which makes it possible to represent negative numbers. The mantissa is the normalized binary representation of the number to be multiplied by 2 raised to the power defined by the exponent.

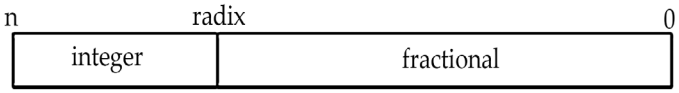
Now look at how to encode 118.625 as a float. The number 118.625 is a positive number, so the sign bit is 0. To find the exponent and mantissa, first write the number in binary, which are 1110110.101. Next, normalize the number to 1.110110101 × 26, which is the binary equivalent of scientific notation. The exponent is 6 and the mantissa is 1.110110101. The exponent must be biased, which is 6 + 127 = 133. The binary representation of 133 is 10000101.

Thus, the floating-point encoded value of 118.65 is 0100 0010 1111 0110 1010 0000 0000 0000. Binary values are often referred to in their hexadecimal equivalent. In this case, the hexadecimal value is 42F6A000.

Fixed Point Representation

In fixed-point representation, a specific radix point called a decimal point in English and written "." is chosen so there is a fixed number of bits to the right and a fixed number of bits to the

left of the radix point. The bits to the left of the radix point are called the integer bits. The bits to the right of the radix point are called the fractional bits.



In this example, assume a 16-bit fractional number with 8 magnitude bits and 8 radix bits, which is typically represented as 8.8 representations. Like most signed integers, fixed-point numbers are represented in two's complement binary. Using a positive number keeps this example simple.

To encode 118.625, first find the value of the integer bits. The binary representation of 118 is 01110110, so this is the upper 8 bits of the 16-bit number. The fractional part of the number is represented as 0.625×2^n where n is the number of fractional bits. Because $0.625 \times 256 = 160$, and use the binary representation of 160, which is 10100000, to determine the fractional bits. Thus, the binary representation for 118.625 is 0111 0110 1010 0000. The value is typically referred to using the hexadecimal equivalent, which is 76A0.

The major advantage of using fixed-point representation for real numbers is that fixed-point adheres to the same basic arithmetic principles as integers. Therefore, fixed-point numbers can take advantage of the general optimizations made to the Arithmetic Logic Unit (ALU) of most microprocessors, and do not require any additional libraries or any additional hardware logic. On processors without a floating-point unit (FPU), such as the Analog Devices Black fin Processor, fixed-point representation can result in much more efficient embedded code when performing mathematically heavy operations.

In general, the disadvantage of using fixed-point numbers is that fixed-point numbers can represent only a limited range of values, so fixed-point numbers are susceptible to common numeric computational inaccuracies. *For example*, the range of possible values in the 8.8 notation that can be represented is +127.99609375

to -128.0. If you add 100 + 100 and exceed the valid range of the data type which is called overflow. In most cases, the values that overflow are saturated, or truncated, so that the result is the largest representable number.

3.2.3 BCD

In computing and electronic systems, binary-coded decimal (BCD) or, in its most common modern implementation, packed decimal, is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display, and allows faster decimal calculations. Its drawbacks are a small increase in the complexity of circuits needed to implement mathematical operations. Uncompressed BCD is also a relatively inefficient encoding it occupies more space than a purely binary representation.

Short for Binary Coded Decimal, BCD is also known as packet decimal and is numbers 0 through 9 converted to four-digit binary. There is a list of the decimal numbers 0 through 9 and the binary conversion.

In BCD, a digit is usually represented by four bits which, in general, represent the decimal digits 0 through 9. Other bit combinations are sometimes used for a sign or for other indications (e.g., error or overflow).

Although uncompressed BCD is not as widely used as it once was, decimal fixed-point and floating-point are still important and continue to be used in financial, commercial, and industrial computing.

Recent decimal floating-point representations use base-10 exponents, but not BCD encodings. Current hardware implementations, however, convert the compressed decimal encodings to BCD internally before carrying out computations. Software implementations of decimal arithmetic typically use BCD or some other 10^n base, depending on the operation.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Using this conversion, the number 25, *for example*, would have a BCD number of 0010 0101 or 00100101. However, in binary, 25 are represented as 11001.Number larger than 9 having two or more digits in the decimal system, are expressed digit by digit. *For example*, the BCD rendition of the base-10 number 1895 is

0001 1000 1001 0101

The binary equivalents of 1, 8, 9, and 5, always in a four-digit format, go from left to right.

The BCD representation of a number is not the same, in general, as its simple binary representation. In binary form, *for example*, the decimal quantity 1895 appears as

11101100111

Other bit patterns are sometimes used in BCD format to represent special characters relevant to a particular system, such as sign (positive or negative), error condition, or overflow condition.

The BCD system offers relative ease of conversion between machine-readable and human-readable numerals. As compared to the simple binary system, however, BCD increases the circuit

complexity. The BCD system is not as widely used today as it was a few decades ago, although some systems still employ BCD in financial applications.

3.2.4 EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is a character encoding set used by IBM mainframes. Unlike virtually every computer system in the world which uses a variant of ASCII, IBM mainframes and midrange systems such as the AS/400 tend to use a wholly incompatible character set primarily designed for ease of use on punched cards. (For an excellent page on punched cards, see Doug Jones's Punched Card Codes).EBCDIC uses the full 8 bits available to it, so parity checking cannot be used on an 8 bit system. Also, EBCDIC has a wider range of control characters than ASCII.

The character encoding is based on Binary Coded Decimal (BCD), so the contiguous characters in the alphanumeric range are formed up in blocks of up to 10 from 0000 binary to 1001 binary. Non alphanumeric characters are almost all outside the BCD range. There are four main blocks in the EBCDIC code page: 0000 0000 to 0011 1111 is reserved for control characters; 0100 0000 to 0111 1111 is for punctuation; 1000 0000 to 1011 1111 for lowercase characters and 1100 0000 to 1111 1111 for uppercase characters and numbers.

3.2.5 ASCII

ASCII is the American Standard Code for Information Interchange, also known as ANSI X3.4. There are many variants of this standard, typically to allow different code pages for language encoding, but they all basically follow the same format. ASCII is quite elegant in the way it represents characters, and it is very easy to write code to manipulate upper/lowercase and check for valid data ranges.ASCII is essentially a 7-bit code which allows the 8th most significant bit (MSB) to be used for error checking, however most modern computer systems tend to use ASCII values of 128 for extended character sets.

Table 12: Table of ASCII characters

binary	MSN	0000		0001		0010		0011		0100		0101		0110		0111	
LSN	hex	*0*		*1*		*2*		*3*		*4*		*5*		*6*		*7*	
0000	*0*	**NUL**	000	**DLE**	16 10	**SP**	32 20	**0**	48 30	**@**	64 40	**P**	80 50	**`**	96 60	**p**	112 70
0001	*1*	**SOH**	1 01	**XON(DC1)**	17 11	**!**	33 21	**1**	49 31	**A**	65 41	**Q**	81 51	**a**	97 61	**q**	113 71
0010	*2*	**STX**	2 02	**DC2**	18 12	**“**	34 22	**2**	50 32	**B**	66 42	**R**	82 52	**b**	98 62	**r**	114 72
0011	*3*	**ETX**	3 03	**XOFF (DC2)**	19 13	**#**	35 23	**3**	51 33	**C**	67 43	**S**	83 53	**c**	99 63	**s**	115 73
0100	*4*	**EOT**	4 04	**DC4**	20 14	**$**	36 24	**4**	52 34	**D**	68 44	**T**	84 54	**d**	100 64	**t**	116 74
0101	*5*	**ENQ**	5 05	**NAK**	21 15	**%**	37 25	**5**	53 35	**E**	69 45	**U**	85 55	**e**	101 65	**u**	117 75
0110	*6*	**ACK**	6 06	**SYN**	22 16	**&**	38 26	**6**	54 36	**F**	70 46	**V**	86 56	**f**	102 66	**v**	118 76

0111	*7*	**BEL**	7 07	**ETB**	23 17	‘	39 27	**7**	55 37	**G**	71 47	**W**	87 57	**g**	103 67	**w**	119 77
1000	*8*	**BS**	8 08	**CAN**	24 18	**(**	40 28	**8**	56 38	**H**	72 48	**X**	88 58	**h**	104 68	**x**	120 78
1001	*9*	**HT**	9 09	**EM**	25 19	**)**	41 29	**9**	57 39	**I**	73 49	**Y**	89 59	**i**	105 69	**y**	121 79
1010	*A*	**LF**	10 0A	**SUB**	26 1A	*	42 2A	**:**	58 3A	**J**	74 4A	**Z**	90 5A	**j**	106 6A	**z**	122 7A
1011	*B*	**VT**	11 0B	**ESC**	27 1B	+	43 2B	**;**	59 3B	**K**	75 4B	**[**	91 5B	**k**	107 6B	**{**	123 7B
1100	*C*	**FF**	12 0C	**FS**	28 1C	,	44 2C	**<**	60 3C	**L**	76 4C	\	92 5C	**l**	108 6C	|	124 7C
1101	*D*	**CR**	13 0D	**GS**	29 1D	**-**	45 2D	**=**	61 3D	**M**	77 4D	**]**	93 5D	**m**	109 6D	**}**	125 7D
1110	*E*	**SO**	14 0E	**RS**	30 1E	**.**	46 2E	**>**	62 3E	**N**	78 4E	**^**	94 5E	**n**	110 6E	~	126 7E
1111	*F*	**SI**	150F	**US**	311F	/	472F	**?**	633F	**O**	794F	_	955F	**o**	1116F	**DEL**	1277F

Excess-3

Excess-3 code is an example of un-weighted code. Excess-3 equivalent of a decimal number is obtained by adding 3 and then converting it to a binary format. For instance to find excess-3 representation of decimal number 4, first 3 is added to 4 to get 7 and then binary equivalent of 7 i.e. 0111 forms the excess-3 equivalent. The truth table of excess-3 is given in Table 13.

Table 13: Truth table of excess-3

Truth Table							
Input (BCD)				Output (Excess-3)			
A	**B**	**C**	**D**	**W**	**X**	**Y**	**Z**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Here is a Table representing excess-3 equivalent of decimal numbers (0-9):

Table 14: Excess-3 equivalent of decimal numbers

Decimal Number	Excess-3 Equivalent
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Excess 3 Code Additions

The operation of addition can be done by very simple method you will illustrate the operation in a simple way using steps.

- *Step 1:* You have to convert the numbers (which are to be added) into excess 3 forms by adding 0011 with each of the four bit groups them or simply increasing them by 3.
- *Step 2:* Now the two numbers are added using the basic laws of binary addition, there is no exception for this method.
- *Step 3:* Now which of the four groups have produced a carry one has to add 0011 with them and subtract 0011 from the groups which have not produced a carry during the addition.
- *Step 4:* The result which you have obtained after this operation is in Excess 3 form and this is desired result

Example

To understand the Excess 3 Code Addition method better you can observe the method with the help of an example,

Let us take two numbers which one will to add.

0011 0101 0110 and 0101 0111 1001 are the two binary numbers. Now follows the first step one take the excess 3 form of these two numbers which are 0110 1000 1001 and 1000 1010 1100, now these numbers are added the basic rules of addition.

0110 1000 1001

1000 1010 1100

1111 0011 0101

Now adding 0011 to the groups which produces a carry and subtracting zero from the groups which did not produced carry one get the result as 1100 0110 1000 is the result of the addition in excess 3 code and the BCD answer is 1001 0011 0101.

Excess 3 Code Subtractions

Similarly binary subtraction can be performed by Excess 3 Code Subtraction method. The operation is illustrated with the help of some steps.

- *Step 1:* The numbers have to be converted into excess 3 codes.
- *Step 2:* Following the basic methods of binary subtraction, subtraction is done.
- *Step 3:* Subtract '0011' from each BCD four-bit group in the answer if the subtraction operation of the relevant four-bit groups required borrow from the next higher adjacent four-bit group.
- *Step 4:* Add '0011' to the remaining four-bit groups, if any, in the result.

- *Step 5:* Finally one gets the desired result in excess 3 codes.

Example

Again an example will make the understanding very easy for us.

Let us take the numbers

0001 1000 0101 and 0000 0000 1000 now the excess 3 equivalent of those numbers are 0100 1011 1000 and 0011 0011 1011

Now performing the operation of binary subtraction one get

$$\begin{array}{r} 0100\ 1011\ 1000 \\ 0011\ 0011\ 1011 \\ \hline 0001\ 0111\ 1101 \end{array}$$

The least significant column which needed borrow and the other two columns did not need borrow. Now you have to subtract 0011 from the result of this column and add 0011 to the other two columns, and get 0100 1010 1010. This is the result expressed in excess 3 codes. And the binary result is 0001 0111 0111

3.2.6 IEEE Standard

Abbreviation of Institute of Electrical and Electronics Engineers, pronounced I-triple-E. Founded in 1884 as the AIEE, the IEEE was formed in 1963 when AIEE merged with IRE. IEEE is an organization composed of engineers, scientists, and students. The IEEE is best known for developing standards for the computer and electronics industry. In particular, the IEEE 802 standards for local-area networks are widely followed.

The IEEE has done notable work in the standards area of networking. This organization is huge with over 300,000 members made up of engineers, technicians, scientists, and students in related areas. The Computer Society of IEEE alone has over 100,000 members.

IEEE is credited with having provided definitive standards in local area networking.

The 802 standards were the culmination of work performed by the subcommittee starting in 1980. This was followed in 1985 with specific LAN-oriented standards titled 802.2 - 802.5. Since that time there have been other references set up as well. Most of the work performed by the 802 Project committee revolves around the first two layers of the OSI model initiated by the ISO. These layers involve the physical medium on which you move data (cable type) and the way that interacts with it. It addresses such crucial issues of how data is placed on the network and how insures its accuracy and flow. In order to better define these functions, the IEEE split the Data Link layer of the OSI model up into two separate components.

802 IEEE committee responsible for setting standards concerning cabling physical topologies logical topologies and physical access methods for networking products. The Computer Society of IEEE's 802 Project Committee is divided into several subcommittees that deal with specific standards in these general areas. Specifically the Physical layer and the Data Link layer of the ISO's OSI model are addressed.

802.1 This work defines an overall picture of LANs and connectivity. 802.1B this set of standards specifically addressed network management.

802.1D Standards for bridges used to connect various types of LANs together were set up with 802.1D.

802.2 called the logical link control (LLC) standards; this specification governs the communication of packets of information from one device to another on a network. Specifically it deals with communication, not access to the network itself.

802.3 Defines the way data has access to a network for multiple topology systems using Carrier Sense Multiple Access/ Collision Detection (CSMA/CD). A prime example is Ethernet and Star LAN systems. These LAN types operate at 10 Mb/sec.

802.4 Standards developed for a token-passing scheme on a bus topology. The primary utilizer of this specification was the Manufacturing Automation Protocol LANs developed by General Motors. Operate at 10 Mb/sec.

802.5 This standard defines token ring systems. It involves the token-passing concept on a ring topology with twisted pair cabling. IBM's token ring system uses this specification. The speed is either 4 Mb/sec or 16 Mb/sec.

802.6 Metropolitan Area Networks are defined by this group. MANs are networks that are larger than LANs typically falling within 50 kilometers. They operate at speeds ranging from 1 Mb/sec up to about 200 Mb/sec.

802.7 These are standards concerning broadband LANs.

802.8 This group sets up standards for LANs using fiber optic cabling and access methods.

802.9 This specification covers voice and digital data integration.

802.10 These members set standards for interoperable security.

802.11 Wireless LANs are the subject of this particular subcommittee's works. Both infrared and radio LANs are covered.

IEEE standard for Floating Point Representation

The IEEE (Institute of Electrical and Electronics Engineers) has produced a standard for floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented, as well as how arithmetic should be carried out on them.

Single Precision

The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right. The first bit is the sign bit, S, the next

eight bits are the exponent bits, 'E', and the final 23 bits are the fraction 'F':

S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF

0 1 8 9 31

The value V represented by the word may be determined as follows:

- If E=255 and F is nonzero, then V=NaN ("Not a number")
- If E=255 and F is zero and S is 1, then V=-Infinity
- If E=255 and F is zero and S is 0, then V=Infinity
- If 0<E<255 then V= (-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V= (-1)**S * 2 ** (-126) * (0.F) These are "normalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

In particular,

0 00000000 00000000000000000000000 = 0

1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinity

1 11111111 00000000000000000000000 = -Infinity

0 11111111 00000100000000000000000 = NaN

1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2

0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5

1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5

0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)

0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)

0 00000000 00000000000000000000001 = +1 * 2**(-126) *
0.00000000000000000000001 =
2**(-149) (Smallest positive value)

Double Precision

The IEEE double precision floating point standard representation requires a 64 bit word, which may be represented as numbered from 0 to 63, left to right. The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F':

S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

0 1 11 12 63

The value V represented by the word may be determined as follows:

- If E=2047 and F is nonzero, then V=NaN ("Not a number")
- If E=2047 and F is zero and S is 1, then V=-Infinity
- If E=2047 and F is zero and S is 0, then V=Infinity
- If 0<E<2047 then V=(-1)**S * 2 ** (E-1023) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-1022) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

3.3 MUTATION

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m. If the probability is very high,

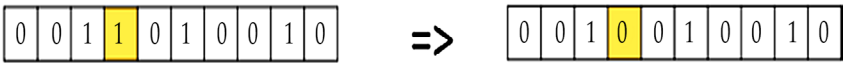
the GA gets reduced to a random search. Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

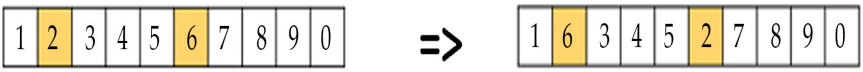


Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

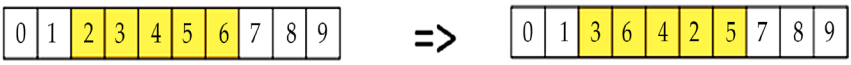
Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



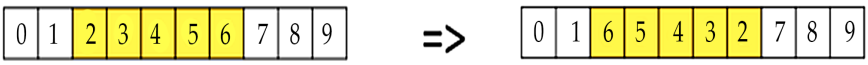
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



3.3.1 Genetic Algorithms - Population

Population is a subset of solutions in the current generation. It can also be defined as a set of chromosomes. There are several things to be kept in mind when dealing with GA population −

- The diversity of the population should be maintained otherwise it might lead to premature convergence.
- The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

The population is usually defined as a two-dimensional array of – size population, size x, chromosome size.

Population Initialization

There are two primary methods to initialize a population in a GA. They are −

- *Random Initialization* − Populate the initial population with completely random solutions.
- *Heuristic initialization* − Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

Population Models

There are two population models widely in use −

Steady State

In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as Incremental GA.

Generational

In a generational model, we generate 'n' off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

3.3.2 Genetic Algorithms - Parent Selection

Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. Maintaining good diversity in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as premature convergence and is an undesirable condition in a GA.

Fitness Proportionate Selection

Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

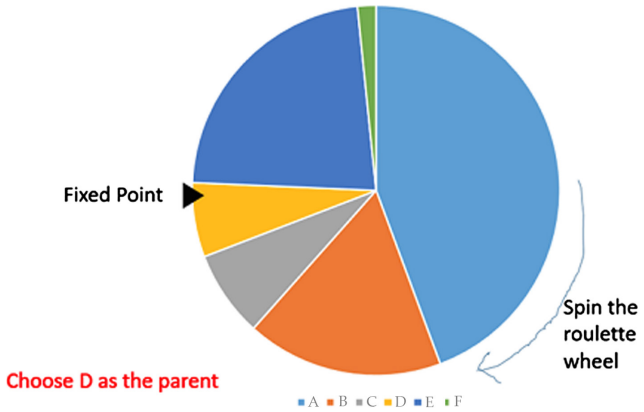
Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

Two implementations of fitness proportionate selection are possible −

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which

comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

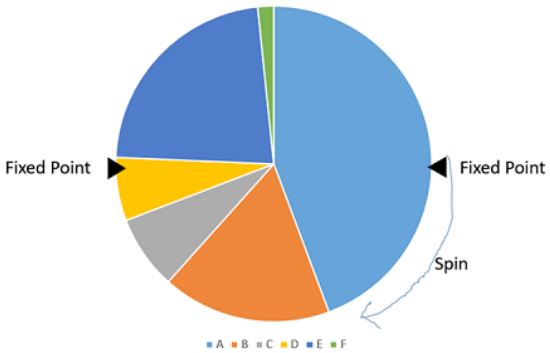
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

Implementation wise, we use the following steps −

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

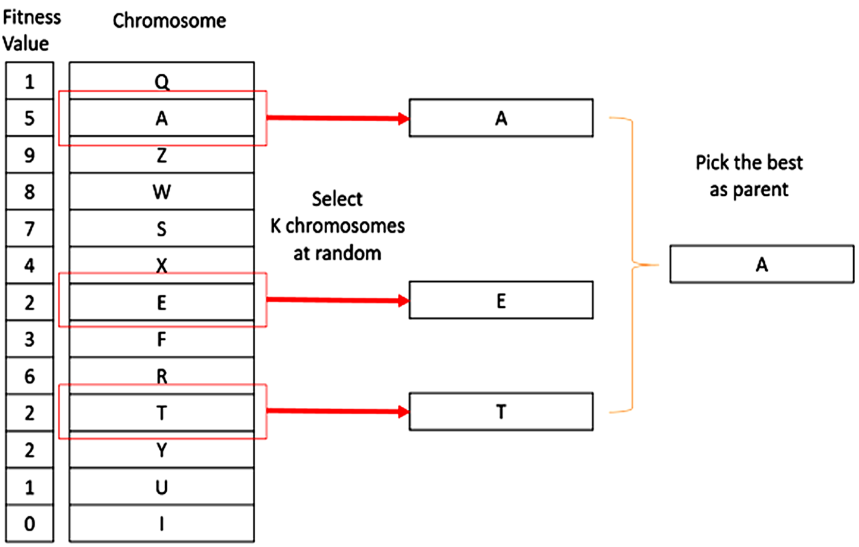


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

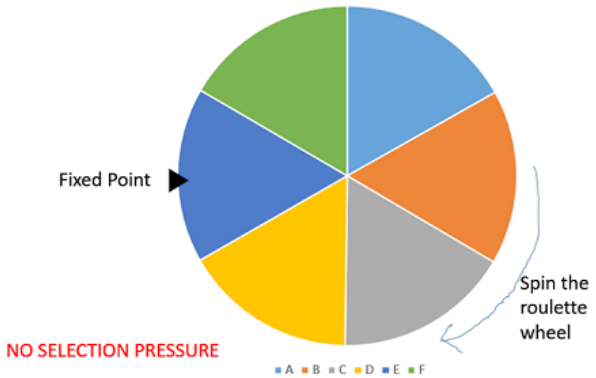
Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	**Fitness Value**	**Rank**
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6

E	8.02	3
F	7.99	5

Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

REFERENCES

1. A. Alameldeen and D. Wood. Variability in architectural simulations of multi-threaded workloads. In Proceedings of the Ninth International Symposium on High-Performance Computer Architecture (HPCA), pages 7–18, February 2003.
2. E. Argollo, A. Falcón, P. Faraboschi,M.Monchiero, and D. Ortega. COTSon: Infrastructure for full system simulation. SIGOPS Operating System Review, 43(1):52–61, January 2009.
3. T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. IEEE Computer, 35(2):59–67, February 2002. 51, 55
4. R. Bell, Jr. and L. K. John. Improved automatic testcase synthesis for performance model validation. In Proceedings of the 19th ACM International Conference on Supercomputing (ICS), pages 111–120, June 2005.
5. E. Berg and E. Hagersten. Fast data-locality profiling of native execution. In Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS), pages 169–180, June 2005.
6. C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 72–81, October 2008.
7. N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. IEEE Micro, 26(4):52–60, 2006.
8. M. Burtscher and I. Ganusov. Automatic synthesis of high-speed processor simulators. In Proceedings of the 37th IEEE/ACM Symposium on Microarchitecture (MICRO), pages 55–66, December 2004. 52, 72
9. M.Burtscher,I.Ganusov,S.J.Jackson,J.Ke,P.Ratanaworabhan, and N. B. Sam. The VPC trace-compression algorithms. IEEE Transactions on Computers, 54(11):1329–1344, November 2005.

10. D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip-multiprocessor architecture. In Proceedings of the Eleventh International Symposium on High Performance Computer Architecture (HPCA), pages 340–351, February 2005. 7, 91, 93
11. J. Chen, M. Annavaram, and M. Dubois. SlackSim: A platform for parallel simulation of CMPs on CMPs. ACM SIGARCH Computer Architecture News, 37(2):20–29, May 2009.
12. X. E. Chen and T. M. Aamodt. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs. In Proceedings of the International Symposium on Microarchitecture (MICRO), pages 59–70, December 2008. 46

CHAPTER 4

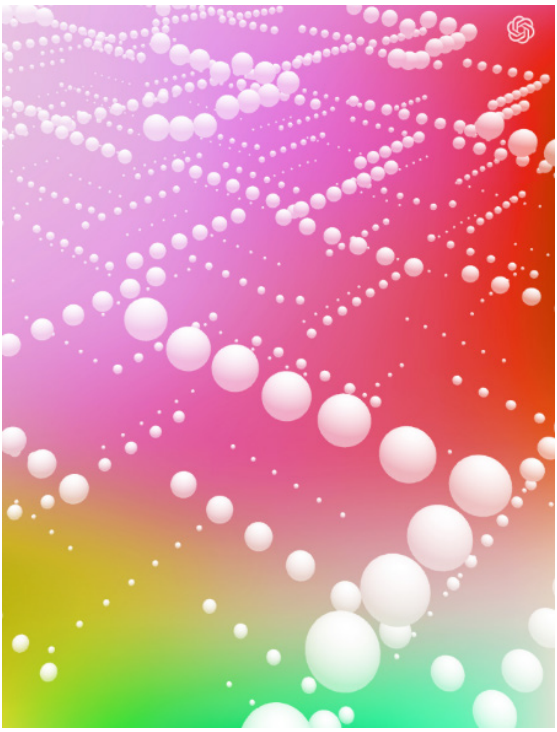
INTRODUCTION TO EVOLUTION STRATEGY

INTRODUCTION

Neural network models are highly expressive and flexible, and if we are able to find a suitable set of model parameters, we can use neural nets to solve many challenging problems. Deep learning's success largely comes from the ability to use the backpropagation algorithm to efficiently calculate the gradient of an objective function over each model parameter. With these gradients, we can efficiently search over the parameter space to find a solution that is often good enough for our neural net to accomplish difficult tasks.

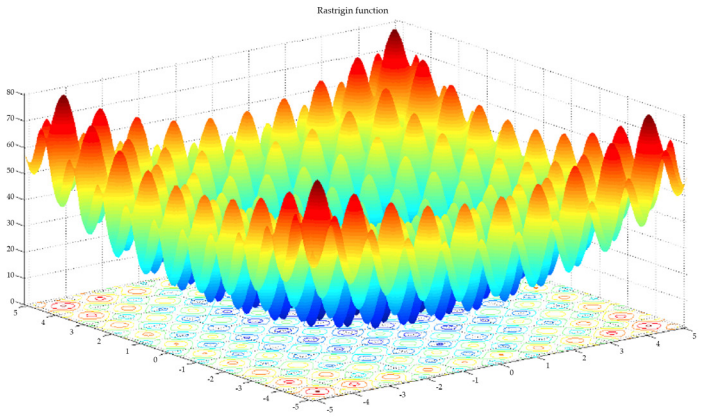
However, there are many problems where the backpropagation algorithm cannot be used. For example, in reinforcement learning (RL) problems, we can also train a neural network to make decisions to perform a sequence of actions to accomplish some task in an environment. However, it is not trivial to estimate the

gradient of reward signals given to the agent in the future to an action performed by the agent right now, especially if the reward is realised many timesteps in the future. Even if we are able to calculate accurate gradients, there is also the issue of being stuck in a local optimum, which exists many for RL tasks.

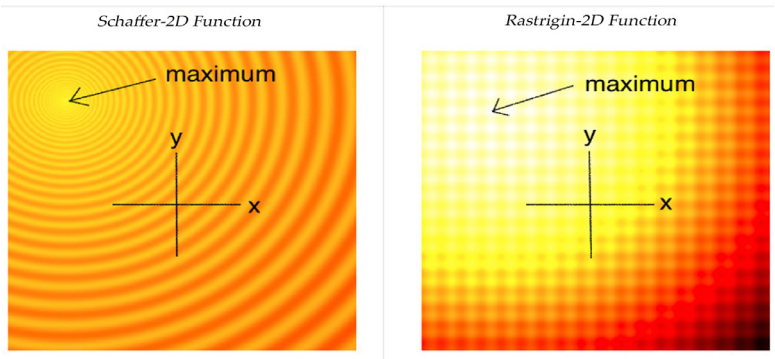


4.1 OVERVIEW OF EVOLUTION STRATEGY

The diagrams below are top-down plots of shifted 2D Schaffer and Rastrigin functions, two of several simple toy problems used for testing continuous black-box optimization algorithms. Lighter regions of the plots represent higher values of F(x,y). As you can see, there are many local optimums in this function. Our job is to find a set of model parameters (x,y), such that F(x,y) is as close as possible to the global maximum.



Although there are many definitions of evolution strategies, we can define an evolution strategy as an algorithm that provides the user a set of candidate solutions to evaluate a problem. The evaluation is based on an *objective function* that takes a given solution and returns a single *fitness* value. Based on the fitness results of the current solutions, the algorithm will then produce the next generation of candidate solutions that is more likely to produce even better results than the current generation. The iterative process will stop once the best known solution is satisfactory for the user.



Given an evolution strategy algorithm called EvolutionStrategy, we can use in the following way:

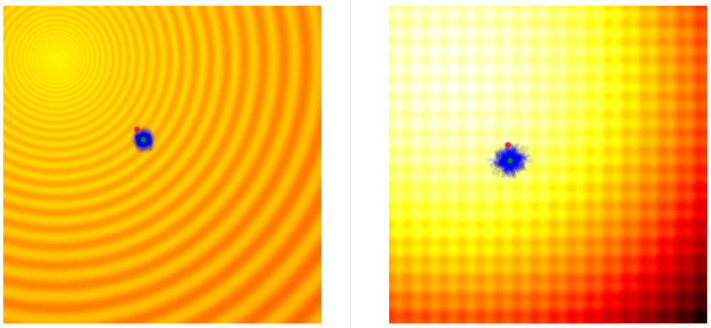
```
solver = EvolutionStrategy()
while True:
```

```
# ask the ES to give us a set of candidate solutions
solutions = solver.ask()
# create an array to hold the fitness results.
fitness_list = np.zeros(solver.popsize)
# evaluate the fitness for each given solution.
for i in range(solver.popsize):
  fitness_list[i] = evaluate(solutions[i])
# give list of fitness results back to ES
solver.tell(fitness_list)
# get best parameter, fitness from ES
best_solution, best_fitness = solver.result()
if best_fitness > MY_REQUIRED_FITNESS:
  break
```

Although the size of the population is usually held constant for each generation, they don't need to be. The ES can generate as many candidate solutions as we want, because the solutions produced by an ES are *sampled* from a distribution whose parameters are being updated by the ES at each generation. I will explain this sampling process with an example of a simple evolution strategy.

4.1.1 SIMPLE EVOLUTION STRATEGY

One of the simplest evolution strategy we can imagine will just sample a set of solutions from a Normal distribution, with a mean \ muμ and a fixed standard deviation \ sigmaσ. In our 2D problem, $\mu = (\mu_x, \mu_y)$ and $\sigma = (\sigma_x, \sigma_y)$ Initially, μ is set at the origin. After the fitness results are evaluated, we set μ to the best solution in the population, and sample the next generation of solutions around this new mean. This is how the algorithm behaves over 20 generations on the two problems mentioned earlier:



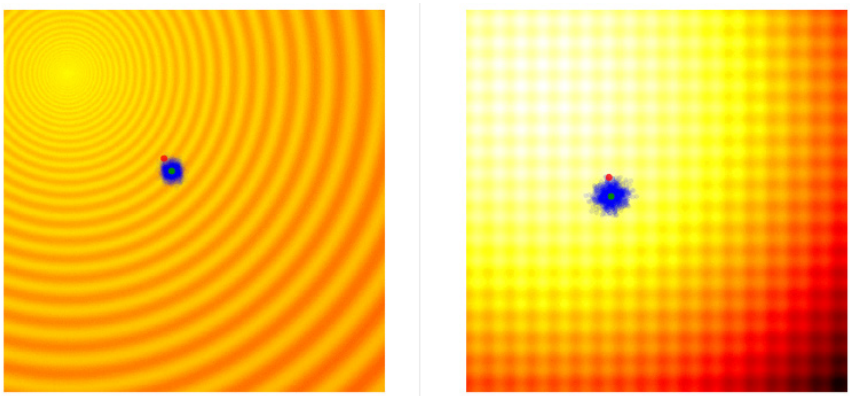
In the visualisation above, the green dot indicates the mean of the distribution at each generation, the blue dots are the sampled solutions, and the red dot is the best solution found so far by our algorithm.

This simple algorithm will generally only work for simple problems. Given its greedy nature, it throws away all but the best solution, and can be prone to be stuck at a local optimum for more complicated problems. It would be beneficial to sample the next generation from a probability distribution that represents a more diverse set of ideas, rather than just from the best solution from the current generation.

4.1.2 Simple Genetic Algorithm

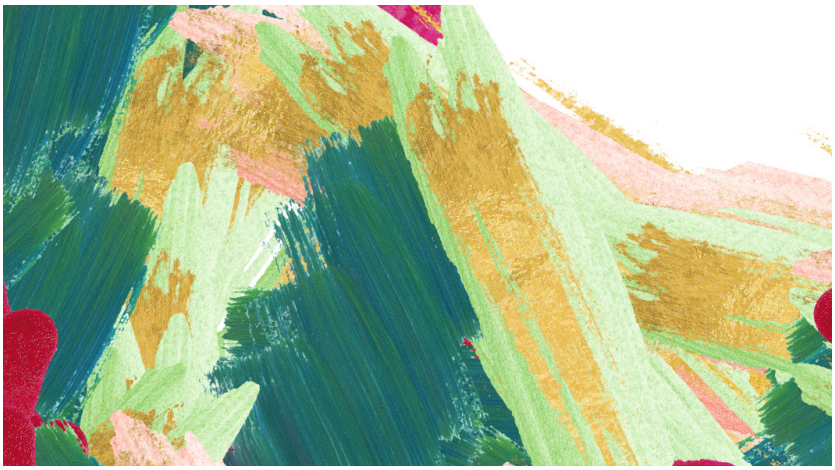
One of the oldest black-box optimization algorithms is the genetic algorithm. There are many variations with many degrees of

sophistication, but I will illustrate the simplest version here. The idea is quite simple: keep only 10% of the best performing solutions in the current generation, and let the rest of the population die. In the next generation, to sample a new solution is to randomly select two solutions from the survivors of the previous generation, and recombine their parameters to form a new solution. This crossover recombination process uses a coin toss to determine which parent to take each parameter from. In the case of our 2D toy function, our new solution might inherit x or y from either parents with 50% chance. Gaussian noise with a fixed standard deviation will also be injected into each new solution after this recombination process.



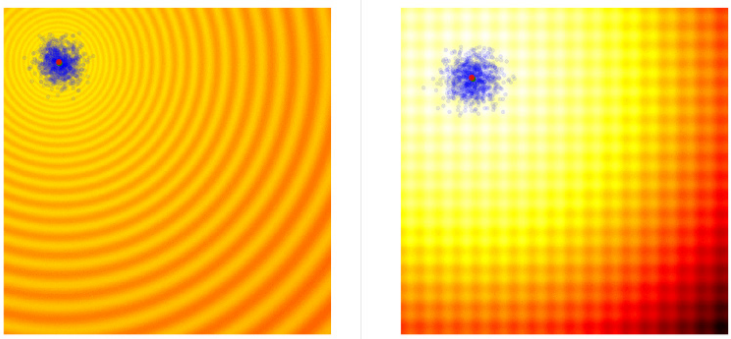
The figure above illustrates how the simple genetic algorithm works. The green dots represent members of the elite population from the previous generation, the blue dots are the offspring's to form the set of candidate solutions, and the red dot is the best solution.

Genetic algorithms help diversity by keeping track of a diverse set of candidate solutions to reproduce the next generation. However, in practice, most of the solutions in the elite surviving population tend to converge to a local optimum over time. There are more sophisticated variations of GA out there, such as CoSyNe, ESP, and NEAT, where the idea is to cluster similar solutions in the population together into different species, to maintain better diversity over time.



4.1.3 Covariance-Matrix Adaptation Evolution Strategy (CMA-ES)

A shortcoming of both the Simple ES and Simple GA is that our standard deviation noise parameter is fixed. There are times when we want to explore more and increase the standard deviation of our search space, and there are times when we are confident we are close to a good optima and just want to fine tune the solution. We basically want our search process to behave like this:



Amazing isn'it it? The search process shown in the figure above is produced by Covariance-Matrix Adaptation Evolution Strategy (CMA-ES). CMA-ES an algorithm that can take the results of each generation, and adaptively increase or decrease the search

space for the next generation. It will not only adapt for the mean \ muμ and sigma \sigmaσ parameters, but will calculate the entire covariance matrix of the parameter space. At each generation, CMA-ES provides the parameters of a multi-variate normal distribution to sample solutions from. So how does it know how to increase or decrease the search space?

Before we discuss its methodology, let's review how to estimate a covariance matrix. This will be important to understand CMA-ES's methodology later on.

If we want to estimate the covariance matrix of our entire sampled population of size of N, we can do so using the set of equations below to calculate the maximum likelihood estimate of a covariance matrix C. We first calculate the means of each of the x_i and y_i in our population:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

$$\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i.$$

The terms of the 2x2 covariance matrix *C* **will** be:

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu_x)^2,$$

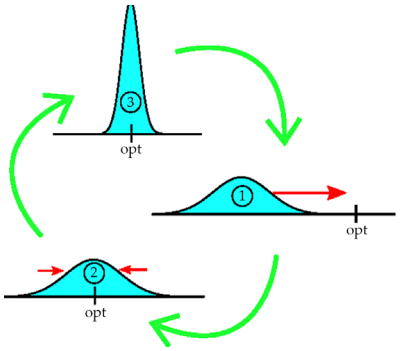
$$\sigma_y^2 = \frac{1}{N}\sum_{i=1}^{N} (y_i - \mu_y)^2,$$

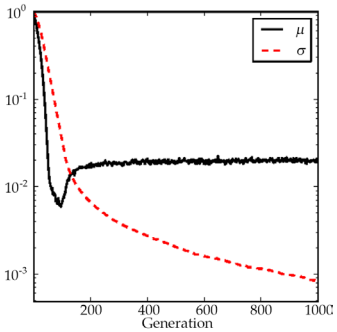
$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y).$$

4.2 NATURAL EVOLUTION STRATEGIES

Imagine if you had built an artificial life simulator, and you sample a different neural network to control the behavior of each ant inside an ant colony. Using the Simple Evolution Strategy for this task will optimize for traits and behaviors that benefit individual ants, and with each successive generation, our population will be full of alpha ants who only care about their own well-being.

Instead of using a rule that is based on the survival of the fittest ants, what if you take an alternative approach where you take the sum of all fitness values of the entire ant population, and optimize for this sum instead to maximize the well-being of the entire ant population over successive generations? Well, you would end up creating a Marxist utopia.





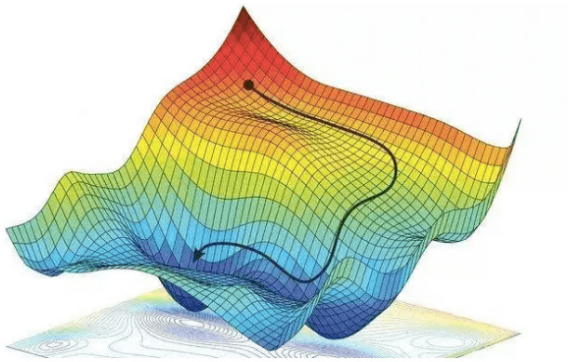
A perceived weakness of the algorithms mentioned so far is that they discard the majority of the solutions and only keep the best solutions. Weak solutions contain information about what not to do, and this is valuable information to calculate a better estimate for the next generation.

In this approach, we want to use all of the information from each member of the population, good or bad, for estimating a gradient signal that can move the entire population to a better direction in the next generation. Since we are estimating a gradient, we can also use this gradient in a standard SGD update rule typically used for deep learning. We can even use this estimated gradient with Momentum SGD, RMSProp, or Adam if we want to.

The idea is to maximize the expected value of the fitness score of a sampled solution. If the expected result is good enough, then the best performing member within a sampled population will be even better, so optimizing for the expectation might be a sensible approach. Maximising the expected fitness score of a sampled solution is almost the same as maximizing the total fitness score of the entire population.

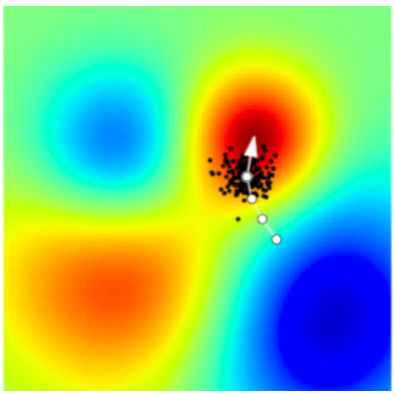
4.3 NUMERICAL OPTIMIZATION

Almost every machine learning algorithm can be posed as an optimization problem. In an ML algorithm, we update the model's parameters to minimize the loss. For example, every supervised learning algorithm can be written as, θ_estimate = argmin $\mathbb{E}$[L(y,f(x,θ))], where x and y represent the features and the target respectively, θ represents model parameters, f represents the function we are trying to model and L represents the Loss function, which measures how good our fit is. Gradient Descent algorithm also known as steepest descent has proven to solve such problems well in most of the cases. It is a first-order iterative algorithm for finding the local minimum of a differentiable function. We take steps proportional to the negative of the gradient of the Loss function at the current point, i.e. θ_new = θ_old — α*∇ L(y, f(x, θ_old)). Newton's Method is another second-order iterative method which converges in fewer iterations but is computationally expensive as the inverse of second-order derivative of the loss function (Hessian matrix) needs to be calculated, i.e. θ_new = θ_old — [∇² L(y, f(x, θ_old))]^(-1) * ∇ L(y, f(x, θ_old)). We are searching for parameter using the gradients as we believe that it will lead us in the direction where loss will get reduced. But can we search for optimal parameters without calculating any gradients? Actually, there are many ways to solve this problem! There are bunch of different Derivitive-free optimization algorithms (also known as Black-Box optimization).



4.3.1 Evolution Strategies

Gradient descent might not always solve our problems. Why? The answer is local optimum in short. For example in case of sparse reward scenarios in reinforcement learning where agent receives reward at the end of episode, like in chess with end reward as +1 or -1 for winning or losing the game respectively. In case we lose the game, we won't know whether we played horribly wrong or just made a small mistake. The reward gradient signal is largely uninformative and can get us stuck. Rather than using noisy gradients to update our parameters we can resort to derivative-free techniques such as Evolution Strategies (ES). ES works out well in such cases and also where we don't know the precise analytic form of an objective function or cannot compute the gradients directly.



Moreover, they found out that ES discovered more diverse policies compared to traditional Reinforcement learning algorithm. ES are nature-inspired optimization methods which use random mutation, recombination, and selection applied to a population of individuals containing candidate solutions in order to evolve iteratively better solutions. It is really useful for non-linear or non-convex continuous optimization problems.

In ES, we don't care much about the function and its relationship with the inputs or parameters. Some million numbers (parameters of the model) go into the algorithm and it spits out 1 value (e.g. loss in supervised setting; reward in case of Reinforcement Learning). We try to find the best set of such numbers which returns good values for our optimization problem. We are optimizing a function J(θ) with respect to the parameters θ, just by evaluating it without making any assumptions about the structure of J, and hence the name 'black-box optimization'. Let's dig deep into the implementation details!

4.3.2 Vanilla Implementation

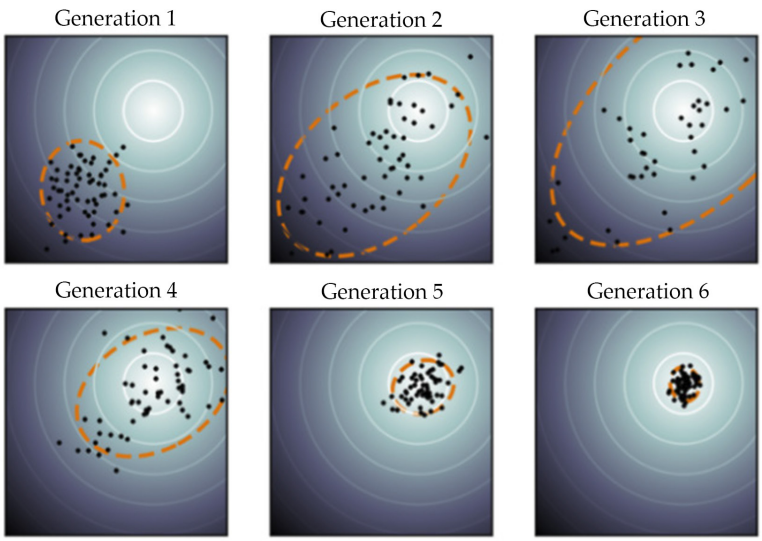
To start with, we randomly generate the parameters and tweak it such that the parameters work better slightly. Mathematically, at each step we take a parameter vector θ and generate a population of, say, 100 slightly different parameter vectors $\theta_1,\theta_2\ldots\theta_{100}$ by jittering θ with Gaussian noise. We then evaluate each one of the 100 candidates independently by running the model and based on the output value evaluate the loss or the objective function. We then select top N best performing elite parameters, N can be say 10, and take the mean of these parameters and call it our best parameter so far. We then repeat the above process by again generating 100 different parameters by adding Gaussian noise to our best parameter obtained so far.

Thinking in terms of natural selection, we are creating a population of parameters (species) randomly and selecting the top parameters that perform well based on our objective function (also known as fitness function). We then take combine the best qualities of

these parameters by taking their mean (this is a crude way but it still works!) and call it our best parameter. We then recreate the population by mutating this parameter by adding random noise and repeat the whole process till convergence.

4.3.3 Pseudo Code

- Randomly initialize the best parameter using a Gaussian distribution
- Loop until convergence:
  - Create population of parameters $\theta_1,\theta_2\ldots\theta_{100}$ by adding Gaussian noise to the best parameter
  - Evaluate the objective function for all the parameters and select the top N best performing parameters (elite parameters)
  - Best parameter = Mean(top N elite parameters)
  - Decay the noise at the end of each iteration by some factor (At the start more noise will help us to explore better but as we reach the convergence point we want the noise to be minimum so as to not deviate away)



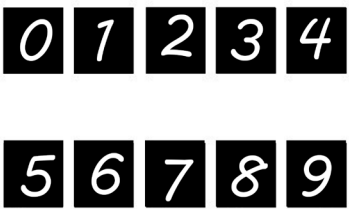
4.3.4 Python Implementation from scratch

Let's go through a simple example in Python to get a better understanding. I tried to add details related to numerical stability as well for few of the things. Please read the comments! We will start by loading the required libraries and the MNIST Handwritten digit dataset.

```
Importing all the required libraries
import numpy as np
import matplotlib.pyplot as plt
import tqdm
import pickle
import warnings
warnings.filterwarnings('ignore')
from keras.datasets import mnist
# Machine Epsilon (needed to calculate logarithms)
eps = np.finfo(np.float64).eps
# Loading MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# x contains the images (features to our model)
# y contains the labels 0 to 9
# Normalizing the inputs between 0 and 1
x_train = x_train/255.
x_test = x_test/255.
# Flattening the image as we are using
# dense neural networks
x_train = x_train.reshape( -1, x_train.shape[1]*x_train.shape[2])
x_test = x_test.reshape( -1, x_test.shape[1]*x_test.shape[2])
# Converting to one-hot representation
identity_matrix = np.eye(10)
y_train = identity_matrix[y_train]
```

```
y_test = identity_matrix[y_test]
# Plotting the images
fig, ax = plt.subplots(2,5)
for i, ax in enumerate(ax.flatten()):
 im_idx = np.argwhere(y_train == i)[0]
 plottable_image = np.reshape(x_train[im_idx], (28, 28))
 ax.set_axis_off()
 ax.imshow(plottable_image, cmap='gray')
plt.savefig('mnist.jpg')
```

This is how the images look like,



We will start by defining our model, which will be a single layer neural network with only forward pass.

```
def soft_max(x):
'''
 Arguments: numpy array
 Returns: numpy array after applying
 softmax function to each
 element
'''
 # Subtracting max of x from each element of x for numerical
 # stability as this results in the largest argument to
 # exp being 0, ruling out the possibility of overflow
```

```
# Read more about it at :
# https://www.deeplearningbook.org/contents/numerical.html
e_x = np.exp(x — np.max(x))
return e_x /e_x.sum()
class Model():
'''
Single layer Neural Network
'''
def __init__(self, input_shape, n_classes):
# Number of output classes
self.n_classes = n_classes
# Parameters/Weights of our network which we will be updating
self.weights = np.random.randn(input_shape, n_classes)
def forward(self,x):
'''
Arguments: numpy array containing the features,
expected shape of input array is
(batch size, number of features)
Returns: numpy array containing the probability,
expected shape of output array is
(batch size, number of classes)
'''
# Multiplying weights with inputs
x = np.dot(x,self.weights)
```

```
# Applying softmax function on each row
x = np.apply_along_axis(soft_max, 1, x)
return x
def __call__(self,x):
'''
This dunder function
enables your model to be callable
When the model is called using model(x),
forward method of the model is called internally
'''
return self.forward(x)
def evaluate(self, x, y, weights = None):
'''
Arguments : x — numpy array of shape (batch size,number of features),
y — numpy array of shape (batch size,number of classes),
weights — numpy array containing the parameters of the model
Returns : Scalar containing the mean of the categorical cross-entropy loss
of the batch
'''
if weights is not None:
self.weights = weights
# Calculating the negative of cross-entropy loss (since
# we are maximizing this score)
```

```
# Adding a small value called epsilon
# to prevent -inf in the output
log_predicted_y = np.log(self.forward(x) + eps)
return (log_predicted_y*y).mean()
```

We will now define our function which will take a model as input and update its parameters.

```
def optimize(model,x,y,
top_n = 5, n_pop = 20, n_iter = 10,
sigma_error = 1, error_weight = 1, decay_rate = 0.95,
min_error_weight = 0.01 ):
'''
Arguments : model — Model object(single layer neural network here),
x — numpy array of shape (batch size, number of features),
y — numpy array of shape (batch size, number of classes),
top_n — Number of elite parameters to consider for calculating the
best parameter by taking mean
n_pop — Population size of the parameters
n_iter — Number of iteration
sigma_error — The standard deviation of errors while creating
population from best parameter
error_weight — Contribution of error for considering new population
decay_rate — Rate at which the weight of the error will reduce after
```

```
each iteration, so that we don't deviate away at the
point of convergence. It controls the balance between
exploration and exploitation
Returns : Model object with updated parameters/weights
'''
# Model weights have been randomly initialized at first
best_weights = model.weights
for i in range(n_iter):
# Generating the population of parameters
pop_weights = [best_weights + error_weight*sigma_error* \
np.random.randn(*model.weights.shape)
for i in range(n_pop)]
# Evaluating the population of parameters
 evaluation_values = [model.evaluate(x,y,weight) for weight in
pop_weights]
# Sorting based on evaluation score
weight_eval_list = zip(evaluation_values, pop_weights)
 weight_eval_list = sorted(weight_eval_list, key = lambda x: x[0],
reverse = True)
evaluation_values, pop_weights = zip(*weight_eval_list)
# Taking the mean of the elite parameters
    best_weights = np.stack(pop_weights[:top_n], axis=0).
mean(axis=0)
#Decaying the weight
error_weight = max(error_weight*decay_rate, min_error_weight)
model.weights = best_weights
```

```
 return model
# Instantiating our model object
model = Model(input_shape= x_train.shape[-1], n_classes= 10)
print("Evaluation on training data", model.evaluate(x_train, y_
train))
# Running it for 200 steps
for i in tqdm.tqdm(range(200)):
 model = optimize(model,
 x_train,
 y_train,
 top_n = 10,
 n_pop = 100,
 n_iter = 1)
 print("Test data cross-entropy loss: ", -1*model.evaluate(x_test,
y_test))
 print("Test Accuracy: ",(np.argmax(model(x_test),axis=1) == y_
test).mean())
# Saving the model for later use
with open('model.pickle','wb') as f:
 pickle.dump(model,f)
```

REFERENCES

1. Abadir K, Magnus J (2005) Matrix algebra. Cambridge University Press, New York
2. Arnold D (2006) Weighted multirecombination evolution strategies. Theor Comput Sci 361(1):18–37
3. Arnold D, Beyer HG (2006) A general noise model and its effects on evolution strategy performance. IEEE Trans Evolut Comput 10(4):380–391
4. Arnold D, Salomon R (2007) Evolutionary gradient search revisited. IEEE Trans Evolut Comput 11(4):480–495
5. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC 2005), vol 2. IEEE Press, Piscataway, NJ, pp 1769–1776
6. Auger A, Teytaud O (2007) Continuous lunches are free! In: Thierens D et al. (eds) Proceedings of the genetic and evolutionary computation conference (GECCO 2007). ACM Press, New York, pp 916–922
7. Auger A, Schoenauer M, Vanhaecke N (2004) LS-CMA-ES: a second-order algorithm for covariance matrix adaptation. In: Yao X et al. (eds) Proceedings of the 8th international conference on parallel problem solving from nature (PPSN VIII). Springer, Berlin, Germany, pp 182–191
8. Bäck T, Hoffmeister F, Schwefel HP (1991) A survey of evolution strategies. In: Belew RK, Booker LB (eds) Proceedings of the fourth international conference on genetic algorithms (ICGA'91). Morgan Kaufmann, San Mateo CA, pp 2–9
9. Bartz-Beielstein T (2006) Experimental research in evolutionary computation. The new experimentalism. Springer, Heidelberg
10. Beyer HG, Meyer-Nieberg S (2006) Self-adaptation of evolution strategies under noisy fitness evaluations. Genet Programming Evolvable Mach 7(4):295–328

11. Beyer HG, Schwefel HP (2002) Evolution strategies – a comprehensive introduction. Nat Comput 1(1):3–52
12. Beyer HG, Sendhoff B (2008) Covariance matrix adaptation revisited – the CMSA evolution strategy. In: Rudolph G et al. (eds) Proceedings of the 10th international conference on parallel problem solving from nature (PPSN X). Springer, Berlin, Germany, pp 123–132
13. Emmerich M, Giotis A, Oezdemir M, Bäck T, Giannakoglou K (2002) Metamodel-assisted evolution strategies. In: Merelo J et al. (eds) Proceedings of the 7th international conference on parallel problem solving from nature (PPSN VII). Springer, Berlin, Germany, pp 361–370
14. Fang KT, Kotz S, Ng KW (1990) Symmetric multivariate and related distributions. Chapman and Hall, London, UK
15. Hansen N (2009) The CMA evolution strategy: A tuto- rial. Continuously updated technical report. Available via http://www.lri.fr/~hansen/cmatutorial.pdf. Accessed April 2009
16. Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of the 1996 IEEE international conference on evolutionary computation (ICEC '96). IEEE Press, Piscataway, NJ, pp 312–317
17. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evolut Comput 9(2):159–195
18. Hansen N, Ostermeier A, Gawelczyk A (1995) On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation. In: Eshelman L (ed) Proceedings of the 6th international conference on genetic algorithms (ICGA 6). Morgan Kaufmann, San Fransisco, CA, pp 57–64
19. Hoffmann F, Hölemann S (2006) Controlled model assisted evolution strategy with adaptive preselection. In: Proceedings of the 2006 international symposium on evolving fuzzy systems. IEEE Press, Piscataway, NJ, pp 182–187

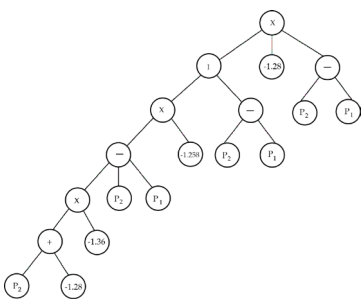
20. Igel C, Hansen C, Roth S (2006) Covariance matrix adaptation for multi-objective optimization. Evolut Comput 15(1):1–28
21. Jastrebski G, Arnold D (2006) Improving evolution strategies through active covariance matrix adaptation. In: Proceedings of the 2006 IEEE congress on evolutionary computation (CEC 2006). IEEE Press, Piscataway, NJ
22. Kramer O, Schwefel HP (2006) On three new approaches to handle constraints within evolution strategies. Nat Comput 5:363–385
23. McKay M, Beckman R, Conover W (1979) A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2):239–245

CHAPTER 5

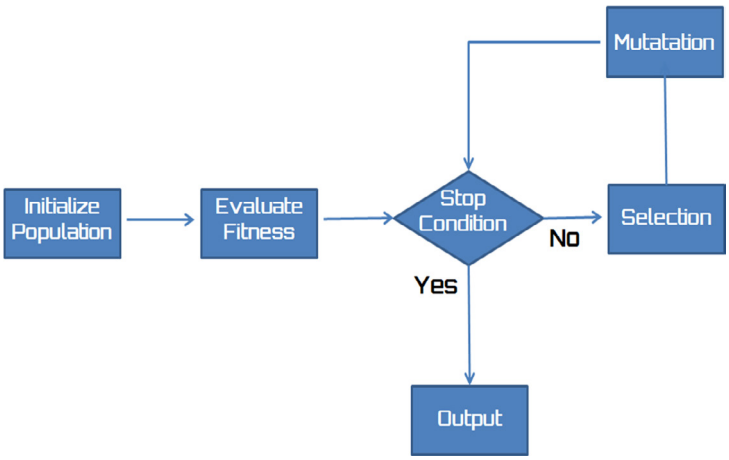
GENETIC PROGRAMMING

INTRODUCTION

Genetic Programming (GP) is a type of Evolutionary Algorithm (EA), a subset of machine learning. EAs are used to discover solutions to problems humans do not know how to solve, directly. Free of human preconceptions or biases, the adaptive nature of EAs can generate solutions that are comparable to, and often better than the best human efforts.



Inspired by biological evolution and its fundamental mechanisms, GP software systems implement an algorithm that uses random mutation, crossover, a fitness function, and multiple generations of evolution to resolve a user-defined task. GP can be used to discover a functional relationship between features in data (symbolic regression), to group data into categories (classification), and to assist in the design of electrical circuits, antennae, and quantum algorithms. GP is applied to software engineering through code synthesis, genetic improvement, automatic bug-fixing, and in developing game-playing strategies, and more.



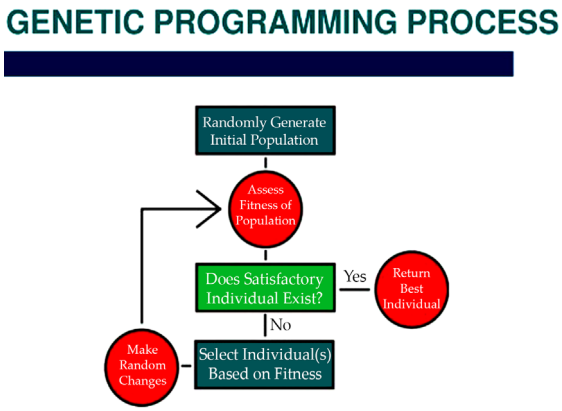
Genetic Programming is a new method to generate computer programs. It was derived from the model of biological evolution. Programs are 'bred' through continuous improvement of an initially random population of programs. Improvements are made possible by stochastic variation of programs and selection according to pre-specified criteria for judging the quality of a solution. Programs of Genetic Programming systems evolve to solve pre-described automatic programming and machine learning problems.

5.1 FUNDAMENTAL OF GENETIC PROGRAMMING

A learning process inspired by evolution and related to genetic algorithms. Whereas genetic algorithms evolve representations

of problem features to find solutions, genetic programming evolves over populations of program fragments to assemble a final program that gives a solution. The output programs may be produced in a subset of a given language or might be in the form of a decision tree.

In artificial intelligence, genetic programming (GP) is a technique of evolving programs, starting from a population of unfit (usually random) programs, fit for a particular task by applying operations analogous to natural genetic processes to the population of programs. It is essentially a heuristic search technique often described as 'hill climbing', i.e. searching for an optimal or at least suitable program among the space of all programs.



The operations are: selection of the fittest programs for reproduction (crossover) and mutation according to a predefined fitness measure, usually proficiency at the desired task. The crossover operation involves swapping random parts of selected pairs (parents) to produce new and different offspring that become part of the new generation of programs. Mutation involves substitution of some random part of a program with some other random part of a program. Some programs not selected for reproduction are copied from the current generation to the new generation. Then the selection and other operations are recursively applied to the new generation of programs. Typically, members of each new generation are on average more fit than the members of the previous generation, and the best-of-generation program is

often better than the best-of-generation programs from previous generations. Termination of the recursion is when some individual program reaches a predefined proficiency or fitness level.

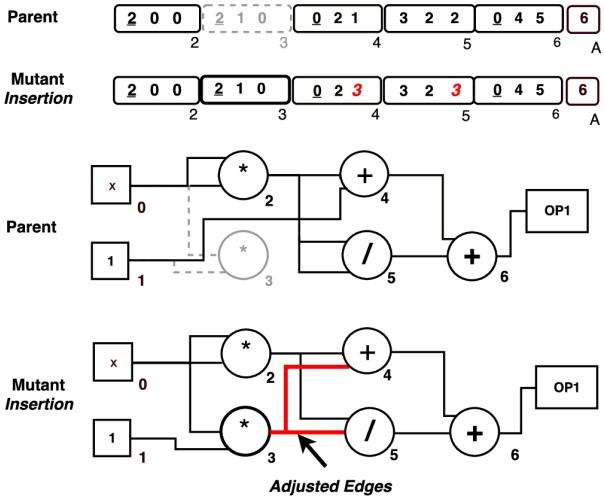
Genetic programming (GP) is an evolutionary approach that extends genetic algorithms to allow the exploration of the space of computer programs. Like other evolutionary algorithms, GP works by defining a goal in the form of a quality criterion (or fitness) and then using this criterion to evolve a set (or population) of candidate solutions (individuals) by mimicking the basic principles of Darwinian evolution. GP breeds the solutions to problems using an iterative process involving the probabilistic selection of the fittest solutions and their variation by means of a set of genetic operators, usually crossover and mutation. GP has been successfully applied to a number of challenging real-world problem domains. Its operations and behavior are now reasonably well understood thanks to a variety of powerful theoretical results.

5.1.1 Preparatory Steps of Genetic Programming

Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a

problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. The genetic operations include crossover (sexual recombination), mutation, reproduction, gene duplication, and gene deletion.

The human user communicates the high-level statement of the problem to the genetic programming system by performing certain well-defined preparatory steps.



The five major preparatory steps for the basic version of genetic programming require the human user to specify

1. The set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) for each branch of the to-be-evolved program,
2. The set of primitive functions for each branch of the to-be-evolved program,
3. The fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population),
4. Certain parameters for controlling the run, and
5. The termination criterion and method for designating the result of the run. Executional Steps of Genetic Programming

Genetic programming typically starts with a population of randomly generated computer programs composed of the available programmatic ingredients. Genetic programming iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations. These operations are applied to individual(s) selected from the population. The individuals are probabilistically selected to participate in the genetic operations based on their fitness (as measured by the fitness measure provided by the human user in the third preparatory step). The iterative transformation of the population is executed inside the main generational loop of the run of genetic programming.

The executional steps of genetic programming (that is, the flowchart of genetic programming) are as follows:

1. Randomly create an initial population (generation 0) of individual computer programs composed of the available functions and terminals.
2. Iteratively perform the following sub-steps (called a generation) on the population until the termination criterion is satisfied:

(a) Execute each program in the population and ascertain its fitness (explicitly or implicitly) using the problem's fitness measure.

(b) Select one or two individual program(s) from the population with a probability based on fitness (with reselection allowed) to participate in the genetic operations in (c).

(c) Create new individual program(s) for the population by applying the following genetic operations with specified probabilities:

(i) Reproduction: Copy the selected individual program to the new population.

(ii) Crossover: Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.

(iii) Mutation: Create one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program.

(iv) Architecture-altering operations: Choose an architecture-altering operation from the available repertoire of such operations and create one new offspring program for the new population by applying the chosen architecture-altering operation to one selected program.

- After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is harvested and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

5.1.2 Multiple predictive model structures using GP

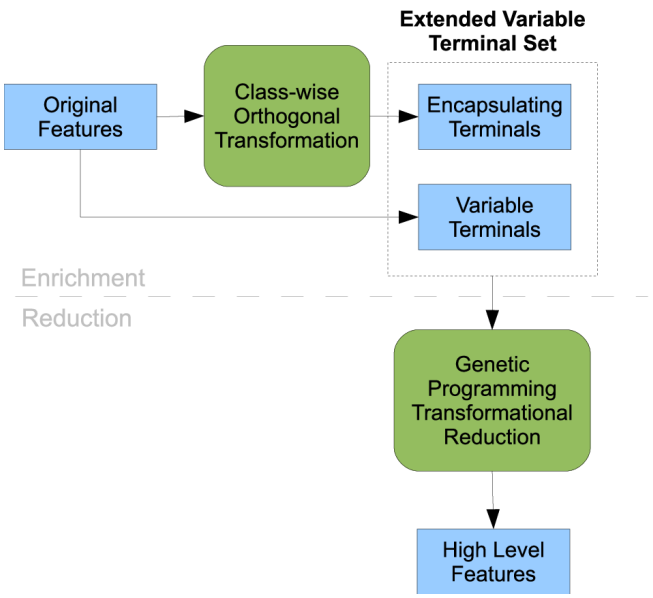
The advent of GP as a modelling tool has paved the way for researches exploring the possibility of multiple optimal models for predicting hydrological processes.

Genetic programming, in its evolutionary approach to derive optimal model structures and parameters, tests millions of model structures which can mimic the physical process under consideration.

Researchers have found that multiple models can be identified using GP which are considerably different in model structures but able to make consistently good predictions. Parasuraman and Elshorbagy developed genetic programming based models for predicting the evapo-transporation. In doing so, multiple optimal GP models were trained and tested and they were applied to quantify the uncertainty in those models. An ensemble of surrogate models based on GP was developed and the ensemble was used to get model predictions with improved reliability levels. The variance of the model predictions were used as the measure of uncertainty in the modelling process.

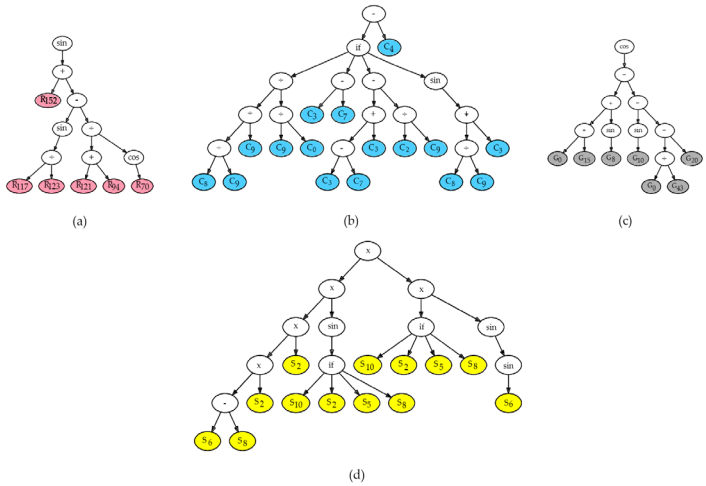
5.1.3 GP as surrogate model for simulation-optimization

A very important application of data intensive modelling approaches is to develop surrogate models to computationally complex numerical simulation models. The potential utility of the surrogates is to replace the numerical simulation model in simulation-optimization frameworks. Simulation-optimization models are used to derive optimal management decisions using optimization algorithms in which a numerical simulation models is run to predict the outcome of implementing the alternative management options. The optimal pumping from the coastal aquifer can be decided only by considering the impact of any alternative pumping strategy on saltwater intrusion. For this the numerical simulation model needs to be integrated with the optimization algorithm and the impact of each candidate pumping strategy is predicted by using the simulation model iteratively. This involve a lot of computational burden as thousands of numerical model runs are required before an optimal pumping strategy is identified.



GP was used a surrogate model within the optimization algorithm as a substitute of the numerical simulation model in the study.

Genetic programming based surrogate models for groundwater pollution source identification. It was found that genetic programming could be used as a superior surrogate model in such application with definite advantages. The study intended to develop optimal pumping strategies for coastal aquifers in which the total pumping could be maximized and at the same time limiting the saltwater intrusion at pre-specified limits.



In doing so, the effect of pumping on the salinity levels was predicted using trained and tested GP models. The GP models were externally coupled to a genetic algorithm based optimization model to derive the optimal management strategies. The results of the GP based simulation-optimization was then compared to the results obtained using an ANN-based simulation-optimization model. The ability of GP in parsimoniously identifying the model inputs helped in reducing the dimension of the decision space in which modelling and optimization was carried out. The smaller dimension of the modelling space helped in reducing the training and testing required to develop the surrogate models. The study identified that GP has potential applicability in developing surrogate models with potential application in simulation-optimization methodology to solve environmental management problems.

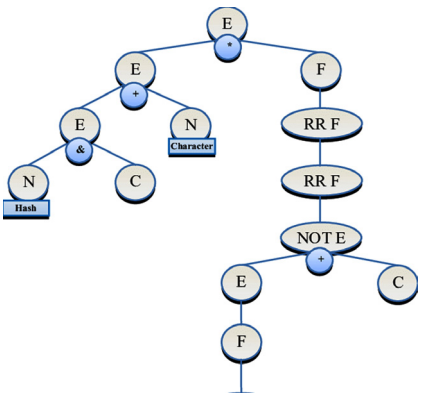
5.2 TYPES OF GENETIC PROGRAMMING

Genetic Programming (GP) is an algorithm for evolving programs to solve specific well-defined problems. It is a type of automatic programming intended for challenging problems where the task is well defined and solutions can be checked easily at a low cost, although the search space of possible solutions is vast, and there is little intuition as to the best way to solve the problem.

This often includes open problems such as controller design, circuit design, as well as predictive modeling tasks such as feature selection, classification, and regression. It can be difficult for a beginner to get started in the field as there is a vast amount of literature going back decades.

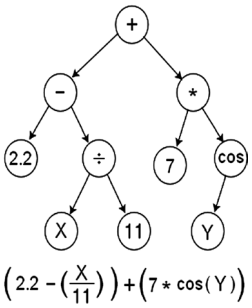
5.2.1 Tree-based Genetic Programming

In tree-based GP, the computer programs are represented in tree structures that are evaluated recursively to produce the resulting multivariate expressions. Traditional nomenclature states that a tree node (or just node) is an operator [+,-,*,/] and a terminal node (or leaf) is a variable [a,b,c,d].



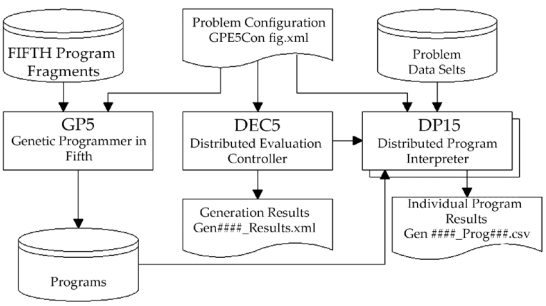
Lisp was the first programming language applied to tree-based GP, as the structure of this language matches the structure of the trees. However, many other languages such as Python, Java, and C++ have been used to develop tree-based GP applications.

Tree-based GP was the first application of Genetic Programming. There are several other types (as presented on the home page of this website) such as linear, Cartesian, and stack-based which are typically more efficient in their execution of the genetic operators. However, Tree-based GP provides a visual means to engage new users of Genetic Programming, and remains viable when built upon a fast programming language or underlying suite of libraries.



5.2.2 Stack-based GP

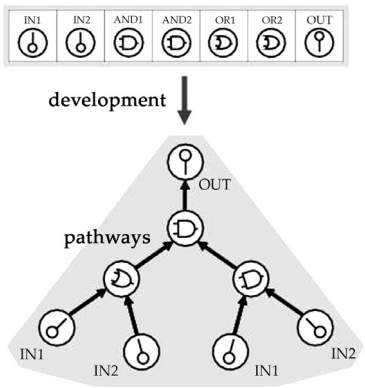
In stack-based genetic programming, the programs in the evolving population are expressed in a stack-based programming language. The specific languages vary among systems, but most are similar to FORTH insofar as programs are composed of instructions that take arguments from data stacks and push results back on those data stacks. In the Push family of languages, which were designed specifically for genetic programming, a separate stack is provided for each data type, and program code itself can be manipulated on data stacks and subsequently executed.



Depending on the specific language and genetic operators used, stack-based genetic programming can have a variety of advantages over tree-based genetic programming. These may include improvements or simplifications to the handling of multiple data types, bloat-free mutation and recombination operators, execution tracing, programs with loops that provide valid outputs even when terminated prematurely, parallelism, evolution of arbitrary control structures, and automatic simplification of evolved programs.

5.2.3 Linear Genetic Programming

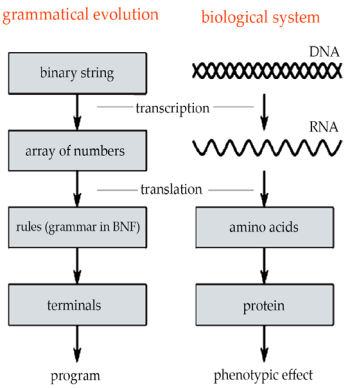
Linear genetic programming (LGP) is a particular subset of genetic programming wherein computer programs in a population are represented as a sequence of instructions from imperative programming language or machine language. The graph-based data flow that results from a multiple usage of register contents and the existence of structurally non-effective code (introns) are two main differences of this genetic representation from the more common tree-based genetic programming (TGP) variant.



In genetic programming (GP) a linear tree is a program composed of a variable number of unary functions and a single terminal. Note that linear tree GP differs from bit string genetic algorithms since a population may contain programs of different lengths and there may be more than two types of functions or more than two types of terminals.

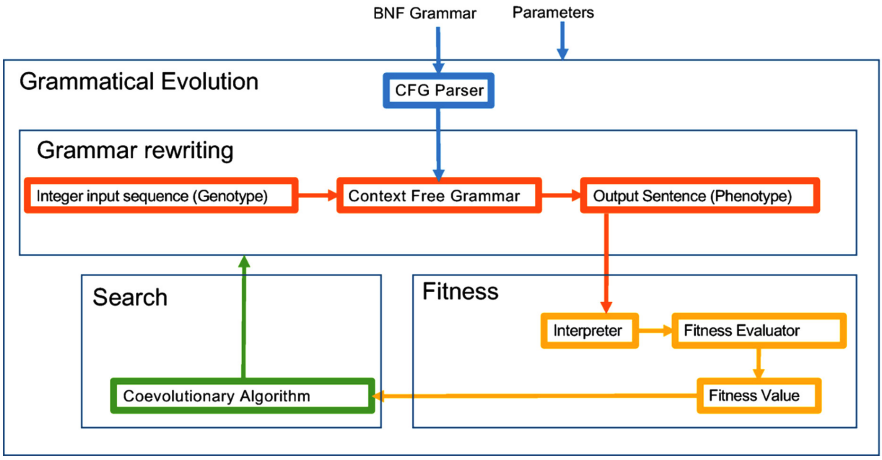
5.2.4 Grammatical Evolution

GE offers a solution to this issue by evolving solutions according to a user-specified grammar (usually a grammar in Backus-Naur form). Therefore the search space can be restricted, and domain knowledge of the problem can be incorporated. The inspiration for this approach comes from a desire to separate the "genotype" from the "phenotype": in GP, the objects the search algorithm operates on and what the fitness evaluation function interprets are one and the same. In contrast, GE's "genotypes" are ordered lists of integers which code for selecting rules from the provided context-free grammar. The phenotype, however, is the same as in Koza-style GP: a tree-like structure that is evaluated recursively. This model is more in line with how genetics work in nature, where there is a separation between an organism's genotype and the final expression of phenotype in proteins, etc.



Separating genotype and phenotype allows a modular approach. In particular, the search portion of the GE paradigm needn't be carried out by any one particular algorithm or method. Observe that the objects GE performs search on are the same as those used in genetic algorithms. This means, in principle that any existing genetic algorithm package, such as the popular GAlib, can be used to carry out the search, and a developer implementing a GE system need only worry about carrying out the mapping from list of integers to program tree. It is also in principle possible to perform the search using some other method, such as particle

swarm optimization; the modular nature of GE creates many opportunities for hybrids as the problem of interest to be solved dictates.



Brabazon and O'Neill have successfully applied GE to predicting corporate bankruptcy, forecasting stock indices, bond credit ratings, and other financial applications.[citation needed] GE has also been used with a classic predator-prey model to explore the impact of parameters such as predator efficiency, niche number, and random mutations on ecological stability.

It is possible to structure a GE grammar that for a given function/terminal set is equivalent to genetic programming.

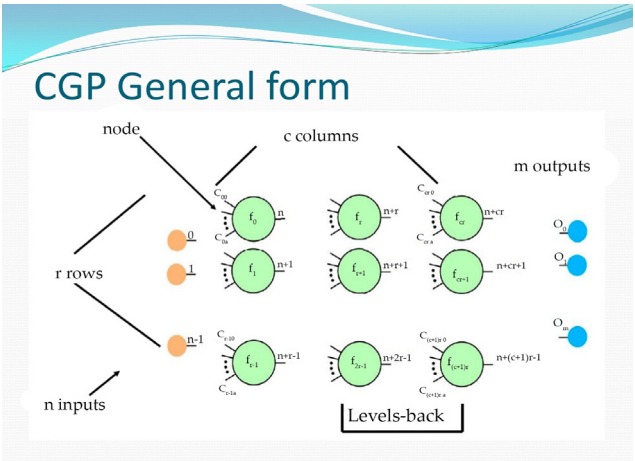
5.2.5 Cartesian Genetic Programming

In CGP, programs are represented in the form of directed acyclic graphs. These graphs are represented as a two-dimensional grid of computational nodes. The genes that make up the genotype in CGP are integers that represent where a node gets its data, what operations the node performs on the data and where the output data required by the user is to be obtained. When the genotype is decoded, some nodes may be ignored. This happens when node outputs are not used in the calculation of output data. When this happens, we refer to the nodes and their genes as 'non-coding'.

We call the program that results from the decoding of a genotype a phenotype. The genotype in CGP has a fixed length. However, the size of the phenotype (in terms of the number of computational nodes) can be anything from zero nodes to the number of nodes defined in the genotype.

A phenotype would have zero nodes if all the program outputs were directly connected to program inputs.

A phenotype would have the same number of nodes as defined in the genotype when every node in the graph was required. The genotype–phenotype mapping used in CGP is one of its defining characteristics.

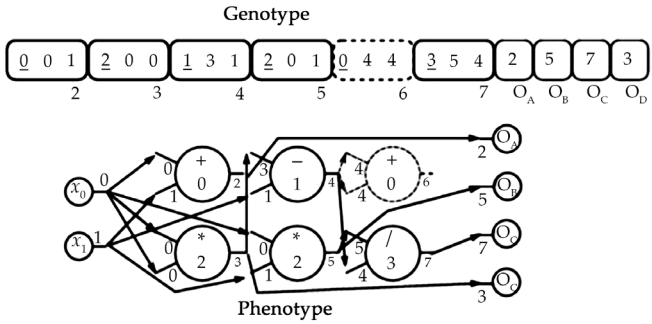


The types of computational node functions used in CGP are decided by the user and are listed in a function look-up table. In CGP, each node in the directed graph represents a particular function and is encoded by a number of genes.

One gene is the address of the computational node function in the function look-up table. We call this a function gene. The remaining node genes say where the node gets its data from.

These genes represent addresses in a data structure (typically an array). We call these connection genes. Nodes take their inputs in a feed-forward manner from either the output of nodes in a previous column or from a program input (which is sometimes

called a terminal). The number of connection genes a node has is chosen to be the maximum number of inputs (often called the arity) that any function in the function look-up table has.



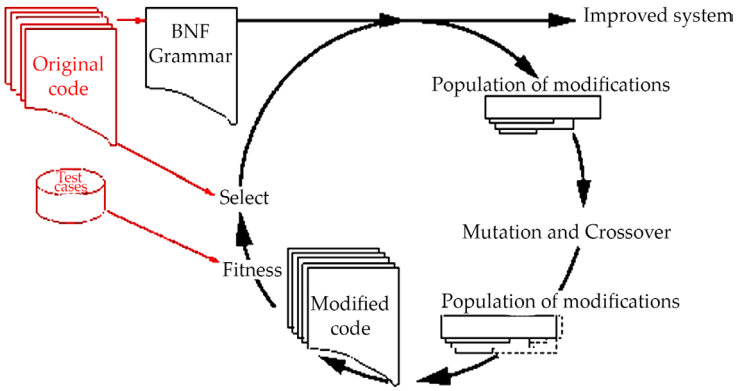
CGP represents computational structures (mathematical equations, circuits, computer programs etc.) as a string of integers. These integers, known as genes determine the functions of nodes in the graph, the connections between nodes, the connections to inputs and the locations in the graph where outputs are taken from.

Using a graph representation is very flexible as many computational structures can be represented as graphs. A good example of this is artificial neural networks (ANNs). These can be easily encoded in CGP.

5.2.6 Genetic Improvement Programming (GIP)

GIP evolves replacement software components that maximize achievement of multiple objectives, while retaining the interfaces between the components so-evolved and the surrounding system.

The GISMO project will develop theory, algorithms and techniques for GIP as a way to automatically optimize multiple software engineering objectives such as maximal throughput, fastest response time and most reliable performance, while minimizing power consumption, faults, memory use, compiled code size, peak disk usage and disk transfers.



The term component should be interpreted in its widest context. It refers to any piece of code that can be identified as a subpart of the overall program or system with well-defined interface to the encasing system. For example, we include functions, files, modules and procedures, for which interfaces are defined by parameters and shared global variables. We also include smaller segments of contiguous code that perform a coherent well-defined task or set of tasks, for which the interface is captured by the defined and referenced variables of the segment of code. What is important is that these pieces of code can be replaced by an evolved component that preserves their functionality and interface, while maximizing achievement of challenging new multiple objectives.

The Problem Addressed by GISMO

The emergent computing application paradigms require systems that are not only correct but are also optimized for many different competing non-functional requirements. Increasingly, we need to adapt existing systems to cater for operating environments with challenging non-functional properties. For instance, the migration from stand-alone systems to large scale distributed systems brings with it a need for optimization of non-functional properties such as response time and throughput. The increasing prevalence of smaller hand-held systems such as communications devices, raises the importance of non-functional properties such as power consumption and memory use.

Managing any one of these non-functional objectives is a challenge, but managing several at once is a daunting prospect, even for the most skilled and experience developer. The multiple objectives that have to be optimized are often conflicting. For instance, one can often trade speed of execution for compiled code size. Humans cannot be expected to optimally balance such competing constraints and may miss potentially valuable solutions.

The GISMO Solution

The GISMO solution rests on two core observations:

1. There is a wealth of relatively well-tested code upon which organizations already rely.
2. Evolutionary computation has proved able to balance many different competing and potentially conflicting criteria.

We therefore seek to use evolutionary computation, not to evolve entire systems but to replace components within existing systems with evolved replacements. The goal of the evolution will be to optimize for a new set of non-functional properties. These non-functional properties will be mapped into fitness functions that will guide evolution. The research challenge is to develop techniques that evolve components that balance these objectives, in a scalable way, while producing code that is useful and acceptable to the developer. The GISMO project will address the scalability issue using parallel computation. It will address the human acceptability issue using interactive evolution.

Why the Project Will be Highly Transformative

Genetic programming has proved to be good at evolving small code fragments for a single objective, while evolutionary computation has proved effective at solving multiple objective problems. The GISMO approach to scalability, human acceptance and multiple objectives are all entirely novel for genetic programming. If the project is even partly successful in its goal of automatically

finding GIP-evolved component replacements, this would be a major breakthrough. It would significantly increase our ability to migrate systems to challenging new operating environments and, simultaneously, dramatically reduce the cost of doing so.

Is it feasible?

The PI and named research fellow, Bill Langdon, have demonstrated the feasibility of the GIP approach. In our initial work we were able to port a critical component of Unix gzip utility to a CUDA platform [1]. The GISMO project will employ Dr. Langdon as a named research fellow for four years, in order to develop our new approach to software development.

GISMO Objectives

To achieve its aims, the GISMO project will:

1. Develop a theory of Genetic Interface Programming (GIP).
2. Develop techniques for parallel GIP computation for scalability and interactive GIP evolution for human involvement.
3. Develop new algorithms for achieving single and multiple objective GIP.
4. Evaluate qualitatively and quantitatively using benchmarks and real world systems from the industrial partners.

5.3 GENETIC PROGRAMMING: APPROACH IN MODELING WATER FLOWS

Like genetic algorithm (GA) the concept of Genetic Programming (GP) follows the principle of 'survival of the fittest' borrowed from the process of evolution occurring in nature. But unlike GA its solution is a computer program or an equation as against a

set of numbers in the GA and hence it is convenient to use the same as a regression tool rather than an optimization one like the GA. GP operates on parse trees rather than on bit strings as in a GA, to approximate the equation (in symbolic form) or computer program that best describes how the output relates to the input variables. A good explanation of various concepts related to GP. GP starts with a population of randomly generated computer programs on which computerized evolution process operates. Then a 'tournament' or competition is conducted by randomly selecting four programs from the population.

GP measures how each program performs the user designated task. The two programs that perform the task best 'win' the tournament.

GP algorithm then copies the two winner programs and transforms these copies into two new programs via crossover and mutation operators i.e. winners now have the 'children.'

These two new child programs are then inserted into the population of programs, replacing the two loser programs from the tournament. Crossover is inspired by the exchange of genetic material occurring in sexual reproduction in biology.

The creation of offspring's continues (in an iterative manner) till a specified number of offspring's in a generation are produced and further till another specified number of generations are created. The resulting offspring at the end of all this process (an equation or a computer program) is the solution of the problem. The GP thus transforms one population of individuals into another one in an iterative manner by following the natural genetic operations like reproduction, mutation and cross-over.

The tree based GP corresponds to the expressions (syntax trees) from a 'functional programming language'. In this type, Functions are located at the inner nodes; while leaves of the tree hold input values and constants.

A population of random trees representing the programs is initially constructed and genetic operations are performed on

these trees to generate individuals with the help of two distinct sets; the terminal set T and the function set F.

Population: These are the programs initially constructed from the data sets in the form of trees to perform genetic operations using Terminal set and Function set.

The function set for a run is comprised of operators to be used in evolving programs e.g. addition, subtraction, absolute value, logarithm, square root etc. The terminal set for a run is made up of the values on which the function set operates.

There can be four types of terminals namely inputs, constant, temporary variables, conditional flags. The population size is the number of programs in the population to be evolved. Larger population can solve more complicated problem. The maximum size of population depends upon RAM of the computer and length of programs in the population.

5.3.1 Genetic Operations

Cross over: Two individuals (programs) are chosen as per the fitness called parents. Two random nodes are selected from inside such program (parents) and thereafter the resultant sub-trees are swapped, generating two new programs. The resulting individuals are inserted into the new population. Individuals are increased by 2. The parents may be identical or different. The allowable range of cross over frequency parameter is 0 to 100%

Mutation: One individual is selected as per the fitness. A sub-tree is replaced by another one randomly. The mutant is inserted into the new population. Individuals are increased by 1. The allowable range of mutation frequency parameter is 0 to 100%

Reproduction: The best program is copied as it is as per the fitness criterion and included in the new population. Individuals are increased by 1. Reproduction rate = 100 – mutation rate – (crossover rate * [1 – mutation rate])

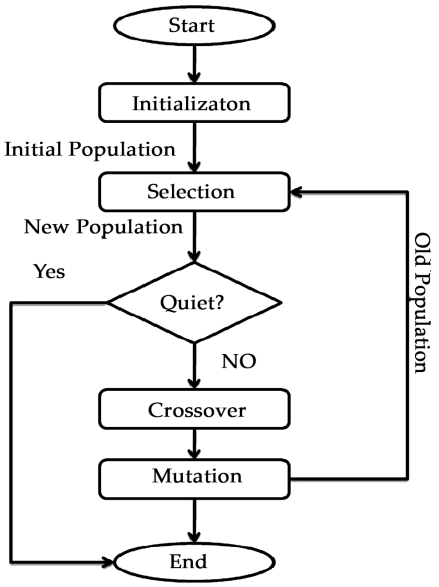


Figure 1. Flowchart of Genetic programming.

The second variant of GP is Linear genetic Programming (LGP) which uses a specific linear representation of computer programs. The name 'linear' refers to the structure of the (imperative) program representation only and does not stand for functional genetic programs that are restricted to a linear list of nodes only. On the contrary, it usually represents highly nonlinear solutions. Each individual (Program) in LGP is represented by a variable-length sequence of simple C language instructions, which operate on the registers or constants from predefined sets. The function setof the system can be composed of arithmetic operations (+, -, X, /), conditional branches, and function calls (f{x, x^n, sqrt, e^x,sin, cos, tan, log, ln }). Each function implicitly includes an assignment to a variable which facilitates use of multiple program outputs in LGP. LGP utilizes two-point string cross-over. A segment of random position and random length of an instruction is selected from each parents and exchanged. If one of the resulting children exceeds the maximum length, this cross-over is abandoned and restarted by exchanging equalized segments. An operand or operator of an instruction is changed by mutation into another symbol over the same set.

Gene-Expression Programming (GEP) is an extension of GP. The genome is encoded as linear chromosomes of fixed length, as in Genetic Algorithm (GA); however, in GEP the genes are then expressed as a phenotype in the form of expression trees. GEP combines the advantages of both its predecessors, GA and GP, and removes their limitations. GEP is a full-fledged genotype/phenotype system in which both are dealt with separately, whereas GP is a simple replicator system. As a consequence of this difference, the complete genotype/phenotype GEP system surpasses the older GP system by a factor of 100 to 60,000. In GEP, just like in other evolutionary methods, the process starts with the random generation of an initial population consisting of individual chromosomes of fixed length. The chromosomes may contain one or more than one genes. Each individual chromosome in the initial population is then expressed and its fitness is evaluated using one of the fitness function equations available in the literature. These chromosomes are then selected based on their fitness values using a roulette wheel selection process. Fitter chromosomes have greater chances of selection for passage to the next generation. After selection, these are reproduced with some modifications performed by the genetic operators. In Gene Expression Programming, genetic operators such as mutation, inversion, transposition and recombination are used for these modifications. Mutation is the most efficient genetic operator, and it is sometime used as the only means of modification. The new individuals are then subjected to the same process of modification, and the process continues until the maximum number of generations is reached or the required accuracy is achieved.

5.3.2 Use of GP in Water Flows Modeling

It is a known fact that many variables in the domain of Hydraulic Engineering are of random nature having a complex underlying phenomenon. For example the generation of ocean waves which are primarily functions of wind forcing is a very complex procedure. Forecasting of the ocean waves is an essential prerequisite for many ocean-coastal related activities. Traditionally this is done

using numerical models like WAM and SWAN. These models are extremely complex in development and application besides being highly computation-intensive. Further they are more useful for forecasting over a large spatial and temporal domain. The accuracy levels of wave forecasts obtained through such numerical models again leaves scope for exploration of alternative schemes. These numerical models suffer from disadvantages like requirement of exogenous data, complex modeling procedure, rounding off errors and large requirement of computer memory and time and there is no guarantee that the results will be accurate. Particularly when point forecasts were required the researchers therefore used the data driven techniques namely ARMA, ARIMA and since last two decades or so the soft computing technique of Neural Networks. A comprehensive review of applications of ANN in Ocean Engineering. Although wave forecasting models were developed using Artificial Neural Networks by many research workers there was scope for use of another data driven techniques in that the ANN based models generally were unable to forecast extreme events with reasonable accuracy and the accuracy of forecasts decreases with increase in lead time. This became an ideal situation for the entry of another soft computing tool of GP which functions in a completely different way than ANN in that it does not involve any transfer function and evolves generations and generations of 'offspring' based on the 'fitness criteria' and genetic operations.The rainfall -runoff modeling is very complex procedure and many numerical schemes are available as well as a large number of attempts by ANNs. Thus Genetic Programming entered in rainfall-runoff modeling. It was also found that GP results were superior to that of M5 Model Trees another data driven modeling technique. Apart from these two variables the use of GP for modeling for many hydraulic engineering processes was found necessary for similar reasons.

5.3.3 Applications in Ocean Engineering

Primarily the applications of GP in Ocean Engineering were found for modeling of oceanic parameters like waves, water levels,

zero cross wave periods, currents, wind, sediment transport and circular pile scour. One of the earlier applications was done to retrieve missing informationin wave records along the west coast. Such a need arises many times due to malfunctioning of instrument or drift of wave measuring buoy making it inoperative as a result of which data is not measured and it is lost forever. Filling up the missing significant wave height (Hs) values at a given location based on the same being collected at the nearby station(s) was done using GP. The wave heights were measured at an interval of 3 hours. Data at six locations around Indian coastline was used in this exercise. Out of the total sample size of four years the observations for the initial 25 months were used to evaluate the final or optimum GP program or equation while those for the last 23 months were employed to validate the performance and achieve gap in-filling with different quanta of missing information. It was found that both tree based and linear GP models worked in similar fashion as far as accuracy of estimation was considered.

When the similar work was also carried out using ANN it was found that GP produces results that are marginally more satisfactory than ANN. Another exercise was also carried out especially to estimate peaks by calibrating a separate model for high wave data which showed a marginal improvement in prediction of peaks. Albeit in altogether different area of Gulf of Mexico near the USA coastline. Gaps in hourly significant wave height records at one location were filled by using the significant wave heights at surrounding 3 locations at same time instant and the soft tool of GP and ANN. In all data spanning over 4 years was used for the study. The exercise was carried out for 4 locations in the Gulf of Mexico. The data can be downloaded from www.ndbc.noaa.gov. The typical value of the population size was 500, number of generations 15 and number of tournaments 90,00,000. The mutation and the cross-over frequency also varied for different testing exercises and it ranged from 20% to 80%. The fitness criterion was the mean squared error between actual observations and corresponding predictions.

The suitability of this approach was also tried for different gap lengths ranging from 1 day to 1 month and it was concluded

on the basis of 3 error measures that the accuracy of gap filling decreases with increase in the gap length. The accuracy of the results were also judged by calculating statistical parameters of the wave records without gaps filled and with gaps filled using GP model. When the gap lengths did not exceed 1 or 5 days all the four statistics were faithfully reproduced. Compared to ANN GP produced marginally better results. In both the cases Linear Genetic Programming technique was employed.

In another earlier works of GP current predictions over a time step of twenty minutes, one hour, 3 hours, 6 hours, 12 hours and 24 hours at 2 locations in the tidal dominated area of the Gulf of Khambhat along west coast of India was carried out using two soft techniques of ANN and GP and 2 hard techniques of traditional harmonic analysis and ARIMA. The work involved antecedent values of current only to forecast the current for various lead times at these locations. The fitness function selected was the mean square error, while the initial population size was 500, mutation frequency was 95%, and the crossover frequency was kept at 50%. For cross shore currents ARIMA performs better than ANN and GP even at longer prediction intervals. In general the three data driven techniques performed better than harmonic analysis. The new technique GP performed at par with ANN if not better. Perhaps the only drawback of the work was that the data (spanning over 7 months) is less than a year indicating that all possible variations in data set were not presented while calibrating the model making it susceptible when it is used at operational level.

Online wave forecasts over lead times of 3, 6, 12 and 24 hours were carried out at two locations in the gulf of Mexico using past values of wave heights (3 in number) and the soft computing technique of GP. The data measured from 1999 to 2004 was available for free download on the web site of National Buoy Centre. The locations chosen were differing to a large extent in that one was a deep water buoy and the other was a coastal buoy. The work was different from others in one aspect that monthly models were developed instead of routine yearly models. However any peculiar effect of this either good or bad on forecasting accuracy

was not evident from the 3 error measures calculated. Though the results of GP were promising (high correlation coefficients for 3 and 6 hr forecast) the forecasting accuracy decreased for longer lead times of 12 hr and 24 hr. It was found that the results of GP were superior to ANN. For GP model the initial population size was 500 while the number of generations was 300. The mutation frequency was 90 percent while the cross over frequency was 50 percent. Values of these control parameters were selected initially and thereafter varied in trials till the best fitness measures were produced.

The fitness criterion was the mean squared error between the actual and the predicted value of the significant wave height. Another exercise on real time forecasting of waves for warning times up to 72 hours at three locations along the Indian coastline using alternative techniques of ANN, GP and MT. The data was measured from 1998 to 2004 by the national data buoy program (www.niot.res.in). Forecasting waves up to 72hr and that too with reasonable accuracy is itself a specialty of this work. The data had many missing values which were filled by using temporal as well as spatial correlation approaches. Both MT and GP results were competitive with that of the ANN forecasts and hence the choice of a model should depend on the convenience of the user. The selected tools were able to forecast satisfactorily even up to a high lead time of 72 hrs. The significant wave height and average wave period at the current and subsequent 24 hr. lead time were predicted from continuous and past 24-hourly measurements of wind speeds and directions as well as two soft computing techniques of GP and MT. The data collected at 8 locations in Arabian Sea and Indian Ocean (www.niot.res.in) was used to develop both hind-casting and forecasting models. Both the methods, GP and MT, performed satisfactorily in the given task of wind wave simulation as reflected in high values of the error statistics of R, R^2, CE and low values of MAE, RMSE and SI. This is noteworthy since MT is not purely non-linear like GP. Although the magnitudes of these statistics did not indicate a significant difference in the relative performance of GP and MT, qualitative scatter diagrams and time histories showed the tendency of MT

to better estimate the higher waves. Forecasting at higher lead times were fairly accurate compared to the same at lower ones. In general the performance of wave period was less satisfactory than that of wave height and this can be expected in view of a highly varying nature of wave period values. Lately, extended their earlier work by forecasting Significant wave height and zero cross wave period over time intervals of 1 to 4 days using the current and previous values of wind velocity and wind direction at 2 locations around the Indian coastline. It was found out that best results were possible when the length of the input sequence matched with that of the output lead time. As observed earlier here also it was found that the accuracy of prediction decreases with increase in lead time. However the results were satisfactory for 4 days ahead predictions also. In general it was observed that results of MT were slightly inferior to that of GP. Separate models were also developed to account for the monsoon (rainfall season in India) which showed a considerable improvement over yearly models. The models calibrated at one location when applied for another nearby locations also shown satisfactory performance provided both sites have spatial homogeneity in terms of openness, long offshore distances and deep water conditions.

GP was used to forecast sea levels averaged over 12 h and 24 h time intervals for time periods from 12 to 120 h ahead at the Cocos (Keeling) Islands in the Indian Ocean. The model produced high quality predictions over all considered time periods. The presented results demonstrates the suitability of GP for learning the non-linear behavior of sea level variations in terms of the R^2(with values no lower than 0.968), MSE(with values generally smaller than 431) and MARE(no larger than 1.94%). This differs from earlier applications particularly for wave forecasting in that for forecasting of waves it was difficult to achieve higher order accuracy in terms of r, rmse and other error measures for as far as 24 hour forecast. Perhaps the recurring nature of sea water levels (the deterministic tidal component which is inherent in water level, is the reason behind this high level accuracy. In order to assess the ability of GP model relative to that of the ANN technique. The developed GP model was found to perform better than the

used ANNs. In the current work, the linear genetic programming approach was employed. The water level at Hillary's Boat Harbor, Australia was predicted three time steps ahead using time series averaged over 12hr, 24hr, 5 day and 10 day time interval and the soft tool of GP. The results are compared with ANN. Total 12 years of data was used out of which 3 years of data is used for model validation. Tree based GP was used. The results of 12 hr averaged input data were found to be better than 24 hr averaged input data and in general the accuracy of prediction reduced for higher lead times. For both the cases GP results were better than ANN. For 5 day averaged inputs performance of GP was inferior to that of ANN though it improved for 10 day averaged inputs. It may be noted that the input data is averaged over 12hr, 24hr, 5days and 10 days which means there is possibility of loss of information which can be major drawback of this work. For both the above works the hourly sea-level records from a SEA-level Fine Resolutions Acoustic Measuring Equipment (SEA-FRAME) station were used. The information about initial parameters of GP is however not mentioned in both the works.

Estimation of wind speed and wind direction using the significant wave height, zero cross wave period, average wave period and the soft tools of ANN and GP was carried out at 5 locations around Indian coastline. The first attempt both ANN and GP were tried for estimating the wind speed in which GP was found better and therefore in the second fold GP was only used to determine both wind speed and direction by calibrating the model by splitting of wind vector into two components. Two variants of GP, one based on Tree based approach and the other on Linear Genetic Programming were also tried though the accuracy of estimation for both the approaches was at par. In the third fold a network of wave buoys were formed and wind direction and wind speed at one location was estimated using the same at other locations. This was also done by combining data of all locations and making a regional model. All the attempts yielded highly satisfactory results as far as accuracy of estimation is considered. It was also confirmed that for estimation of only wind speed the non-splitting of wind velocity gives better results. Similarly wind speed and

its directions were predicted for intervals of 3hr, 6hr, 9hr, 12hr and 24 hr at locations along the west coast of India using two soft computing techniques of ANN and GP and previous values of the same. It was found that GP rivaled ANN predictions at all the cases and even bettered it particularly for open sea location. The results for prediction of wind speed and wind direction together were better when training of GP and ANN models was done on the basis of splitting of wind vector into two components along orthogonal directions although a separate model for wind speed alone was better. In general long interval predictions were less accurate compared to short interval predictions for both the techniques. Data for one location was for about 1.5 years while for the other location it was for 3 years. A similar work was carried out to estimate the wind speed at 5 locations around the Indian coastline using the wave parameters and 3 data driven techniques namely GP (program based- tree type), MT and another data driven tool of locally weighted projection regression (LWPR) by. All models showed tendency to underestimate higher values in given records. When all of the eight error statistics employed were viewed together, no single method appeared distinctly superior to others, but the use of an average evaluation index EI which they have suggested in this work gave equal weightage to each measure showed that the GP was more acceptable than other methods in carrying out the intended inverse modeling. Separate GP models were developed to estimate higher wind speeds that may be encountered in stormy conditions. At all the locations, these models indicated satisfactory performance of GP although with a fall in accuracy with increase in randomness. For all the above works the data was measured by national data buoy program of India however no mention is made about the initial parameters chosen for GP implementation.

The estimation of longshore sediment transport rate at an Indian location was carried out using GP and combined GP-ANN models. The inputs were significant wave height, zero cross wave period, breaking wave height, breaking wave angle and surf zone width. The limitation of the work was the amount of data (81) used for training and testing of the models. The choice of control

parameters was as follows: initial population size = 500; mutation frequency = 95%; crossover frequency = 50%. The initial trial with GP yielded reasonable results (r = 0.87). However by first training the ANN with same inputs and using the output as input for GP model yielded better results (r = 0.92). It may be noted this is a kind of work done in the domain of Ocean Engineering wherein a different parameter (sediment transport rate) is modeled rather than the usual parameters of waves, periods etc. Another different work was carried out by, for prediction of scour depth due to ocean/lake waves around a pile/pier in medium dense silt and sand bed using Linear Genetic Programming and Adaptive Neuro-Fuzzy Inference system and measured laboratory data. The study was carried out in both dimensional and non-dimensional form in which non-dimensional form yielded better results. The relative importance of input parameters on scour process was also investigated by first using all the influential parameters as inputs and then removing them one by one and observing the results. The drawback of the work is perhaps the small number of data used in model making (total 38 data, 28 of which is used for training the model) which may be impediment in operational use of this model. The results were found to be superior to ANFIS results.

In all the above cases where GP is compared with another data driven technique like ANN, MT or LWPR it was found that GP is superior to all of them in terms of accuracy of results. However it can be said that GP needs to be explored further particularly for prediction of extreme events like water levels, wave heights during hurricanes. A detailed study on effect of variation of GP control parameters like initial population, mutation, crossover percentage etc. on model accuracy is now need of the day. Similarly the critic on other approaches about decreasing forecasting accuracy with increase in the lead time seems to be true for GP as well. This needs more attention if GP is here to stay.

5.3.4 Applications in Hydrology

Genetic Programming is used in Hydrology (science of water) for various purposes such as modeling of phenomena like rainfall-

runoff process, evapo-transpiration, flood routing, stage-discharge curve. The GP approach was applied to the flow prediction of the Kirkton catchment in Scotland (U.K.). The results obtained were compared to those attained using optimally calibrated conceptual models and an ANN. The data sets selected for the modeling process were rainfall, streamflow and Penman open water evaporation. The data used for calibration was of 610 days while that of validation was of 1705 days. The models were developed with preceding values of rainfall, evaporation and stream flow for predicting stream flow one time step ahead. Two conceptual models as well as ANN were employed for developing the stream flow forecasting model. It was observed that the rainfall data was the most influencing factor on the output. All models performed well in terms of forecasting accuracy with GP performing better. In another work one day ahead forecasting of runoff knowing the rainfall and runoff of the previous days and the soft computing tool of Linear Genetic Programming was carried out in Lindenborg catchment of Denmark by. The models were developed for forecasting runoff as well as variation of runoff by using previous values of variation of discharge as input as well as previous values of discharge as input along with rainfall information. It was found that it was necessary to include information of discharge rather than variation of discharge. The model predicting discharge gave wrong local peaks in the low regime where as models predicting variation of discharge gave less wrong peaks in the low flow. Both the models had difficulty in predicting high peaks. The models were also developed using ANN. No specific information is provided about the initial values of GP parameters. The results obtained with a deterministic lumped parameter model, based on the unit hydrograph approach were compared with those obtained using a stochastic machine learning model of GP. For the Welsh catchment in UK, the results between the two models were similar. Since rainfall and runoff were highly correlated, the deterministic assumption underlying the IHACRES model (deterministic) was satisfied. Therefore, IHACREX could achieve a satisfactory correlation between calibration and simulation data. The GP approach which did not require any causal relationships achieved similar results. The behavior of the studied Australian catchment

is very different from the Welsh catchment. The runoff ratio was very low (7%), and hence, the a priori assumptions of IHACRES (and other deterministic models) were a poor representation of the real world. This was demonstrated by the inability of IHACREJS to use more than one season's data for calibration purposes and only able to use data from a high rainfall period. Since the GP approach did not make any assumptions about the underlying physical processes, calibration periods over more than one season could be used. These led to significantly improved generalizations for the modeled behavior of the catchment. In summary, either approach worked satisfactorily when rainfall and runoff were correlated. However, when this correlation was poor, the CFG-GP had some advantages because it did not assume any underlying relationships. This is particularly important when considering the modeling of environmental problems, where typically the relationships are nonlinear, and are often measured at a scale which does not match with conceptual or deterministic modeling assumptions. In their work of GP in hydrology, [30] first used a simple example of the Bernoulli equation to illustrate how GP symbolically regresses or infers the relationship between the input and output variables. An important conclusion from this study was that non-dimensionalizing the variables prior to symbolic regression process significantly enhance the success of GSR (Genetic Symbolic Regression). GP was then applied to the problem of real-time runoff forecasting for the Orgeval catchment in France. GP functions as an error updating procedure complementing the rainfall-runoff model, MIKE11/ NAM. Ten storm events were used to infer the relationship between the NAM simulated runoff and the corresponding prediction error. That relationship was subsequently used for real-time forecasting of six storm events. The results indicated that the proposed methodology was able to forecast different storm events with great accuracy for different updating intervals. The forecast hydrograph performs well even for a long forecast horizon of up to nine hours. However, it was found that for practical applications in real-time runoff forecasting, the updating interval should be less than or equal to the time of concentration of the catchment. The results were also compared with two known updating methods such as

the auto-regression and Kalman filter. Comparisons showed that the proposed scheme, NAM-GSR, is comparable to these methods for real time runoff forecasting. The rainfall-runoff models were created on the basis of data alone as well as in combination with conceptual models and Genetic Programming. The study was carried out in Orgeval catchment of France having an area about 104 km^2 using hourly rainfall runoff data of 10 storms for calibration and 6 storms for testing the models. The models were calibrated to forecast the temporal difference between the current and future discharge rather than absolute value of discharge for the lead times of 1 to 12 hours. The results were superior to conceptual numerical model. The model was then calibrated using a hybrid method in that the surface runoff value was first forecasted by using a conceptual forecasting model and then using the simulation error and GP to forecast the stream flow. The hybrid models provided a many fold improvement over the raw GP models. Additionally the models were developed using multilayer perceprton as well as Generalized Regression Neural Networks (GRNN). The statistical ARMA method was also used to develop the stream flow forecasting model. The results showed that both LGP and NN techniques predicted the daily time series of discharge with quite good agreement as indicated by high value of coefficient of determination and low values of error measures with the observed data. LGP models generally predicted the maximum and minimum discharge values better than the NN models though LGP results were also far from accurate. The robustness of the developed models was tested by using applied data which was neither used in training or testing and the results were judged using Akaike Information Criterion (AIC).

The potential of the GP-based model for flood routing between two river gauging stations on river Walla in USA was explored for single peaked as well as multi-peaked flood hydrographs. The accuracy of GP models was far superior than modified Muskingum method which is a traditional physics based hydrologic flood routing model which also showed time lag in predictions. The inputs were current and antecedent discharge at upstream station and antecedent discharge at downstream station while

the output was current discharge at the downstream station. The LGP was employed for the flood routing exercise. The optimal GP parameters used in this study were: crossover rate, 0.9; mutation rate, 0.5; population size, 200; number of generations, 500; and functional set, i.e. simple arithmetic functions (plus, minus, multiply, divide).

The utility of genetic programming in modeling the eddy-covariance (EC) measured evapo-transpiration flux was investigated. The performance of the GP technique was compared with artificial neural network and Penman-Monteith model estimates. EC measured evapo-transpiration fluxes from two distinct case-studies with different climatic and topographic conditions were considered for the analysis and latent heat is modeled as a function of net radiation, ground temperature, air temperature, wind speed and relative humidity. Results from the study indicated that both data-driven models (ANN and GP) performed better than the Penman-Monteith method. However, the performance of the GP model is comparable with that of ANN models. One of the important advantages of employing GP to model evapo-transpiration process is that, unlike the ANN model, GP resulted in an explicit model structure that can be easily comprehended and adopted. Another advantage of GP over ANN was found that unlike ANN, GP can evolve its own model structure with relevant inputs reducing the tedious task of identifying optimal input combinations. This work was extended by [34] where in an additional data driven tool of Evolutionary Polynomial Regression was used to model the evapo-transpiration process. Additionally the effect of previous states of input variable (lags) on modeling the EC measured AET (actual evapo-transpiration) is investigated. The evapo-transpiration is estimated using the environmental variables such as net radiation (NR), ground temperature (GT), air temperature (AT), wind speed (WS) and relative humidity (RH). It has been found out that random search and evolutionary-based techniques, such as GP and EPR techniques, do not guarantee consistent performance in all case studies e.g. good and/or bad performance for modelling AET. The results of ANN, GP and EPR were mostly at par with each other though EPR models were

easier to understand. Recently the stage –discharge relationship for the Pahang River in Malaysia was modeled using Genetic Programming (GP) and Gene Expression Programming (GEP). The data was provided by Malaysian Department of Irrigation and Drainage (DID). Gene Expression Programming is an extension of GP. GEP is a full-fledged genotype/phenotype system in which both are dealt with separately, whereas GP is a simple replicator system. Stage and discharge data from 2 years were used to compare the performance of the GP and GEP models against that of the more conventional (stage-rating curve) SRC and (Regression) REG approaches. The GEP model was found to be considerably better than the conventional SRC, REG and GP models. GEP was also relatively more successful than GP, especially in estimating large discharge values during flood events.

REFERENCES

1. Altenberg L (2009) Modularity in evolution: Some low-level questions. In: Rasskin-Gutman D, Callebaut W (eds), Modularity: Understanding the Development and Evolution of Complex Natural Systems. MIT Press, Cambridge, MA
2. Alves da Silva AP, Abrao PJ (2002) Applications of evolutionary computation in electric power systems. In: Fogel DB et al. (eds), Proceedings of the 2002 Congress on Evolutionary Computation CEC2002, pp. 1057–1062. IEEE Press
3. Archetti F, Messina E, Lanzeni S, Vanneschi L (2007) Genetic programming for computational pharmacokinetics in drug discovery and development. Genet Programming Evol Mach 8(4):17–26
4. Azaria Y, Sipper M (2005) GP-gammon: Genetically programming backgammon players. Genet Programming Evol Mach 6(3):283–300, Sept. Published online: 12 August 2005
5. Barrett SJ, Langdon WB (2006) Advances in the application of machine learning techniques in drug discovery, design and development. In: Tiwari A et al. (eds), Applications of Soft Computing: Recent Trends, Advances in Soft Computing, On the World Wide Web, 19 Sept.–7 Oct. 2005. Springer, Berlin, 99–110
6. Bojarczuk CC, Lopes HS, Freitas AA (July–Aug. 2008) Genetic programming for knowledge discovery in chest-pain diagnosis. IEEE Eng Med Biol Mag 19(4):38–44
7. Brameier M, Banzhaf W (2001) A comparison of linear genetic programming and neural networks in medical data mining. IEEE Trans Evol Comput 5(1):17–26
8. Cagnoni S, Rivero D, Vanneschi L (2005) A purely-evolutionary memetic algorithm as a first step towards symbiotic coevolution. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland, 2005. IEEE Press, Piscataway, NJ. pp. 1156–1163

9. Castillo F, Kordon A, Smits G (2006) Robust pareto front genetic programming parameter selection based on design of experiments and industrial data. In: Riolo RL, et al. (ed) Genetic Programming Theory and Practice IV, vol 5 of Genetic and Evolutionary Computation, chapter 2. Springer, Ann Arbor, 11–13 May
10. Chen S-H, Liao C-C (2005) Agent-based computational modeling of the stock price-volume relation. Inf Sci 170(1):75–100, 18 Feb
11. Da Costa LE, Landry JA (2006) Relaxed genetic programming. In: Keijzer M et al., editor, GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, vol 1, Seattle, Washington, DC, 8–12 July. ACM Press pp. 937–938
12. Dassau E, Grosman B, Lewin DR (2006) Modeling and temperature control of rapid thermal processing. Comput Chem Eng 30(4):686–697, 15 Feb
13. Dempsey I (2007) Grammatical evolution in dynamic environments. Ph.D. thesis, University College Dublin, Ireland
14. Dignum S, Poli R (2007) Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In: Thierens, D et al. (eds), GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, vol 2 London, 7–11 July 2007. ACM Press, pp. 1588–1595
15. Eiben AE, Jelasity M (2002) A critical note on experimental research methodology in EC. In: Congress on Evolutionary Computation (CEC'02), Honolulu, HI, 2002. IEEE Press, Piscataway, NJ, pp. 582–587
16. Esparcia-Alcazar AI, Sharman KC (Sept. 1996) Genetic programming techniques that evolve recurrent neural networks architectures for signal processing. In: IEEE Workshop on Neural Networks for Signal Processing, Seiko, Kyoto, Japan

17. Fernandez F, Martin A (2004) Saving effort in parallel GP by means of plagues. In: Keijzer M, et al. (eds), Genetic Programming 7th European Conference, EuroGP 2004, Proceedings, vol 3003 of LNCS, Coimbra, Portugal, 5–7 Apr. Springer-Verlag, pp. 269–278

CHAPTER 6

MEMETIC ALGORITHMS

INTRODUCTION

Memetic algorithm (MA) is an extension of the traditional genetic algorithm. It uses a local search technique to reduce the likelihood of the premature convergence. Memetic algorithms represent one of the recent growing areas of research in evolutionary computation. The term MA is now widely used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. Memetic algorithms address the difficulty of developing high-performance universal heuristics by encouraging the exploitation of multiple heuristics acting in concert, making use of all available sources of information for a problem. This approach has resulted in a rich arsenal of heuristic algorithms and metaheuristic frameworks for many problems.

6.1 BASIC CONCEPT OF MEMETIC ALGORITHM

The generic denomination of 'Memetic Algorithms' (MAs) is used to encompass a broad class of metaheuristics (i.e. general purpose methods aimed to guide an underlying heuristic). The method is based on a population of agents and proved to be of practical success in a variety of problem domains and in particular for the approximate solution of NP-hard optimization problems.

Unlike traditional evolutionary computation (EC) methods, MAs are intrinsically concerned with exploiting all available knowledge about the problem under study. The incorporation of problem domain knowledge is not an optional mechanism, but a fundamental feature that characterizes MAs. This functioning philosophy is perfectly illustrated by the term "memetic".

In contrast, it is processed and enhanced by the communicating parts. This enhancement is accomplished in MAs by incorporating heuristics, approximation algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. In essence, most MAs can be interpreted as a search strategy in which a population of optimizing agents cooperate and compete. The success of MAs can probably be explained as being a direct consequence of the synergy of the different search approaches they incorporate.

The most crucial and distinctive feature of MAs, the inclusion of problem knowledge mentioned above, is also supported by strong theoretical results. As Hart and Belew initially stated and Wolpert and Macready later popularized in the so-called No-Free-Lunch Theorem, a search algorithm strictly performs in accordance with the amount and quality of the problem knowledge they incorporate. This fact clearly underpins the exploitation of problem knowledge intrinsic to MAs. Given that the term hybridization is often used to denote the process of incorporating problem knowledge, it is not surprising that MAs are sometimes called 'Hybrid Evolutionary Algorithms' (hybrid EAs) as well. One of the first algorithms to which the MA label was assigned dates from 1988, and was

regarded by many as a hybrid of traditional Genetic Algorithms (GAs) and Simulated Annealing (SA). Part of the initial motivation was to find a way out of the limitations of both techniques on a well-studied combinatorial optimization problem the Min Euclidean Traveling Salesman problem (Min ETSP). According to the authors, the original inspiration came from computer game tournaments used to study "the evolution of cooperation". That approach had several features which anticipated many current algorithms in practice today. The competitive phase of the algorithm was based on the new allocation of search points in configuration phase, a process involving a "battle" for survival followed by the so-called "cloning", which has a strong similarity with 'go with the winners' algorithms. The cooperative phase followed by local search may be better named "go-with-the-local-winners" since the optimizing agents were arranged with a topology of a two dimensional toroidal lattice. After initial computer experiments, an insight was derived on the particular relevance that the "spatial" organization, when coupled with an appropriate set of rules, had for the overall performance of population search processes. A few months later, Moscato and Norman discovered that they shared similar views with other researchers and other authors proposing "island models" for GAs. Spacialization is now being recognized as the "catalyzer" responsible of a variety of phenomena.

6.1.1 Basic Model of a Memetic Algorithm

Figure 1 shows the block diagram of a basic population metaheuristic, indicating the four points where a local search can be included in order to form a MA:

1. On the population, to simulate the cultural development that will be transmitted from one generation to another; it can be applied to the whole set of agents or to specific elements, and even to the initial group.
2. On the parent or selected parents, before reproduction stage.
3. When new solutions are generated, to produce a better offspring.

4. On the offspring, before selecting a survivor according to fitness criteria.

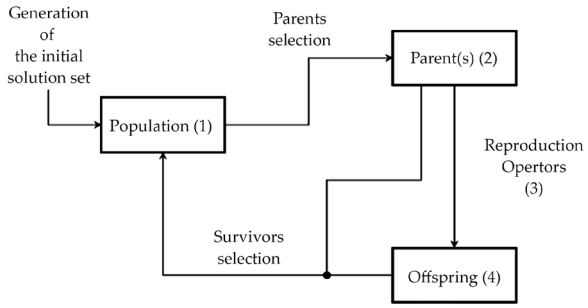


Figure 1. Block diagram of a basic population metaheuristic.

From this basic model several versions of MAs have been developed, differing between each other in at least one of the following aspects:

- Population metaheuristic used as a base.
- Selected algorithm for local search (exact method or metaheuristic, number of memes to consider.)
- Conditions for local search (trigger event, frequency, intensity, number of individuals to improve, etc.)

6.1.2 The Development of MAs

1st Generation

The first generation of MA refers to hybrid algorithms, a marriage between a population-based global search (often in the form of an evolutionary algorithm) coupled with a cultural evolutionary stage. This first generation of MA although encompasses characteristics of cultural evolution (in the form of local refinement) in the search cycle, it may not qualify as a true evolving system according to universal Darwinism, since all the core principles of inheritance/memetic transmission, variation, and selection are missing. This suggests why the term MA stirred up criticisms and controversies among researchers when first introduced.

Pseudo code

Procedure Memetic Algorithm

Initialize: Generate an initial population;

while Stopping conditions are not satisfied do

Evaluate all individuals in the population.

Evolve a new population using stochastic search operators.

Select the subset of individuals, Ω_{il}, that should undergo the individual improvement procedure.

for each individual in Ω_{il} do

Perform individual learning using meme(s) with frequency or probability of f_{il}, for a period of t_{il}.

Proceed with Lamarckian or Baldwinian learning.

end for

end while

2nd Generation

Multi-meme, hyper-heuristic and meta-Lamarckian MA are referred to as second generation MA exhibiting the principles of memetic transmission and selection in their design. In Multi-meme MA, the memetic material is encoded as part of the genotype. Subsequently, the decoded meme of each respective individual/ chromosome is then used to perform a local refinement. The memetic material is then transmitted through a simple inheritance mechanism from parent to offspring(s). On the other hand, in hyper-heuristic and meta-Lamarckian MA, the pool of candidate memes considered will compete, based on their past merits in generating local improvements through a reward mechanism, deciding on which meme to be selected to proceed for future local refinements. Memes with a higher reward have a greater chance of being replicated or copied. For a review on second generation MA; i.e., MA considering multiple individual learning methods within an evolutionary system, the reader is referred to.

3rd Generatio

Co-evolutionand self-generating MAs may be regarded as 3rd generation MA where all three principles satisfying the definitions of a basic evolving system have been considered. In contrast to 2nd generation MA which assumes that the memes to be used are known a priori, 3rd generation MA utilizes a rule-based local search to supplement candidate solutions within the evolutionary system, thus capturing regularly repeated features or patterns in the problem space.

6.1.3 The Need for Memetic Algorithms

In order to understand in depth the role and need of MAs, it is fundamental to consider the historical context within which MAs have been defined. In 1988, when the first MAs were defined, Genetic Algorithms (GAs) were extremely popular among computer scientists and their related research was oriented towards the design of algorithms having a superior performance with respect to all the other algorithms. Unlike all the algorithms proposed at that time, a MA was not a specific algorithm but was something much more general than an optimization algorithm: since MAs consists of the concept of combining global and local search algorithms, they represented a broad and flexible class of algorithms which somehow contained the previous work on Evolutionary Algorithms (EAs) and thus, constituted a new philosophy in optimization. Probably, due to their excessively innovative contents, MAs had to face for about one decade, the skepticism of the scientific community which repeatedly rejected the memetic approach as a valuable possibility in optimization.

Since 1997, researchers in optimization had to dramatically change their view about the subject. More specifically, in the light of increasing interest in general purpose optimization algorithms, it has become important, in the end of 90's to understand the relationship between how well an algorithm a performs on a given optimization problem f on which it is run on the basis of the

features of the problem f . A slightly counter intuitive result has been derived by Wolpert and Macready in which states that for a given pair of algorithms A and B:

$$\sum_f P(x_m|f,A) = \sum_f P(x_m|f,B)$$

where $P(x_m | f,A)$ is the probability that algorithm A detects the optimal solution for a generic objective function f and $P(x_m | f,B)$ is the analogue probability for algorithm B. The statement is proved for both static and time dependent case and are named "No Free Lunch Theorems" (NFLT). In other words, in 1997 it was mathematically proved that the average performance of any pair of algorithms across all possible problems is identical. Thus, if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems as this is the only way that all algorithms can have the same performance averaged over all functions. Strictly speaking, the proof of NFLT is made under the hypothesis that both the algorithms A and B are non-revisiting, i.e. the algorithms do not perform the fitness evaluation of the same candidate solution more often than once during the optimization run. Although this hypothesis is de facto not respected for most of the computational intelligence optimization algorithms, the concept that there is no universal optimizer had a significant impact on the scientific community.

It should be highlighted that a class of problems on which an algorithms performs well is not defined by the nature of the application but rather by the features of the fitness function within the search space. For example an optimization problem is characterized by:

- The shape and properties of a corresponding fitness landscape,
- Multi-modality,
- Separability of the problem,
- Absence or presence of a noise in the values of the objective function (optionally, the type of noise),

- Time dependency of the objective function (dynamic problems)
- Shape and connectivity of the search domain

More formally, the fitness landscape (S, f, d) of a problem instance for a given problem consists of a set of points S, a fitness function f which assigns values (fitness) to solutions from S, and a distance measure $d : S \times S \rightarrow R$ which defines the spacial structure of the landscape. This rather abstract concept has proven to be useful for understanding the functionality of various optimization methods.

One of the most important properties of the fitness landscape is epistasis whose concept has been borrowed from biology where it refers to the degree to which the genes are correlated. As well known, a function is separable if it can be rewritten as a sum of functions of just one variable. The separability is closely related to the concept of epistasis. In the field of evolutionary computation, the epistasis measures how much the contribution of a gene to the fitness of the individual depends on the values of other genes. Nonseparable functions are more difficult to optimize as the accurate search direction depends on two or more genes. On the other hand, separable functions can be optimized for each variable in turn. However, epistasis does not provide any piece of information on how the fitness values are topologically related to each other. By knowing the epistasis of an optimization problem, it cannot be established whether the fitness values form a smooth progression resulting in a solitary optimum or whether they form a spiky pattern of many isolated optima.

The impossibility of understanding each detail of the fitness landscape depends not only on the fitness function but also on the search algorithm since an observed landscape appears to be an artefact of the algorithm used or, more specifically, of the neighborhood structure induced by the operators used by the algorithm. The neighborhood structure is defined as a set of points that can be reached by a single move of a search algorithm. Closely related to the concept of the neighborhood structure is the notion of a basin of attraction induced by this structure. More specifically, a basin of attraction of a local optimum x is the set of

points X of the search space such that a search algorithm starting from any point from X ends in the local optimum x. A special note should be made regarding the landscapes with plateaus, i.e. regions in search domain where the function has constant or nearly constant values. If a search method is trapped on such region it cannot get any information regarding the gradient or even its estimates. Generally speaking, this situation is rather complicated and special algorithmic components should be used in this case. Finally, an important feature of a fitness landscape is the presence or absence of symmetry. Special components can be included in the algorithms for symmetrical problems.

In addition, two features can be mentioned which appear to be semi-defining when distinguishing the classes of problems on which an algorithm performs well. The first one is dimensionality of the problem. Two problems with high dimensionality of the search domain can be put into the same class, however an algorithm that performs well for one of them might not necessarily work well for the other one. At the same time, two specialized algorithms for these two problems will have some common features intended to overcome difficulties arising from high dimensionality. The second semi-defining feature is computational cost of a single evaluation of the objective function. Clearly, two problems with computationally expensive objective functions can have different features mentioned above that will put them into different classes. However, these problems are unsolvable (in practice) if treated as computationally cheap functions, therefore algorithms for such problems should have common type components which allow proper handling of the computational cost.

There is generally a performance advantage in incorporating prior knowledge into the algorithm, however the results of NFLT do not deem the use of unspecialized algorithms futile. It is impossible to determine the fraction of practical problems for which an algorithm yields good results rapidly, therefore a practical free lunch is possible. NFLT constitute, in a certain sense, the "Full Employment Theorem" (FET) for optimization professionals. In computer science and mathematics, the term FET is used to refer

to a theorem that shows that no algorithm can optimally perform a particular task done by some class of professionals. In this sense, as no efficient general purpose solver exists, there is always scope for improving algorithms for better performance on particular problems. Since MAs, as mentioned above, represent a broad class of algorithms which combine various algorithmic components, a suitable combination is necessary for a given problem. Since, during the last decade, computer scientists had to observe the features of their optimization problem in order to propose an ad-hoc optimization algorithm, the approach of combining various search operators within the algorithmic design became a common practice. The development of NFLT implicitly encouraged the use and development of MAs, which became extremely popular and often necessary, in computer science, at first, and in engineering and applied science, more recently, thus constituting the FET for MAs.

6.1.4 Recombination

Local search is based on the application of a mutation operator to a single configuration. Despite the apparent simplicity of this mechanism, "mutation-based" local search has revealed itself a very powerful mechanism for obtaining good quality solutions for NP−hard problems. For this reason, some researchers have tried to provide a more theoretically-solid background to this class of search. It is worth mentioning the definition of the Polynomial Local Search class (PLS) by Johnson et al. Basically, this complexity class comprises a problem and an associated search landscape such that we can decide in polynomial time if we can find a better solution in the neighborhood. Unfortunately, it is very likely that no NP−hard problem is contained in class PLS, since that would imply that NP=co-NP, a conjecture usually assumed to be false. This fact has justified the quest for additional search mechanisms to be used as stand-alone operators or as complements to standard mutation.

Recall that population-based search allowed the definition of generalized move operators termed recombination operators. In

essence, recombination can be defined as a process in which a set S_{par} of n configurations (informally referred to as "parents") is manipulated to create a set $S_{desc} \circledcirc sol_P(x)$ of m new configurations (informally termed "descendants"). The creation of these descendants involves the identification and combination of features extracted from the parents.

At this point, it is possible to consider properties of interest that can be exhibited by recombination operators. The first property, respect, represents the exploitation side of recombination. A recombination operator is said to be respectful, regarding a particular type of features of the configurations, if, and only if, it generates descendants carrying all basic features common to all parents. Notice that, if all parent configurations are identical, a respectful recombination operator is forced to return the same configuration as a descendant. This property is termed purity, and can be achieved even when the recombination operator is not generally respectful.

On the other hand, assortment represents the exploratory side of recombination. A recombination operator is said to be properly assorting if, and only if, it can generate descendants carrying any combination of compatible features taken from the parents. The assortment is said to be weak if it is necessary to perform several recombinations within the offspring to achieve this effect.

Finally, transmission is a very important property that captures the intuitive role of recombination. An operator is said to be transmitting if every feature exhibited by the offspring is present in at least one of the parents. Thus, a transmitting recombination operator combines the information present in the parents but does not introduce new information. This latter task is usually left to the mutation operator. For this reason, a non-transmitting recombination operator is said to introduce implicit mutation.

The three properties above suffice to describe the abstract input/output behavior of a recombination operator regarding some particular features. It provides a characterization of the possible descendants that can be produced by the operator. Nevertheless,

there exist other aspects of the functioning of recombination that must be studied. In particular, it is interesting to consider how the construction of S_{desc} is approached.

First of all, a recombination operator is said to be blind if it has no other input than S_{par}, i.e., it does not use any information from the problem instance. This definition is certainly very restrictive, and hence is sometimes relaxed as to allow the recombination operator to use information regarding the problem constraints (so as to construct feasible descendants), and possibly the fitness values of configurations $y \in S_{par}$ (so as to bias the generation of descendants toward the best parents). A typical example of a blind recombination operator is the classical Uniform crossover. This operator is defined on search spaces $S \equiv \Sigma^n$, i.e., strings of n symbols taken from an alphabet Σ. The construction of the descendant is done by randomly selecting at each position one of the symbols appearing in that position in any of the parents. This random selection can be totally uniform or can be biased according to the fitness values of the parents as mentioned before. Furthermore, the selection can be done so as to enforce feasibility (e.g., consider the binary representation of solutions in the 0-1 MKP). Notice that, the resulting operator is neither respectful nor transmitting in general.

The use of blind recombination operators has been usually justified on the grounds of not introducing excessive bias in the search algorithm, thus preventing extremely fast convergence to suboptimal solutions. This is questionable though. First, notice that the behavior of the algorithm is in fact biased by the choice of representation and the mechanics of the particular operators. Second, there exist widely known mechanisms (e.g., spatial isolation) to hinder these problems. Finally, it can be better to quickly obtain a suboptimal solution and restart the algorithm than using blind operators for a long time in pursuit of an asymptotically optimal behavior.

Recombination operators that use problem knowledge are commonly termed heuristic or hybrid. In these operators, problem information is utilized to guide the process of constructing the

descendants. This can be done in a plethora of ways for each problem, so it is difficult to provide a taxonomy of heuristic recombination operators. Nevertheless, there exist two main aspects into which problem knowledge can be injected: the selection of the parental features that will be transmitted to the descendant, and the selection of nonparental features that will be added to it. A heuristic recombination operator can focus in one of these aspects, or in both of them simultaneously.

As an example of a heuristic recombination operator focusing on the first aspect, Dynastically Optimal Recombination (DOR) can be mentioned. This operator explores the dynastic potential (i.e., the set of possible children) of the configurations being recombined, so as to find the best member of this set (notice that, since configurations in the dynastic potential are entirely composed of features taken from any of the parents, this is a transmitting operator). This exploration is done using a subordinate complete algorithm, and its goal is thus to find the best combination of parental features giving rise to a feasible child. Hence, this operator is monotonic in the sense that any child generated is at least as good as the best parent.

Examples of heuristic recombination operators concentrating on the selection of non-parental features, one can cite the patching-by-forma-completion operators proposed by Radcliffe and Surry. These operators are based on generating an incomplete child using a non-heuristic procedure (e.g., the RAR_{ω} operator), and then completing the child either using a local hill climbing procedure restricted to non-specified features (locally optimal forma completion) or a global search procedure that finds the globally best solution carrying the specified features (globally optimal forma completion). Notice the similarity of this latter approach with DOR.

Finally, there exist some operators trying to exploit knowledge in both of the above aspects. A distinguished example is the Edge Assembly Crossover (EAX). EAX is a specialized operator for the TSP (both for symmetric and asymmetric instances) in which

the construction of the child comprises two-phases: the first one involves the generation of an incomplete child via the so-called E-sets (subtours composed of alternating edges from each parent); subsequently, these subtours are merged into a single feasible subtours using a greedy repair algorithm. The authors of this operator reported impressive results in terms of accuracy and speed.

A final comment must be made in relation to the computational complexity of recombination. It is clear that combining the features of several solutions is in general computationally more expensive than modifying a single solution (i.e., a mutation). Furthermore, the recombination operation will be usually invoked a large number of times. For this reason, it is convenient (and in many situations mandatory) to keep it at a low computational cost. A reasonable guideline is to consider an O(N log N) upper bound for its complexity, where N is the size of the input (the set S_{par} and the problem instance x). Such limit is easily affordable for blind recombination operators, which are called crossover, a reasonable name to convey their low complexity (yet not always used in this context). However, this limit can be relatively astringent in the case of heuristic recombination, mainly when epistasis (nonadditive inter-feature influence on the fitness value) is involved. This admits several solutions depending upon the particular heuristic used. For example, DOR has exponential worst case behavior, but it can be made affordable by picking larger pieces of information from each parent (the larger the size of these pieces of information, the lower the number of them needed to complete the child). Consider that heuristic recombination operators provide better solutions than blind recombination operators, and hence they need not be invoked the same number of times.

6.2 GENERAL STRUCTURE OF MEMETIC ALGORITHMS

In order to define the notation used in this section, let us consider a solution x, i.e., a vector of n design variables $(x_1, x_2, \ldots, x_i, \ldots, x_n)$.

Each design variable xi can take values from a domain D_i (e.g., an interval $[x_i^L, x_i^U]$ if variables are continuous, or a certain collection of values otherwise). The Cartesian product of these domains for each design variable is called the decision space D. Let us consider a set of (either deterministic or stochastic) functions $f_1, f_2, \ldots, f_m$ defined in D and returning some values. Under these conditions, the most general statement of an optimization problem is given by the following formulas:

$$\begin{array}{lccl} \max / \min & f_m & & m = 1, 2, \ldots, M \\ \text{subject to} & g_j(x) \leqslant 0 & & j = 1, 2, \ldots, J \\ & h_k(x) = 0 & & k = 1, 2, \ldots, K \\ & x_i^L \leqslant x_i \leqslant x_i^U & & i = 1, 2, \ldots, n \end{array} \quad (1)$$

where g_j and h_k are inequality and equality constraints, respectively.

If m = 1 the problem is single-objective, while for m > 1 the problem is multi-objective. The particular structure of the functions g_j and h_k in each particular problem determines its constrainedness, which is often related to the hardness of its resolution. Finally, the continuous or combinatorial nature of the problem is given by the fact that D is a discrete or dense set.

MAs address the problem in (1) by means of a specific algorithmic structure which can be seen as an iterated sequence of the following operations, aimed at having a population (pool) of tentative solution converge (i.e., evolve from an initial high-diversity, scattered state to a low-diversity, more homogeneous state) towards an optimal (or quasi-optimal) solution:

- Selection of parents: Selection aims to determine the candidate solutions that will survive in the following generations and be used to create new solutions. Selection for reproduction often operates in relation with the fitness (performance) of the candidate solutions; Here, performance typically amounts to the extent to which the solution maximizes/minimizes the objective function(s) f_m (although in some cases fitness may be measured by means of a different guiding function, related to the objective function but not identical, e.g.,

in the SAT problem the objective function is binary – satisfied/unsatisfied– yet the most common fitness function is maximizing the number of satisfied clauses). High quality solutions have thus more chances to be chosen. For example, roulette-wheel and tournament selections can be applied. Selection can also be done according to other criteria such as diversity. In such a case, only spread out individuals are allowed to survive and reproduce. If the solutions of the population are sufficiently diversified, selection can also be carried out randomly.

- Combination of parents for offspring generation: Combination aims to create new promising candidate solutions by blending existing solutions (parents), a solution being promising if it can potentially lead the optimization process to new search areas where better solutions may be found.
- Local improvement of offspring: The goal of local improvement is to improve the quality of an offspring as far as possible. Candidate solutions undergo refinement which correspond the life-time learning of the individuals in the original metaphor of MAs.
- Update of the population: This step decides whether a new solution should become a member of the population and which existing solution of the population should be replaced. Often, these decisions are made according to criteria related to both quality and diversity. Such a strategy is commonly employed in methods like Scatter Search and many Evolutionary Algorithms. For instance, a basic quality-based updating rule would replace the worst solution of the population while a diversity-based rule would substitute for a similar solution according to a distance metric. Other criteria like recency (age) can also be considered. The policies employed for managing the population are essential to maintain an appropriate diversity of the population, to prevent the search process from premature convergence (i.e., too fast convergence

towards a suboptimal region of the search space), and to help the algorithm to continually discover new promising search areas.

MAs blend together ideas from different search methodologies, and most prominently ideas from local search techniques and population-based search. Indeed, from a very general point of view a basic MA can be regarded as one (or several) local search procedure(s) acting on a pool pop of $|pop| \geq 2$ solutions which engage in periodical episodes of cooperation via recombination procedures. This is shown in Algorithm 1.

```
function BasicMA (in P: Problem, in par: Parameters): Solution;
begin
    pop ← Initialize(par, P);
    repeat
        newpop1 ← Cooperate(pop, par, P);
        newpop2 ← Improve(newpop1, par, P);
        pop ← Compete (pop, newpop2);
        if Converged(pop) then
            pop ← Restart(pop, par);
        end
    until TerminationCriterion(par);
    return GetNthBest(pop, 1);
end
```

Algorithm 1: A Basic Memetic Algorithm.

Let us analyze this template. First of all, the Initialize procedure is responsible for producing the initial set of $|pop|$ solutions. Traditional evolutionary algorithms usually resort to simply generating $|pop|$ solutions at random (systematic procedures to ensure a good coverage of the search space are sometimes defined, although these are not often used). Opposed to this, it is typical for MAs to attempt to use high-quality solutions as starting point. This can be done either using a more sophisticated mechanism (for instance, some constructive heuristic) to inject good solutions in the initial population, or by using a local-search procedure to improve random solutions (see Algorithm 2).

```
function Initialize(in par: Parameters, in P: Problem):
Bag{Solution};
begin
    pop ← ∅;
    for j ← 1 to par.popsize do
        i ← RandomSolution(P);
        i ← LocalSearch (i, par, P);
        pop ← pop ∪ {i};
    end
    return pop;
end
```

Algorithm 2: Injecting high-quality solutions in the initial population.

As for the Termination Criterion function, it typically amounts to checking a limit on the total number of iterations, reaching a maximum number of iterations without improvement, having performed a certain number of population restarts, or reaching a certain target fitness.

The procedures Cooperate and Improve constitute the core of the MA. Starting with the former, its most typical realization arises from the use of two operators for selecting solutions from the population and recombining them.

Table 1: Parameters used in the algorithmic description of MAs

parameter	interpretation
popsize	size of the population (number of solutions in *pop*)
numop	number of operators used
numapps	array of size 1..numop indicating the number of times each operator is applied in the main loop.
arityin	array of size 1..numop indicating how many input solutions are required by each operator.
arityout	array of size 1..numop indicating how many output solutions are produced by each operator.
op	array of size 1..numop comprising the actual operators
preserved	number of solutions in the current population that are preserved when a restart is made.

```
function Cooperate (in pop: Bag{Solution}, in par: Parameters, in P:
Problem): Bag{Solution};
begin
    lastpop ← pop;
    for j ← 1 to par.numop do
        newpop ← ∅;
        for k ← 1 to par.numapps^j do
            parents ← Select (lastpop, par.arityin^j);
            newpop ← newpop ∪ ApplyOperator (par.op^j, parents, P);
        end
        lastpop ← newpop;
    end
    return newpop;
end
```

Algorithm 3: The pipelined Cooperate procedure.

This procedure can be easily extended to use a larger collection of variation operators applied in a pipeline fashion. As shown in Algorithm 3, this procedure comprises numop stages, each one corresponding to the iterated application of a particular operator op^j that takes $arityin^j$ solutions from the previous stage, generating arityoutj new solutions.

As to the Improve procedure, it embodies the application of a local search procedure to solutions in the population. Notice that in an abstract sense a local search method can be modelled as a unary operator (we adhere here to a strict definition of local search as a procedure for iteratively exploring the surroundings/ neighborhood of a certain solution at any given time step), and hence it could have been included within the Cooperate procedure above. However, local search plays such an important role in MAs that it deserves separate treatment. Indeed, there are several important design decisions involved in the application of local search to solutions, i.e., to which solutions should it be applied, how often, for how long, etc.

Next, the Compete procedure is used to reconstruct the current population using the old population pop and the population of offspring $newpop_2$. Using the terminology commonly used by the evolution strategy community, there exist two main possibilities for this purpose: the plus strategy and the comma strategy. The non-elitist nature of the latter makes it less prone to stagnation, being

the ratio $|\text{newpop}|/|\text{pop}| \simeq 6$ a customary choice. The generation of a large number of offspring can be somewhat computationally expensive if the fitness function is complex and time-consuming though. A suitable alternative in this context is using a plus strategy with a low value of $|\text{newpop}|$, an elitist variant which is strongly related to the so-called steady-state replacement strategy in GAs. While this option usually provides a faster convergence to high-quality solutions, premature convergence to suboptimal regions of the search space can take place, and hence corrective measures may be required. This leads to the last component of the template shown in Algorithm 1, namely the restarting procedure.

First of all, it must be decided whether the population has degraded or has not, using some measure of information diversity in the population (e.g., average Hamming distance or Shannon's entropy in the discrete case, or some dispersion measure in the continuous case). Once the diversity indicator provides a value below a suitable threshold, the population can be regarded as degenerate and the restart procedure is called. Again, this can be implemented in a number of ways. A very typical strategy is to keep a fraction of the current population, generating new (random or heuristic) solutions to complete the population, as shown in Algorithm 4. The term random-immigrant strategy has been coined to describe this procedure. Alternatively, a strong or heavy mutation operator can be activated in order to drive the population away from its current location in the search space.

```
function Restart (in pop: Bag{Solution}, in par: Parameters, in P:
Problem): Bag{Solution};
begin
    newpop ← ∅;
    for j ← 1 to par.preserved do
        i ← GetNthBest(pop, j);
        newpop ← {i};
    end
    for j ← par.preserved + 1 to par.popsize do
        i ← RandomSolution(P);
        i ← LocalSearch (i, par, P);
        newpop ← {i};
    end
    return newpop;
end
```

Algorithm 4: The Restart procedure.

On the basis of the definitions of MA and MC reported above, while an algorithmic characterization of MA can be given, any MC specific outline would be restrictive. In other words, while MA is a class of optimization algorithms having specific implementation features, MC is a subject and an implementation philosophy. On one hand, the concept of MC appears excessively vague as all the computer science implementations if not most of the natural sciences and engineering can be seen as a subset of MC. If we look at MC in a sceptical way, it may appear as an empty box or a label to put on every single human thought. On the other hand, the importance of MC is in the unifying role taken and the novel perspective that MC suggests to computer science community. MC considers algorithms as evolving structures composed by cooperative and competitive operators. This perspective suggests the automatic generation of algorithms by properly combining the operators (memes). We may think that a computational device stores a set of operators and combines (some of) them according to a certain criterion to efficiently address a problem. This will be a firther step with respect to adaptive and self-adaptive systems in MAs.

6.3 MEMETIC COMPUTING SPECIFIC IMPLEMENTATIONS

MA/MC implementations for various classes of optimization problems. More specifically the present divided into the following methods:

- MAs in discrete optimization
- MAs in continuous optimization
- MAs in multimodal optimization
- MAs in constrained optimization
- MAs in multi-objective optimization
- MAs in the presence of uncertainties

6.3.1 MAs in Discrete Optimization

Discrete optimization is the search for the configuration with highest performance (optimal solution) among a set of finite candidate configurations. There are several ways to describe a discrete optimization problem. In its most general form, it can be defined as a collection of problem instances, each being specified by a pair (S, f), where S is the a finite set of candidate configurations, defining the decision space; f is the cost or objective function, given by a mapping f: S → Q.

Unlike continuous problems, discrete optimization can in principle be solved by enumeration, i.e., by exhaustively counting and evaluating all the candidate solutions. In addition, discrete problems cannot utilize the gradient for searching the directions as a minimum distance between two solutions is set.

Discrete problems and more specifically the Travelling Salesman problem (TSP) have been the earliest application domains for MAs. Implementations of hybrid algorithms were in use even before the term MA was coined. In an early attempt to hybridize an evolutionary framework with local search for solving the TSP has been presented. Subsequently, still with reference to the TSP, in a visionary approach which theorizes the integration of extra components and especially crossover techniques within an evolutionary framework is presented. A similar approach is given in. Another related technique, which can also be considered as an early memetic approach is the so called genetic edge recombination. More recently, actual MAs (which fit in the definition above) have been implemented to address the TSP; the role and effect of local search within evolutionary algorithms is extensively studied.

The solution of an optimization problem in a discrete space (as well as for continuous problems) must be achieved by efficiently balancing the exploitation and exploration. Exploitation is the action, performed by the algorithm, of intensively analyzing a portion of the decision space in order to quickly enhance upon the best current solution while exploration is the action which leads to the detection of a candidate solution located in an unexplored

areas of the decision space. The dual concept of exploitation and exploration covers two fundamental and complementary aspects of any effective search procedure. This concept is at the basic of optimization and has been termed under the names intensification and diversification, respectively, introduced within the Tabu Search (TS) methodology.

MA implementations for discrete optimization problems essentially tend to combine searchers for exploring the entire decision space and searchers which focus on portions of the decision space. Local search in MAs for discrete optimization performs an intensive exploitation of the search space attempting to enhance the performance by slightly modifying some design variables. For example, an analysis of the frequency and application point of the local search, in the context of continuous optimization, is carried out. This analysis has been extended for combinatorial optimization problems and introduced the concept of sniff (or local/global ratio) for balancing genetic and local search.

6.3.2 MAs in Continuous Optimization

When a MA is designed two of the most relevant features to take into account are 1) the cost of local search; 2) the underlying search landscape. In order to come up with efficient memetic solvers, in continuous optimization, these features must be tackled differently with respect to the discrete case.

Regarding the cost of local search, in many combinatorial domains it is frequently possible to compute the fitness of a perturbed solution incrementally, e.g., let x be a solution and let $x' \in N(x)$ be a neighboring solution; then the fitness $f(x')$ can be often computed as $f(x') = f(x) + \Delta f(x, x')$, where $\Delta f(x, x')$ is a term that depends on the particular perturbation done on x and is typically efficient to compute (much more efficiently that a full fitness computation). For example, in the context of the TSP and the 2-opt neighborhood, the fitness of a perturbed solution can be computed in constant time by calculating the difference between the weights of the two edges added and the two edges removed. This is much

more difficult in the context of continuous optimization problems, which are often non-linear and hard to decompose as the sum of linearly-coupled terms. Hence local search usually has to resort to full fitness computations.

Concerning the underlying search landscape, it should be observed that the interplay among the different search operators used in memetic algorithms (or even in simple evolutionary algorithms) is a crucial issue for achieving good performance in any optimization domain. When tackling a combinatorial problem, this interplay is a complex topic since each operator may be based on a different search landscape. It is then essential to understand these different landscape structures and how they are navigated; this concept is also known ad the "one operator, one landscape" view and is expressed in depth. In the continuous domain the situation is somewhat simpler, in the sense that there exists a natural underlying landscape in D (typically $D = Q^n$), namely that induced by distance measures such as Euclidean distance. In other words, in continuous optimization, the set of points which can be reached by the application of unary operators to a starting point may be represented by closed spheres of radius ϵ. On the contrary, the set of points reachable by recombination operators (recall for example the BLX–α operator) can be visualized by means of a hypercubes within the decision space. The intuitive imagery of local optima and basins of attraction naturally fits here, and allows the designer to exert some control on the search dynamics by carefully adjusting the intensification/diversification properties of the operators used.

Starting with the first one (the cost of local search), it emphasizes the need for carefully selecting when and how local search is applied (obviously this is a general issue, also relevant in combinatorial problems, but definitely crucial in continuous ones). This decision-making is very hard in general, but some strategies have been put forward in previous works. A rather simple one is to resort to partial Lamarckianism by randomly applying local search with probability $pLS < 1$. Obviously, the application frequency is not the only parameter that can be adjusted to tune the computational cost of local search: the intensity of local search

(i.e., for how long is local improvement attempted on a particular solution) is another parameter to be tweaked. This adjustment can be done blindly (i.e., prefixing a constant value or a variation schedule across the run), or adaptively. For example, Molina et al. define three different solution classes (on the basis of fitness) and associate a different set of local-search parameters for each of them. Related to this, Nguyen et al. consider a stratified approach, in which the population is sorted and divided into n levels (n being the number of local search applications), and one individual per level is randomly selected. This is shown to provide better results than random selection. We refer to for an in-depth empirical analysis of the time/quality tradeoffs when applying parameterized local search within memetic algorithms. This adaptive parameterization has been also exploited in so-called local-search chains, by saving the state of the local-search upon completion on a certain solution for later use if the same solution is selected again for local improvement. Let us finally note with respect to this parameterization issue that adaptive strategies can be taken one step further, entering into the realm of self-adaptation.

As to what the exploitation/exploration balance regards, it is typically the case that the population-based component is used to navigate through the search space, providing interesting starting points to intensify the search via the local improvement operator. The diversification aspect of the populationbased search can be strengthened in several ways, such as for example using multiple subpopulations, or diversity-oriented replacement strategies. An optimization paradigm closely related to memetic algorithms in which the population (or reference set in the SS jargon) is divided in tiers: entrance to them is gained by solution on the basis of fitness in one case, or diversity in the other case.

Diversification can be also introduced via selective mating, as it is done in CHC (Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation). A related strategy was proposed by Lozano et al. via the use of negative assortative mating: after picking a solution for recombination, a collection of

potential mates is selected and the most diverse one is used. Other strategies range from the use of clustering (to detect solutions likely within the same basin of attraction upon which it may not be fruitful to apply local search), or the use of standard diversity preservation techniques in multimodal contexts such as sharing or crowding. It should be also mentioned that sometimes the intensification component of the memetic algorithm is strongly imbricated in the population-based engine, without resorting to a separate local search component. This is for example the case of the so-called crossover hill climbing, a procedure which essentially amount to using a hill climbing procedure on states composed of a collection of solutions, using crossover as move operator. A different intensifying strategy was used by, by considering an exact procedure for finding the best combination of variable values from the parents (a so-called optimal discrete recombination). This obviously requires the objective function is amenable to the application of an efficient procedure for exploring the dynastic potential (set of possible children) of the solutions being recombined.

6.3.3 MAs in Multimodal Optimization

In some cases, it may be required to detect multiple local optima rather than only the global optimum. This problem is usually indicated as multimodal optimization problem. Obviously, this situation occurs only when there is a continuous landscape because in discrete optimization there is no absolute concept of local optimum. MC approaches have been used in various contexts to address this issue. For example, a memetic approach composed of sequential threshold operation, global and local search allows the detection of multiple optima under fitness constrains. In a heuristic mapping is proposed in order to promote the multiple convergence within a unique evolutionary cycle. By means of a similar logic, in a memetic swarm intelligence approach is used for multimodal optimization.

6.3.4 MAs in Large Scale Optimization

Optimization problems, both discrete and continuous, when characterized by a high number of variables are known as large scale optimization problems, or briefly Large Scale Problems (LSPs). The detection of an efficient solver for LSPs can be a very valuable achievement in applied science and engineering since in many applications a high number of design variables may be of interest for an accurate problem description. For example, in structural optimization an accurate description of complex spatial objects might require the formulation of a LSP; similarly such a situation also occurs in scheduling problems.

Several memetic approaches have been largely applied in order to solve LSPs. This fact is due to the fact that a single search logic might easily turn into stagnation or premature convergence. On the other hand, a proper coordination of multiple search operators can compensate the limits of the others and thus allow the overcome of a critical algorithmic situation characterized by no improvements. For example, a MA which integrates a simplex crossover within the DE framework has been proposed in order to solve LSPs. A DE for LSPs has proposed. The algorithm proposed in performs a probabilistic update of the control parameter of DE variation operators and a progressive size reduction of the population size. Although the theoretical justifications of the success of this algorithm are not fully clear, the proposed approach seems to be extremely promising for various problems. A memetic algorithm which hybridizes the self-adaptive DE described and a local search applied to the scale factor in order to generate candidate solutions with a high performance has been proposed. Since the local search on the scale factor (or scale factor local search) is independent on the dimensionality of the problem, the resulting memetic algorithm offered a good performance for relatively large scale problems. By combining the latest two philosophies, Caponio et al. propose a MA which integrates the potential of the scale factor local search within the self-adaptive DE with automatic reduction of the population size in order to guarantee a high performance, in

terms of convergence speed and solution detection, for large scale problems.

A DE framework with self-adaptively coordinated multiple mutation strategies, is hybridized in a memetic fashion with the multitrajectory search proposed. The resulting algorithm appears very promising for handling LSPs.

Finally, another memetic approach, used for handling LSPs, is by means of structured populations. One example is given in where multiple DE search strategies are reproduced within a ring topology by means of a simple and natural randomized adaptation throughout the islands of the structured populations. The scale factor of the most successful islands is inherited by the other islands after a perturbation which prevents from premature convergence. A more efficient scheme for handling LSPs is proposed in where the premature convergence is achieved by means of the cooperative/competitive application of two simple mechanisms: the first, namely shuffling, consists of randomly rearranging the individuals over the sub-populations; the second consists of updating all the scale factors of the sub-populations.

6.3.5 MAs in Constrained Optimization

When MAs are applied to constrained optimization problems, the integration of algorithmic components in the memetic framework to handle the constraints becomes fundamental. In a MA composed of a GA framework and a gradient based local search integrates the constraint violation criterion proposed in: (i) the feasible individual is preferred over the infeasible one; (ii) for two feasible individuals, the individual with better fitness is preferred; and (iii) for two infeasible individuals, the individual with lower constraint violation is preferred. Their experimental results indicated that MA outperformed conventional algorithms in terms of both quality of solution and the rate of convergence. The same set of rules has been used to handle the constraints, where, in the context of multi-objective optimization, a MA which

makes use of a local search strategy based on the interior point method, has been proposed.

A MA composed by an evolutionary framework and Sequential Quadratic Programming (SQP) employs the constraint violation procedure. An MA containing an adaptive penalty method and a line search technique is proposed. An agent based MA in which four local search algorithms were used for adaptive learning has been proposed. The algorithms included random perturbation, neighborhood and gradient search methods. Subsequently, another specialized local search method was designed to deal with equality constraints.

A memetic co-evolutionary differential evolution algorithm where the population was divided into two sub-populations has been proposed. The purpose of one sub-population is to minimize the fitness function, and the other is to minimize the constraint violation. The optimization was achieved through interactions between the two sub-populations. No penalty coefficient has been used in the method while a Gaussian random number was used to modify the individuals when the best solution remained unchanged over several generations.

Some domain-specific applications are solved by means of MAs for constraint optimization. Boudia and Prins considered the problem of cost minimization of a production-distribution system. A repair mechanism was applied for constraint satisfaction. Park et al. combined a GA framework with a tunnel-based dynamic programming scheme to solve highly constrained non-linear discrete dynamic optimization problems arising from long-term planning. The infeasible solutions were repaired by randomly sampling part of the solutions and replacing some of the previous variables (regenerate partial characters). The algorithm successfully solved reasonable sized practical problems which cannot be solved by means of conventional approaches. A multistage capacitated lot-sizing problem was solved by the memetic algorithm proposed in using heuristics as local search and standard recombination operators. Gallardo et al. propose a multilevel MA for solving

weighted constrained satisfaction problems, based on the integration of exact techniques within the MA for recombination purposes, and the use of upper coordination level involving the MA and an incomplete branch and bound derivate.

Some other studies, instead of dealing with conventional candidate solutions, require the encoding of mixed continuous/integer variables or the inclusion of boolean variables. Within this class of problems, mixed representations of the constrained Vehicle Routing Problems (VRPs) have been extensively studied in literature and several MA implementations have been proposed.

6.3.6 MAs in Multi-Objective Optimization

In order to tackle multi-objective optimization problems, a well designed algorithm should capable to detect a set of points representative of the Pareto front being well sparse over it. Multi-Objective MAs (MOMAs) attempt to obtain this result properly hybridizing evolutionary operators and local search. In order to pursue this aim, the selection mechanism, i.e., that mechanism that chooses which solutions should be retained and which discarded, must be well designed. A first important feature of the selection mechanism is that within a set of solutions, those that dominate the others should be chosen. However, dominance relation alone leaves many pairs of solutions incomparable. For this reason, the employment of only the dominance relation may not be able to define a single best solution in a neighborhood or in a tournament.

There are mainly two big families of multi-objective solvers (regardless of their memetic nature) and can be classified in the following way: 1) algorithms that do not combine the objective functions and perform the selection by means of a dominance based criterion; 2) algorithms that make use of combinations of objectives for selecting new individuals.

The first category is based on the dominance sorting defined in and consists of a dominance-based ranking of all the solutions of a population. This mechanism has been employed by popular

evolutionary algorithms for multi-objective optimization.

In MOMAs the selection criterion involves not only the evolutionary framework but also the local search components. In a greedy local search method based on dominance relation is proposed. This mechanism simply allows the acceptance of a newly generated neighbor solution if it dominates the current solution. In population-based Pareto local search, the neighborhood of each solution of the current population is explored, and if no solution of the population weakly dominates a generated neighbor, the neighbor is added to the population. Lust and Jaszkiewicz propose a method to speed-up local search algorithms based on dominance sorting. A dominance criterion is integrated into the evolutionary framework and multiple local search components such as Simulated Annealing and Rosenbrock These approaches have the advantage of not requiring extra parameters for performing their implementation. On the other hand, this criterion does not allow a control on the solution spread in proximity of the Pareto front. This drawback imposes the employment of extra components which guarantee the population spread (in terms of fitness values), see e.g.,. In addition, while dominance allows a good ranking when few objectives are involved, it is often unreliable when the problem handles many simultaneous objectives. It is likely to have sets solutions which do not dominate each other and thus the algorithm cannot perform an efficient selection.

The second category is based on the idea that if a ranking amongst the objectives can be performed then the multiple objectives can be combined to generate a single-objective optimization problem. The ranking is performed by associating to each objective a weight value. The functions combining the objectives are usually indicated as aggregation functions. When this approach is employed the algorithm obviously does not detect a Pareto front but only one solution. However, this drawback can be overcome by the use of multiple aggregation functions defined by various weight vectors. A scheduled variation of weight parameters is employed in. A deterministic updated of the weight parameters to generate a repulsion among solution and thus dispersion in proximity of

the Pareto front is proposed. A meta-evolution of the weights is presented in. A randomized weight update, similar to a random walk local search, is proposed in while a fully random update is presented. The employment of multiple set of weight parameters allows a natural dispersion of the solutions and thus, unlike dominance based sorting methods, no additional components are required. In addition, several speed-up techniques may easily be used in local search based on aggregation functions. On the other hand, this category of methods has the drawback that the selection of a proper set of weights must be performed. In order to overcome this problem, some research is focused on the automatic selection of the weights.

6.3.7 MAs in the Presence of Uncertainties

Uncertainties in optimization problems are very common in real-world applications due to the presence of measurement devices and approximation models. A fitness function contains uncertainties if the variable "time" takes place in the fitness evaluation of a solution. In other words, if for a given candidate solution x, the fitness calculation f (x) can return different values in different moments, then the fitness function f is said to be affected by uncertainties. In the survey proposed the sources of uncertainties are categorized as 1) uncertainties due to approximation 2) uncertainties due to robustness 3) uncertainties due to noise 4) uncertainties due to time-variance.

In some applications, the actual fitness function can be unavailable throughout the entire optimization process or, due to its excessive computational cost, can be replaced by an approximation model. When the fitness value is computed by an approximation model a slightly different value than the actual fitness is expected. In addition, an approximation procedure can be adjusted over the optimization time and alternated with the actual fitness thus resulting in multiple fitness values for a single candidate solution. The employment of approximation models introduces an uncertainty in the landscape. In order to face this difficulty, in the Inexact Pre-Evaluation (IPE) framework is proposed. IPE

uses the expensive function in the first few generations and then uses the model almost exclusively while only a portion of the elites are evaluated with the expensive function and are used to update the model. This mechanism has been integrated into a hierarchical distributed algorithm. This idea has been expanded such that each layer may use different solvers, within a memetic framework employing a gradient based search. The Controlled Evaluations (CE) framework has been proposed. This framework monitors the model accuracy using cross-validation: a memory structure containing the previously evaluated vectors is split into two sets which are then used to train the approximation model. In the context of expensive multi-objective optimization, a memetic approach integrated fuzzy logic for alternating real and approximated fitness evaluation has been proposed. Another widely used option is a memetic approach employing the Trust Region (TR), i.e., a portion of the decision space where the approximation model can be reliably used. Memetic frameworks combining an EA as a global search, where at each generation every non-duplicated vector in the population is refined using a TR, has been proposed. The authors proposed a TR memetic framework which uses quadratic models and clustering. Zhou et al. proposed a memetic framework which occasionally uses an inaccurate model capable to detect proposing solutions. Lim at el. have recently proposed a framework composed of an ensemble of approximation models as well smoothing models. Other approaches, namely model-adaptive frameworks, have been proposed. Model-adaptive frameworks employ a set of candidate models which are automatically selected by a supervising system.

Robust parameters of a system are those parameters which lie in a region of the parameter hyperspace characterized by similar system responses. In other words, if a robust parameter is slightly perturbed, the system response only slightly varies. Robust optimization is a field of optimization theory which aims to detecting robust parameters. Reversely, if a parameter is not robust, small parameter variations can result into large variation of the system response. Very close solution, ideally identically can give very different system response and thus in robust optimization,

identical solutions can be characterized by very different fitness values. In order to address these problems, in an algorithm for robust optimization of digital filters where the uncertainty in performance is due to material imperfections has been proposed. The problem of optimizing a robust aircraft control system using a memetic algorithms is studied. Still in the context of aircraft design, a surrogated based approach, i.e., an approximation model, for computationally expensive optimization problems is proposed in. The robust control design of a control system for an electric motor is proposed in by applying a surrogate assisted model. Other examples of memetic robust design, regarding multi-objective optimization, are given in. addressed the problem of robust optimization when no a-priori information about the distribution of uncertainties is known. The problem of robust design in constrained multi-objective optimization is analyzed by means of a MA. In the latter work, micro-populations act as local search within the decision space. In a robust airline scheduling problem where the goal was to obtain a fleet assignment which accounts for flight re-timing and aircraft rerouting has been proposed.

The noise in optimization is a typical condition which plagues real-world applications and occurs every time measurements concur to the fitness value computation. These measurements can be physical instruments, or computational devices which contain uncertainties, such as a Neural Network, see e.g.,. Some examples of memetic frameworks addressing noisy landscapes are given in the following. Kim and Abraham combine a bacteria foraging algorithm with a real-coded evolutionary algorithm for addressing a control engineering design problem. The noise is handled by re-sampling and filtering. In MA based on differential evolution where the scale factor was adjusted with a line search is proposed and combined with an adaptive resampling technique. The authors considered the noisy pattern recognition problem of inexact graph matching, that is, determining whether two images match when one is corrupted by noise. Ozcan and Mohan studied the problem of matching an input image to one from an available data set. The difficulty being that the input image may be partially

obscured, deformed and so on which results in a noisy optimization problem. A resampling technique is integrated within a MA which uses a self-organizing map (SOM) as a local search. The algorithm was designed to solve the VRP with emphasis on noisy data. The authors tackled the problem of training a neural network used for controlling resource discovery in peer-to-peer (P2P) networks. In order to face this kind of problem, a diversity based adaptation is proposed.

Time-variance occurs when the fitness values of (at least some of) the points depend on time. This situation can be visualized as a landscape which is not stationary but moves over time, twisting and changing shape. This fact obviously implies that the position of the optima varies with time and thus, when the optima are detected, the algorithm should be able to follow the basins of attraction to find and locate them anew. It should be remarked that while the three previous categories the uncertainties are due to an erroneous estimation of the fitness value in a point, in time-variant problems the actually fitness value of a solution varies over time. In order to tackle this class of problems, a MA combining a binary evolutionary framework with the variable local search (VLS) operator to track optima in dynamic (time-variance) problems has been proposed. A MA based on Particle Swarm Optimization (PSO) for dynamic optimization problems has been proposed. This modified PSO employs multiple techniques for handling the time-dependence. Moser and Hendtlass combined the Extremal Optimization algorithm (EO) and a deterministic local search. Due to its structure, EO naturally adapts to changing environments and thus is a promising background for this class of problems. Another variant, employing the Hooke-Jeeves Algorithm. A MC approach based on the scatter search framework for dynamic and highly constrained problems. In the context of dynamic multiobjective problems, a multi start system is achieved by accelerating the convergence of the algorithm. This aim is pursued by means of a modified gradient capable to predict the changes in the Pareto set. Wang et al. proposed a MA for dynamic optimization which used a binary representation where at each generation the elite

was refined by a local search algorithm and added and updated while the fitness landscape changes.

6.4 ALGORITHMIC EXTENSIONS OF MEMETIC ALGORITHMS

Multiobjective problems are frequent in real-world applications. Rather than having a single objective to be optimized, the solver is faced with multiple, partially conflicting objectives. There is no a priori single optimal solution, but rather a collection of optimal solutions, providing different trade-offs among the objectives considered. The notion of Pareto-dominance is essential: given two solutions s, $s' \in sol_P(x)$, s is said to dominate s' if it is better than s' in at least one of the objectives, and it is no worse in the remaining ones. This clearly induces a partial order $\prec P$, since given two solutions it may be the case that none of them dominates the other. This collection of optimal solutions is termed the optimal Pareto front, or the optimal non-dominated front.

Population-based search techniques, in particular evolutionary algorithms (EAs), are naturally fit to deal with multiobjective problems, due to the availability of a population of solutions which can approach the optimal Pareto front from different directions. MAs can obviously benefit from this corpus of knowledge. However, MAs typically incorporate a local search mechanism, and it has to be adapted to the multiobjective setting as well. This can be done in different ways, which can be roughly classified into two major classes: scalarizing approaches, and Pareto-based approaches. The scalarizing approaches are based on the use of some aggregation mechanism to combine the multiple objectives into a single scalar value. This is usually done using a linear combination of the objective values, with weights that are either fixed (at random or otherwise) for the whole execution of the local search procedure, or adapted as the local search progresses. As to Pareto-based approaches, they consider the notion of Pareto-dominance for deciding transitions among neighboring solutions,

typically coupled with the use of some measure of crowding to spread the search.

A full-fledged multiobjective MA (MOMA) is obtained by appropriately combining population-based and local search-based components for multiobjective optimization. Again, the strategy used in the local search mechanism can be used to classify most MOMAs. Thus, two proposals due to Ishibuchi and Murata and to Jaszkiewicz are based on the use of random scalarization each time a local search is to be used. Alternatively, a single-objective local search could be used to optimize individual objectives. Ad hoc mating strategies based on the particular weights chosen at each local search invocation (whereby the solutions to be recombined are picked according to these weights) are used as well. A related approach – including the on-line adjustment of scalarizing weights– is followed by Guo et al. On the other hand, a MA based on PAES (Pareto Archived Evolution Strategy) was defined by Knowles and Corne. More recently, a MOMA based on particle swarm optimization (PSO) has been defined by Liu et al. In this algorithm, an archive of nondominated solutions is maintained and randomly sampled to obtain reference points for particles. A different approach is used by Schuetze et al. for numerical-optimization problems. The continuous nature of solution variables allows using their values for computing search directions. This fact is exploited in their local search procedure (HCS for Hill Climber with Sidestep) for directing the search toward specific regions (e.g., along the Pareto front) when required.

6.4.1 Adaptive Memetic Algorithms

The fact that these were heuristics that ultimately relied on the problem-knowledge available was stressed. This is not a particular feature of MAs, but affects the field of metaheuristics as a whole. Indeed, one of the keystones in practical metaheuristic problem-solving is the necessity of customizing the solver for the problem at hand. Therefore, it is not surprising that attempts to transfer a part of this tuning effort to the metaheuristic technique itself have been common. Such attempts can take place at different levels, or

can affect different components of the algorithm. The first –and more intuitive one– is the parametric level involving the numerical values of parameters, such as the operator application rates.

A slightly more general approach –termed 'meta-lamarckian learning' by Ong and Keane– takes place at the algorithmic level. They consider a setting in which the MA has a collection of local search operators available, and how the selection of the particular operator(s) to be applied to a specific solution can be done on the basis of past performance of the operator, or on the basis of the similarity of the solution to previous successful cases of operator application. Some analogies can also be drawn here with hyperheuristics, a high-level heuristic that controls the application of a set of low-level heuristics to solutions, using strategies ranging from pure random to performance-based rules.

In general terms, the approaches mentioned before are based on static, hard-wired mechanisms that the MA uses to react to the environment. Hence, they can be regarded as adaptive, but not as self-adaptive. The actual definition of the search mechanisms can evolve during the search. This is a goal that has been pursued for long in MAs. Back in the early days of the field, it was already envisioned that future generations of MAs would work in at least two levels and two time scales. During the short-time scale, a set of agents would be searching in the search space associated to the problem. The long-time scale would adapt the algorithms associated with the agents. Here we encompass individual search strategies, recombination operators, etc. A simple example of this kind of self-adaptation can be found in the so-called multi-memetic algorithms, in which each solution carries a gene that indicates which local search has to be applied on it. This can be a simple pointer to an existing local search operator, or even the parametrization of a general local search template, with items such as the neighborhood to use, acceptance criterion, etc. Going beyond, a grammar can be defined to specify a more complex local search operator. At an even higher level, this evolution of local search operators can be made fully symbiotic, rather than merely endosymbiotic. For this purpose, two co-evolving populations can

be considered: a population of solutions, and a population of local search operators. These two populations co-operate by means of an appropriate pairing mechanism, that associates solutions with operators. The latter receive fitness in response on their ability to improve solutions, thus providing a fully self-adaptive strategy for exploring the search landscape.

6.4.2 Complete Memetic Algorithms

The combination of exact techniques with metaheuristics is an increasingly popular approach. Focusing on local search techniques, Dumitrescu and St¨uztle have provided a classification of methods in which exact algorithms are used to strengthen local search, i.e., to explore large neighborhoods, to solve exactly some subproblems, to provide bounds and problem relaxations to guide the search, etc. Some of these combinations can be also found in the literature on population-based methods. For example, exact techniques –such as branch-and-bound (BnB) or dynamic programming among others– have been used to perform recombination, and approaches in which exact techniques solved some subproblems provided by EAs date back to 1995.

Puchinger and Raidl have provided a classification of this kind of hybrid techniques in which algorithmic combinations are either collaborative (sequential or intertwined execution of the combined algorithms) or integrative (one technique works inside the other one, as a subordinate). Some of the exact/metaheuristic hybrid approaches defined before are clearly integrative –i.e., using an exact technique to explore neighborhoods. Further examples are the use of BnB in the decoding process of a genetic algorithm (i.e., exact method within a metaheuristic technique), or the use of evolutionary techniques for the strategic guidance of BnB (metaheuristic approach within an exact method).

As to collaborative combinations, a sequential approach in which the execution of a MA is followed by a branch-and-cut method can be found in. Intertwined approaches are also popular. For example, Denzinger and Offerman combine genetic algorithms and BnB

within a parallel multi-agent system. These two algorithms also cooperate in, the exact technique providing partial promising solutions, and the metaheuristic returning improved bound.

6.5 DESIGN ISSUES

MAs are commonly implemented as EAs endowed with a local search component, and therefore the theoretical corpus available for the former can be used to guide some aspects of the design process, e.g., the representation of solutions in terms of meaningful information units.

The most MA-specific design decisions are those related to the local search component, not just from the point of view of parameterization (see below) but also with the actual inner working of the component and its interplay with the remaining operators. This latter issue is well exemplified in the work of Merz and Freisleben on the TSP. They consider the use of the Lin-Kernighan heuristic, a highly intensive local search procedure, and note that the average distance between local optima is similar to the average distance between a local optimum and the global optimum. For this reason, they introduce a - operator that generate offspring whose distance from the parents is the same as the distance between the parents themselves. Such an operator is likely to be less effective if a less powerful local improvement method, e.g., 2-opt, was used, inducing a different distribution of local optima.

Once a local search procedure is selected, an adequate parameterization must be determined, i.e., how often it must be applied, how to select the solutions that will undergo local improvement, and how long must improvement epochs last. These are delicate issues since there exists theoretical evidence that an inadequate parameter setting can turn the algorithmic

solution from easily solvable to nonpolynomially solvable. Regarding the probability of application of local search, its precise values largely depends on the problem under consideration, and its determination is in many cases an art. For this reason, adaptive and self-adaptive mechanisms have been defined in order to let the algorithm learn what the most appropriate setting is. The term partial lamarckianism is used to denote these strategies where not every individual is subject to local search.

As to the selection of individuals that will undergo local search, most common options are random-selection, and fitness-based selection, where only the best individuals are subject to local improvement. For example, Nguyen et al. [56] consider an approach in which the population is sorted and divided into n levels (n being the number of local search applications), and one individual per level is randomly selected. Note that such a strategy can be readily deployed on a structured MA as defined by Moscato et al., in which fitness-based layers are explicitly available.

6.6 APPLICATIONS OF MEMETIC ALGORITHMS

This overview is far from exhaustive since new applications are being developed continuously. However, it is intended to illustrate the practical impact of these optimization techniques. We have organized references in five major areas: machine learning and knowledge discovery (Table 2), traditional combinatorial optimization (Table 3), planning, scheduling and timetabling (Table 4), bioinformatics (Table 5), and electronics, engineering, and telecommunications (Table 6). We have tried to be illustrative rather than exhaustive, pointing out some selected references from these well-known application areas.

Table 2. Applications in machine learning and knowledge discovery

Data Mining and Knowledge Discovery	Image analysis
	Fuzzy clustering
	Feature selection
	Pattern recognition
Machine Learning	Decision trees
	Inductive learning
	Neural networks

Table 3. Applications in combinatorial optimization

Binary & Set Problems	Binary quadratic programming
	Knapsack problem
	Low autocorrelation sequences
	Max-SAT
	Set covering
Graph-based Problems	Crossdock optimization
	Graph coloring
	Graph matching
	Hamiltonian cycle
	Maximum cut
	Quadratic assignment
	Routing problems
	Spanning tree
	Steiner tree
	TSP
Constrained Optimization	Golomb ruler
	Social golfer
	Maximum density still life

Table 4. Applications in planning, scheduling, timetabling, and manufacturing

Manufacturing	Assembly line
	Flexible manufacturing
	Lot sizing
	Multi-tool milling
	Supply chain network
Planning	Temporal planning
Scheduling	Flowshop scheduling
	Job-shop
	Parallel machine scheduling
	Project scheduling
	Single machine scheduling

Timetabling	Driver scheduling
	Examination timetabling
	Rostering
	Sport league
	Train timetabling
	University course

Table 5. Applications in bioinformatics

Phylogeny	Phylogenetic inference
	Consensus tree
Microarrays	Biclustering
	Feature Selection
	Gene ordering
Sequence Analysis	Shortest common supersequence
	DNA sequencing
Protein Science	Sequence assignment
	Structure comparison
	Structure prediction
Systems Biology	Gene regulatory networks
	Cell models
Biomedicine	Drug therapy design

Table 6. Applications in electronics, telecommunications and engineering

Electronics	Analog circuit design
	Circuit partitioning
	Electromagnetism
	Filter design
	VLSI design
Engineering	Chemical kinetics
	Crystallography
	Drive design
	Power systems
	Structural optimization
	System modelling
Computer Science	Code optimization
	Information forensics
	Information theory
	Software engineering
Telecommunications	Antenna design
	Mobile networks
	P2P networks
	Wavelength Assignment
	Wireless networks

Although these fields encompass the vast majority of applications of MAs, it must be noted that success stories are not restricted to these major fields. To cite an example, there are several applications of MAs in economics, e.g., in portfolio optimization, risk analysis, and labor-market delineation.

REFERENCES

1. Abbass, H. A., 2002. An evolutionary artificial neural networks approach for breast cancer diagnosis. Artificial Intelligence in Medicine 25 (3), 265–281.
2. Barkat Ullah, A. S. S. M., Sarker, R., Cornforth, D., Lokan, C., 2009. AMA: A new approach for solving constrained real-valued optimization problems. Soft Computing 13 (8–9), 741–762.
3. Barkat Ullah, A. S. S. M., Sarker, R., Lokan, C., 2009. An agent-based memetic algorithm (AMA) for nonlinear optimization with equality constraints. In: CEC 2009. IEEE Press, Trondheim, Norway, pp. 70– 77.
4. Basseur, M., 2006. Design of cooperative algorithms for multi-objective optimization: application to the flow-shop scheduling problem. 4OR: A Quarterly Journal of Operations Research 4 (3), 255–258.
5. Berretta, R., Rodrigues, L. F., 2004. A memetic algorithm for a multistage capacitated lot-sizing problem. International Journal of Production Economics 87 (1), 67 – 81.
6. Boudia, M., Prins, C., 2009. A memetic algorithm with dynamic population management for an integrated production-distribution problem. European Journal of Operational Research 195 (3), 703 – 715.
7. Brest, J., Mauˇcec, M. S., 2008. Population size reduction for the differential evolution algorithm. Applied Intelligence 29 (3), 228–247.
8. Brest, J., Maucec, M. S., 2011. Self-adaptive differential evolution algorithm using population size reduction and three strategies. Soft Computing - A Fusion of Foundations, Methodologies and Applications 15 (11), 2157–2174.
9. Burke, E. K., De Causmaecker, P., De Maere, G., Mulder, J., Paelinck, M., Berghe, G. V., 2010. A multi-objective approach for robust airline scheduling. Computers and Operations Research 37, 822–832.

10. Burke, E. K., Kendall, G., Soubeiga, E., 2003. A tabu search hyperheuristic for timetabling and rostering. Journal of Heuristics 9 (6), 451– 470.
11. Caponio, A., Cascella, G. L., Neri, F., Salvatore, N., Sumner, M., 2007. A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives. IEEE Transactions on System Man and Cybernetics-part B, special issue on Memetic Algorithms 37 (1), 28– 41.
12. Caponio, A., Neri, F., 2009. Integrating cross-dominance adaptation in multi-objective memetic algorithms. In: C.-K. Goh, Y.-S. Ong, K. T. (Ed.), Multi-Objective Memetic Algorithms. Vol. 171 of Studies in Computational Intelligence. Springer, pp. 325–351.
13. Caponio, A., Neri, F., Tirronen, V., 2009. Super-fit control adaptation in memetic differential evolution frameworks. Soft Computing-A Fusion of Foundations, Methodologies and Applications 13 (8), 811–831.
14. Chakhlevitch, K., Cowling, P., 2008. Hyperheuristics: Recent developments. In: Cotta, C., Sevaux, M., S¨orensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Vol. 136 of Studies in Computational Intelligence. Springer-Verlag, Berlin Heidelberg, pp. 3–29.
15. Cotta, C., Troya, J., 2003. Embedding branch and bound within evolutionary algorithms. Applied Intelligence 18(2), 137–153.
16. Cowling, P., Kendall, G., Soubeiga, E., 2000. A hyperheuristic approach to scheduling a sales summit. In: Proceedings of the Third International Conference on Practice and Theory of Automated Timetabling. Vol. 2079 of Lecture Notes in Computer Science. Springer, pp. 176–190.
17. Fran¸ca, P. M., Gupta, J. N. D., Mendes, A. S., Moscato, P., Veltnik, K. J., 2005. Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Computers and Industrial Engineering 48, 491–506.

CHAPTER 7

CONSTRAINT HANDLING

INTRODUCTION

The central problem in applications of genetic algorithms is that of constraints few approaches to the constraint problem in genetic algorithms have previously been proposed. One of these uses penalty functions as an adjustment to the optimized objective function, other approaches use "decoders" or "repair" algorithms, which avoid building an illegal individual, or repair one, respectively. However, these approaches suffer from the disadvantage of being tailored to the specific problem and are not sufficiently general to handle a variety of problems.

It is also a theoretically challenging subject since a great deal of intractable problems (NP-hard, NP-complete, etc.) are constrained. The presence of constraints has the effect that not all possible combinations of variable values represent valid solutions to the problem at hand. Unfortunately, constraint handling is not straightforward in an EA, because the variation operators (mutation and

recombination) are typically "blind" to constraints. That is, there is no guarantee that even if the parents satisfy some constraints, the offspring will satisfy them as well. Based on this classification of constrained problems, we discuss what constraint handling means from an EA perspective, and review the most commonly applied EA techniques to treat constraints. Analyzing these techniques, we identify a number of common features and arrive at the conclusion that the presence of constraints is not harmful, but rather helpful in that it provides extra information that EAs can utilize.

7.1. CONSTRAINT HANDLING TECHNIQUES

The various existing techniques are available, for instance in. They usually involve a distinction between the following classes:

- Elimination of infeasible individuals;
- Penalization of the objective function;
- Dominance concepts;
- Preservation of feasibility;
- Infeasible individuals repairing;
- Hybrid methods.

7.1.1. Elimination

This method, also called "death penalty method", consists in rejecting infeasible individuals. The most common way to implement this strategy is to set their fitness equal to 0, which prevents infeasible solutions to pass the selection step. This method is very simply implemented, but encounters problems for harshly constrained problems. In addition, its second weakness is that no information is taken from the infeasible space, which could help to guide the search towards the global optimum. Nevertheless, this technique constitutes a first valid approach when no feature allows to previously determine a specific problem-fitted method.

7.1.2. Penalty Functions

This second class is certainly the most popular one, because of its understanding and implementation simplicity. The constrained problem is transformed into an unconstrained one by introducing the constraints in the objective function via penalty terms. Then, it is possible to formulate this penalty term according to a wide diversity of techniques. Firstly, it is of common knowledge that the penalization will be more efficient if its expression is related to the amount of constraint violation than to the violated constraint number.

Let us consider the classical optimization problem formulation:

$$\text{Min } f(x) \quad \text{s.t. } g_j(x) \leq 0, \quad j = 1, m \tag{1}$$

Then, with the unconstrained formulation including the penalty term, the new criterion F to minimize can be generally written as follows:

$$F(x) = f(x) + \sum_{j=1}^{m} R_j \max[0, g_j(x)]^{\beta} \tag{2}$$

Most of time, the penalty is expressed under a quadratic form, corresponding to β equal to 2. Equality constraints such as $h_k(x) = 0$ can be reformulated as $|h_k(x)| - e \leq 0$, where ε is a very small value. Then, the R_j factor can be expressed in many ways, showing various complexity and solution efficiency for the tackled problem. General principles can however be stated in order to guide the development of performing penalisation strategies.

The first one points out that, in most problems, the global optimum is located on the feasible space boundary. So, on the one hand, if the influence of the penalty factor is too important, the pressure exerted to push the individuals inside the feasible space will be too strong, preventing them from heading for more promising regions. Furthermore, in case of disjointed feasible spaces, a too high penalty factor can confine the population to one feasible region without allowing individuals to cross infeasible zones and

head for other feasible regions (where the global optimum could be located). On the other hand, a too low penalty factor can lead to an exhaustive search in the infeasible space, visiting regions where the objective function is very low but that are strongly infeasible.

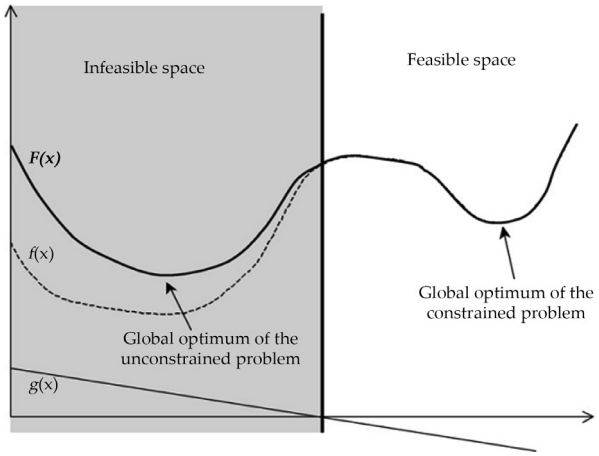


Figure. 1. Too weak penalty factor.

In addition, it is commonly admitted that the penalty term should be preferentially pretty low at the beginning of the search, in order to explore a wide region of the search space. At the end of the run, promising regions should be determined yet. It is then more relevant to have a high penalty term, to intensify the search on these zones by forcing the individuals to satisfy the constraints.

According to these principles, a great variety of penalization methods were implemented, some of them are recalled in. The simplest is the static penalty: a numerical value that will not vary during the whole search, is allocated to each factor Rj. Obviously, the drawback is that as many parameters as existing constraints have to be tuned without any known methodology. Normalizing the constraints enables however to reduce the number of parameters to be chosen from m to 1.

A modified static penalty technique is proposed in, in which violation levels are set for each constraint. So considering l levels in a problem with m constraints, it was shown that the method needs the tuning of m(2l + 1) parameters.

Another proposal is a dynamic penalty strategy, for which R_j is written as $(C \times t)^{a}$ where t is the generation number. Here, two parameters must be tuned, i.e. C and a. Common values are 0.5 and 2, respectively. Thus, this method enables to increase the pressure on infeasible solutions along the search. A similar effect can be obtained with a method presenting an analogy with Simulated Annealing:

$$R_j = \frac{1}{2t} \tag{3}$$

where τ is a decreasing temperature. It is necessary to determine initial and final temperatures, τi and τf, as well as a cooling scheme for τ. This technique has two special features. First, it involves a difference between linear and non-linear constraints. Feasibility as regard with the former is maintained by specific operators, so that only the latter has to be included in the annealing penalty term. In addition, the initial population is composed of clones of a same feasible individual that respects linear constraints.

Different approaches, called adaptive penalties, are based on learning from the population behavior in the generations. In, the penalty factor decreases (resp. increases) if the best individual was always feasible (resp. infeasible) during the k last generations. For indeterminate cases, the factor value is kept unchanged. This methodology imposes the tuning of the initial value for the penalty factor and of the number of learning generation's k.

New techniques now rest on self-adaptive penalty approaches, which also learn from the current run, without any parameter tuning. In, the constraints and the objective function are first normalized. Then, the method consists in computing the penalty

factor for constraint j at generation q as the product of the factor at generation q − 1 with a coefficient depending on the ratio of individuals violating constraint j at generation q. If this ratio is fewer to 50%, then the coefficient is inferior to 1 in order to favor individuals located in the infeasible side of the boundary. On the contrary, if the feasible individual's number is weak, the value increases up to 1 to have the population heading for the inside part of the feasible region. This operating mode enables to concentrate the search on the boundary built by each constraint, i.e. where the global optimum is likely to be located. The initial value is the ratio of the interquartile range of the objective function by the interquartile range of the considered constraint at first generation, which implicitly carries out normalization. No parameter is thus necessary in this method.

Another kind of self-adaptive penalty is proposed by Coello Colleo, but this one is based on the principle of co-evolution. In addition to the classical population P1 coding the tackled problem, the method considers a population P2 representing two penalty coefficients that enable to evaluate population P1 (w_1 for the amount of violation of all constraints and w_2 for the number of violated constraints). Thus, each individual of P1 is evaluated as many times as there are individuals in P2. Then, P1 evolves during a fixed number of generations and each individual of P2, i.e. each set of two penalty factors is evaluated. This mechanism is depicted in Figure 2. Basically, the evaluation is calculated as the average of all objective functions of P1 evaluated by each individual of P2. Then P2 evolves like in any GA process, given that one generation for P2 is equivalent to a complete evolution of P1. The evident drawback is the huge number of objective function evaluations, making this method computationally expensive. In addition, co-evolution involves the introduction and tuning of a new GAs parameters set: population size, maximum generation number, etc.

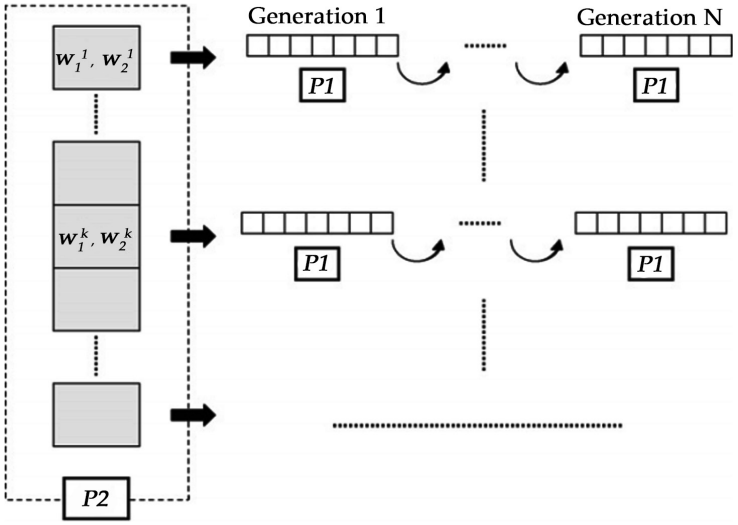


Figure 2. Self-adaptive penalty by co-evolution.

Finally, the technique proposed by Deb is half-way between classical penalisation and dominance-based methods. Superiority of feasible individuals on infeasible ones is expressed by the following penalised criterion:

$$\begin{cases} F(x) = f(x) & \text{if } g_j(x) \le 0, \quad j = 1, \ldots, m \\ F(x) = f_{\max} + \sum_{j}^{m} g_j(x) & \text{else} \end{cases} \tag{4}$$

f_{max} is the worst objective function value of all feasible solutions in the current population. The selection step is made out through a tournament process, but it could have been a Goldberg's roulette wheel as well. Most individuals are infeasible at the beginning of the search, hence the selection exerts a pressure exclusively towards the feasible space until enough feasible solutions are located. Without the stochastic effect of both tournament or roulette wheel, the minimization of the objective function would only occur when the feasible individual number exceeds the survivor number. In spite of the random effect introduced, efficient mutation procedures and sometimes niching methods are necessary to maintain diversity in the population and to prevent the search from being trapped in a local optimum.

Let us recall that niching helps to avoid that two solutions characterized by a close set of variables both survive. Metric considerations (usually, the euclidean distance) help to estimate how close an individual is from another one. In tournament between two feasible individuals is authorized only if the distance between them is lower than a constant threshold.

Among this profusion of techniques, some of the most classical methods were tested and evaluated in for some benchmark examples. Some methods are really adapted to particular problems, but the authors finally chose the static penalty technique, which is the simplest and the most generic one.

7.1.3. Dominance-Based Methods

This class of constraint handling techniques is based on principles drawn from multi objective optimization and, in particular, on the dominance concept. The first idea is thus to transform a constrained mono-objective optimization problem into an unconstrained multi objective problem, where each constraint represents a new criterion to be minimized. Sorting procedures based on the domination in the sense of Pareto (x dominates y if and only if it is better than y for at least one criterion and as good as y for the other ones) leads toward the ideal solution x^*: $g_j(x^*) \leq 0$ for $j = 1, ..., m$ and $f(x^*) \leq f(y)$ for all feasible y.

These concepts are used again by Coello Coello in the framework of mono-objective constrained optimization in order to state particular dominance rules setting the superiority of feasible solutions on infeasible ones:

1. An infeasible solution is dominated by a feasible one;
2. If both individuals are feasible, the one with the worst objective function is dominated;
3. If both individuals are infeasible, the one with greatest constraint violation is dominated.

These rules are implemented in a tournament: it is to note that this technique is finally exactly identical to Deb's one, who just

formalizes the rules as a penalty term added in the objective function.

Silva and Biscaia use a quite similar methodology for multi objective optimization by setting additional domination rankings. The constraints are normalized and four domination levels are created and defined according to the range of constraint violation amount. The union of the range of all levels is 1. Each individual is classified in these levels according to its largest constraint violation amount. Then, a positive integer is assigned to each domination level and added as a penalty term to the normalized objective functions. The selection is carried out through successive Pareto sorting rounds, during which non-dominated individuals are chosen until obtaining enough surviving individuals.

Thus, the two mentioned examples highlight how tenuous the boundary is between this constraint handling mode and some kinds of penalization techniques.

7.1.4. Other Techniques

The description of other techniques is merged because they are usually applicable only with some assumptions, or even defined exclusively for a particular problem. Firstly, in many cases, an adapted encoding method may enable to handle some constraints. A good is example is presented in, in which the number of 0-valued and 1-valued bits are coded instead of the bits themselves.

Besides, methods preserving solutions feasibility are usually based on specific crossover and mutation operators that are able to build, from feasible individual(s), one or several individuals that are feasible too. The GENOCOP algorithm provides a good example for linear problems. Equality constraints are removed by substitution of an equal number of variables, so that the feasible space is then a convex set defined by linear inequalities. Due to this property, genetic operators consisting of linear combinations can ensure the feasibility of the created solutions. Maintaining the feasibility can also be carried out through the use of decoders,

i.e., instructions contained in the chromosome that state rules for building a feasible solution.

Moreover, repairing infeasible chromosomes is a quite famous method. Indeed, in many cases of combinatorial optimization, it is easy to create rules that, starting from an infeasible individual, enable to modify its structure to get a feasible one. In for instance, repair procedures are implemented to act on individuals whose chromosome, resulting from crossover or mutation, has no physical meaning with regard to the used encoding method. However, repair rules are always devoted to the particular case of the studied problem and there is no existing heuristic for a general perspective. The particularity of the repair methods is also the possibility to replace in the population the infeasible individual by its repaired version or, on the contrary, to use this version only for the solution evaluation.

A generalized repair method proposed in involves the first order development of the constraint violation vector ΔV, according to Δ x, which represents a tiny variation in the optimization variables x:

$$\Delta V = \nabla_x V \times \Delta x, \quad \text{so } \Delta x = \nabla_x V^{-1} \times \Delta V \tag{5}$$

where matrix ∇xV is the constraint violation gradient according to variables x. So, if the constraint violation amount is known and by approximating numerically its gradient, it is theoretically possible to determine the repair vector Δx for the considered infeasible individual. Since ∇xV is usually not a square matrix, pseudoinverse computations provide an approximate inverse that can be used in (5). Despite its genericity ambition, it is predictable that such a method will only be applicable in some cases for which the functions and the nature of the involved variables are quite favourable.

This last technique can also be classified in the hybrid methods, just like the integration of Lagrange parameters in a penalty function, or the application of concepts drawn from fuzzy logic, etc.

7.2. CURRENT CONSTRAINT-HANDLING TECHNIQUES

Presents a set of recent constraint-handling techniques which have had a relatively high impact in the area. The number of approaches reviewed in this case is lower. This is due to the fact that the differences among approaches are, in this case, more focused on modifications to the elements of the NIA adopted, and not on the constraint-handling technique itself. From the list presented the first three and the seventh approach are all new constraint-handling techniques, while the fourth and the fifth are updated versions of constraint handling techniques previously discussed? The use of multi-objective concepts is now considered as a separate class due to its popularity in recent years. The approaches considered here are:

1. Feasibility rules
2. Stochastic ranking
3. ε-constrained method
4. Novel penalty functions
5. Novel special operators
6. Multi-objective concepts
7. Ensemble of constraint-handling techniques

7.2.1. Feasibility Rules

The three feasibility rules proposed for binary tournaments constitute an example of a constraint-handling technique that was proposed several years ago, but whose impact is still present. The popularity of this simple constraint-handling scheme lies on its ability to be coupled to a variety of algorithms, without introducing new parameters. The importance of combining these feasibility rules with other mechanisms (e.g., retaining infeasible solutions which are close to the feasible region) in order to produce a constraint-handling technique that is able to deal with problems having active constraints. However, such approach required a

dynamic decreasing mechanism for the tolerance value (ε) for equality constraints.

Mezura-Montes extended these feasibility rules to the selection process between target and trial vectors in differential evolution (DE). Although the resulting approaches were easy to implement, premature convergence was observed for some test problems and two additional modifications to the search algorithm (DE in this case) were required: (1) each target vector generated more than one trial vector (a user-defined parameter was added for this sake) and (2) a new DE variant was designed. Lampinen used a similar DE-based approach in. However, the third criterion was based on Pareto dominance in constraints space instead of the sum of constraint violation. Lampinen's approach was adopted by Kukkonen and Lampinen in their Generalized Differential Evolution (GDE) algorithm which showed promising results in a set of 24 benchmark problems. However, GDE had difficulties when facing more than three constraints. This seems to be the same limitation faced when attempting to use Pareto dominance in problems having four or more objective functions (the so-called many-objective optimization problems).

Feasibility rules have also been used for designing parameter control mechanisms in DE-based constrained numerical optimization. However, if self-adaptation mechanisms are incorporated as well, the computational cost of the approach may considerably increase, since more iterations will be required to collect enough information about the search as to make these self-adaptation mechanisms work in a proper manner.

Zielinski and Laur coupled DE with the feasibility rules in a greedy selection scheme between target and trial vectors. Their approach, which is indeed very simple to implement, presented some difficulties in high dimensionality problems with equality constraints. They also analyzed, in a further work, different termination conditions (e.g., improvement-based criteria, movement-based criteria distribution-based criteria) for their algorithm. They determined that the last criterion from the list indicated before was the most competitive. Zielinski also used

the feasibility rules in a study to adapt two DE parameters (F and CR) when solving CNOPs. They concluded that the adaptation mechanism was not as significant as expected in the performance of the algorithm. Furthermore, Zielinski and Laur studied the effect of the tolerance utilized in the equality constraints, where values between $\varrho = 1 \times 10^{-7}$ and $\varrho = 1 \times 10^{-15}$ allowed the algorithm, coupled with the feasibility rules, to reach competitive results.

The feasibility rules have also been adopted by DE-based approaches which use self-adaptive mechanisms to choose among their variants, such as SaDE. In this approach, sequential quadratic programming (SQP) is applied during some iterations to a subset of solutions in the population. Although this approach is very competitive, it heavily relies on the use of SQP, which may limit its applicability.

Brest used the feasibility rules in his self-adaptive approach called jDE-2, which also combines different DE variants into a single approach to solve CNOPs. A replacement mechanism to keep diversity in the population was implemented to eliminate those k worst vectors at every l generations with new randomly-generated vectors. This mechanism reflects the premature convergence that the feasibility rules may cause in some test problems despite providing competitive results in others mainly related with inequality constraints.

Landa and Coello also adopted the feasibility rules in an approach in which a cultural DE-based mechanism was developed, with the aim of incorporating knowledge from the problem into the search process when solving CNOPs. This approach adopts a belief space with four types of knowledge generated and stored during the search. Such knowledge is used to speed up convergence. The approach was able to provide competitive results in a set of benchmark problems. However, there are two main shortcomings of the approach: (1) it requires the use of spatial data structures for knowledge handling and (2) it also needs several parameters which must be defined by the user. Furthermore, spacial data structures are not trivial to implement.

Menchaca-Mendez and Coello Coello proposed a hybrid approach which combines DE and the Nelder-Mead method. The authors extended a variant of the Nelder-Mead method called Low Dimensional Simplex Evolution and used it to solve CNOPs. The set of feasibility rules are used in this case to deal with the constraints of the problems in both, the DE algorithm and the m-simplex-operator. However, a fourth rule which considers a tie between the sums of the constraint values, is also incorporated. The objective function value is used in this case to break this tie. This approach also adds some concepts from stochastic ranking. The approach was tested in some benchmark problems and the results obtained were highly competitive, while requiring a lower number of fitness function evaluations than state-of-the-art algorithms. However, the approach added a significant set of parameters which must be fine-tuned by the user, such as those related to the m-simplex-operator.

The use of feasibility rules coupled with a mechanism to force infeasible individuals to move to the feasible region through the application of search space reduction and diversity checking mechanisms designed to avoid premature convergence. This approach, which adopts an evolutionary agent system as its search engine, requires several parameters to be defined by the user and it was not compared against state-of-the-art NIAs to solve CNOPs. However, in the test bed reported by the authors the results were competitive.

The feasibility rules have been a popular constraint-handling mechanism in PSO-based approaches, too. Zielinski and Laur added feasibility rules into a local best PSO. The approach presented premature convergence in test problems with a high number of equality constraints due to the lack of a diversity maintenance mechanism. In a similar approach, but focused on mixed-variable optimization problems, Sun added feasibility rules to a global-best PSO. This approach was tested only in two engineering design problems.

He and Wang used feasibility rules to select the global best (gbest) and for updating the personal best (p best) of each particle in

a PSO-based approach designed to solve CNOPs. They used simulated annealing (SA) as a local search operator and applied it to the gbest particle at each generation. The approach was tested on a small set of benchmark problems as well as on a set of engineering design problems. The usage of SA improved the PSO performance. However, the main shortcoming of the approach is that it requires several user-defined parameter for both the PSO and the SA algorithms.

Toscano-Pulido and Coello Coello combined feasibility rules with a global-best PSO but required a mutation operator to avoid converging to local optimum solutions. This same problem (premature convergence) was tackled by using two mutation operators. Additionally, they also tackled this problem employing different topologies in local-best PSO algorithms. The evident need of a mutation operator showed that the feasibility rules combined with PSO may cause premature convergence.

Cagnina tackled the premature convergence of PSO combined with feasibility rules by using a global-local best PSO. However, the use of a dynamic mutation operator was also required. The results obtained by this approach showed evident signs of stagnation in some test problems. In a further version of this approach, a bi-population scheme and a "shake" operator were added.

In feasibility rules were used as a constraint-handling mechanism in an empirical study aimed to determine which PSO variant was the most competitive when solving CNOPs. The authors found that the version adopting a constriction factor performed better than the (popular) version that uses inertia weight. Furthermore, local-best was found to be better than global-best PSO.

The use of feasibility rules motivated the definition of the relative feasibility degree in which is a measure of constraint violation in pairwise comparisons. The aim of this work was to compare pairs of solutions but with a feasibility value based only on the values of their constraints and on the ratio of the feasible region of a given constraint with respect to the entire feasible region of the search space. The approach was coupled to DE and tested in

some benchmark problems. The convergence rate was better with respect to the original feasibility rules but the final results were not considerable better with respect to those obtained by other state-of-art algorithms.

Karaboga and Basturk and Karaboga and Akay changed a greedy selection based only on the objective function values by the use of feasibility rules with the aim of adapting an artificial bee colony algorithm (ABC) to solve CNOPs. The authors also modified the probability assignment for their roulette wheel selection employed to focus the search on the most promising solutions. The approach was tested on a well-known set of 13 test problems and the results obtained were comparable with those obtained by the homomorphous maps stochastic ranking and other approaches based on penalty functions. However, the approach modified one ABC operator adding a new parameter to be fine-tuned by the user.

Mezura-Montes and Cetina-Dom´ınguez extended Karabogas' approach by using feasibility rules as a constraint-handling technique but with a special operator designed to locate solutions close to the best feasible solution. This approach was tested on 13 test problems and the results that they obtained were shown to be better than those reported by Karaboga and Basturk. However, this approach added extra parameters related to the tolerance used to handle equality constraints. An improved version was proposed in where two operators were improved and a direct-search local operator was added to the algorithm. The approach provided competitive results in a set of eighteen scalable test problems but its main disadvantage was the definition of the schedule to apply the local search method.

The Bacterial Foraging Optimization Algorithm to solve CNOPs. The feasibility rules were used in the greedy selection mechanism within the chemotactic loop, which considers the generation of a new solution (swim) based on the random search direction (tumble). This approach, called Modified Bacterial Foraging Optimization Algorithm (MBFOA), considered a swarming mechanism that uses the best solution in the population as an

attractor for the other solutions. The approach was used to solve engineering design problems.

Mezura-Montes used feasibility rules as a constraint-handling mechanism in an in-depth empirical study of the use of DE as an optimizer in constrained search spaces. A set of well-known test problems and performance measures were used to analyze the behavior of different DE variants and their sensitivity to two user-defined parameters. From such analysis, the simple combination of two of them (DE/rand/1/bin and DE/best/1/bin) called Differential Evolution Combined Variants (DECV) was proposed by the authors. This approach is able to switch from one variant to the other based on a certain percentage of feasible solutions present in the population. The results obtained in a set of 24 test problems were competitive with respect to state-of-the-art algorithms. However, the performance of this approach strongly depends on the percentage used to perform the switch from one variant to the other and this value is problem-dependent.

Elsayed proposed two multi-operator NIAs to solve CNOPs. A four sub-populations scheme is handled by one of two options: (1) a static approach where each sub-population with a fixed size evolves by using a particular crossover and mutation operator and, at some periods of time, the sub-populations migrate the best solutions that they had found to another sub-population, and (2) an adaptive approach in which the size of each subpopulation varies based on the feasibility of the best solution in the population in two contiguous generations. This approach was tested in two versions: with a real-coded GA using four crossover-mutation combinations and also with DE adopting four DE mutation variants, all of them with binomial crossover. The latter version outperformed the former after being extensively tested in 60 benchmark problems. The approach requires the definition of some additional parameters related to the minimum size that a sub-population can have, as well as to the generational interval for migrating solutions among sub-populations.

Elsayed proposed a modified GA where a novel crossover operator called multi-parent crossover and also a randomized operator

were added to a real-coded GA to solve CNOPs. The feasibility rules were adopted as the constraint-handling mechanism. The approach was tested on a set of eighteen recently proposed test problems in 10D and 30D show in very competitive results. However, some disadvantages were found in separable test problems with a high dimensionality. The approach provided better results with respect to other approaches based on DE and PSO.

Elsayed compared ten different GA variants to solve CNOPs by using, in all cases, the feasibility rules as the constraint-handling technique. The crossover operators employed were triangular crossover, Simulated binary crossover, parent-centric crossover, simplex crossover, and blend crossover. The mutation operators adopted were non-uniform mutation and polynomial crossover. Statistical tests were applied to the samples of runs to provide confidence on the performances showed. An interesting conclusion of the comparison was that no GA was clearly superior with respect to the others GAs compared. Nonetheless, non-uniform mutation and polynomial mutation provided competitive results in 10D and 30D test problems.

Hamza proposed a DE algorithm to solve CNOPs where the feasibility rules were used as the constraint-hanlding mechanism and the population was divided in feasible and infeasible vectors. A constraint-concensus operator was applied to infeasible vectors so as to become them feasible. Even the approach showed competitive results in a set of tirtheen well-known test problems, the constraint-concensus operator requires gradient calculations, which were made by numerical methods.

In an interesting adaptation of the feasibility rules combined with the idea of focusing first on decreasing the sum of constraint violation, Tvrd´ık and Pol´akov´a adapted DE to solve CNOPs. If feasible solutions were present in the population, in a single cycle of the algorithm two generations were carried out, the first one based only on the sum of constraint violation and the second one based on the feasibility rules with a simple modification on the third rule, where between two infeasible solutions the one with

the lowest sum of constraint violation was preferred if it also had a better value of the objective function. The approach was tested on eighteen scalable test problems in 10D and 30D. Even some competitive results were obtained, premature convergence was generally observed in the approach because the isolated usage of the sum of constraint violation kept the algorithm for sampling, in a more convenient way, the feasible region of the search space.

The feasibility rules were used in a real-coded GA with simulated binary crossover and adaptive polynomial mutation. A special operator based on gradient information was employed to favor the generation of feasible solutions in presence of equality constraints. Even the results improved by the approach in such test problems, the parameter which mainly controls the special operator required a careful fine-tuning based on the difficulty of the test problem.

The feasibility rules were added by Tseng and Chen to the multiple trajectory search (MTS) algorithm to solve CNOPs. Those rules worked as the criteria to choose solutions in three region searches which allowed MTS to generate new solutions. The approach was able to provide feasible solutions in most of eighteen scalable test problems. However, it presented premature convergence.

Wang implicitly used feasibility rules to rank the particles in a hybrid multi-swarm PSO (HMPSO). They took inspiration from two (1) the way Liang and Suganthan constructed sub-swarms to promote more exploration of the search space and (2) the way Mu˜noz-Zavala used a differential mutation operator to update the local-best particle. The results agree with those found by Mezura-Montes and Flores-Mendoza, in which local-best PSO performs better than global-best PSO when solving CNOPs. The main shortcoming of the approach relies in its implementation due to the mechanisms added to PSO.

HMPSO was improved by Lui where the DE mutation operator was extended by using two other operators. The number of evaluations required by the improved approach, which was called PSO-DE, decreased with respect to those required by HMPSO in almost 50%. This approach was validated using some engineering design

problems but was not further tested on benchmark problems with higher dimensionalities.

The use of feasibility rules has been particularly popular in approaches based on artificial immune systems (AISs). The first attempts to solve CNOPs with an AIS were based on hybrid GA-AIS approaches, in which the constraint-handling technique was the main task performed by the AIS embedded within a GA. The AIS was evolved with the aim of making an infeasible solution (the antibody) as similar as possible (at a binary string level) as a feasible solution used as a reference (the antigen). After increasing the number of feasible solutions in the population, the outer GA continued with the optimization process. The main advantage of this technique is its simplicity. In further AIS-based approaches in which the clonal selection principle was adopted feasibility rules were incorporated as a way to rank antibodies (i.e., solutions) based on their affinity (objective function values and sum of constraint violation). In another approach based on a T-cell model, in which three types of cells (solutions) are adopted the replacement mechanism uses feasibility rules as the criteria to select the survivors for the next iteration.

Liu proposed the organizational evolutionary algorithm (OEA) to solve numerical optimization problems. When extending this approach to constrained problems, a static penalty function and feasibility rules are compared as constraint-handling techniques. As expected, the static penalty function required specific values for each test problem solved. Although the use of the static penalty function allowed OEA to provide slightly better results than the use of feasibility rules, such results were only comparable with respect to state-of-the-art algorithms used to solve CNOPs.

Sun and Garibaldi proposed a memetic algorithm to solve CNOPs. In this approach, the search engine is an estimation of distribution algorithm (EDA) while the local search operator is based on SQP. Some knowledge, called history, is extracted from the application of the local search and is given to the EDA with the aim of improving its performance. This knowledge consists in the application of a variation operator which uses the location of

the best solution found so far to influence the generation of new solutions. Feasibility rules are used as the constraint-handling mechanism in the selection process. In fact, a comparison against a version of this approach but using stochastic ranking as the mechanism to deal with the constraints showed that the feasibility rules were more suitable for this approach. The approach provided competitive results with respect to state-of-the-art algorithms. However, the local search adopted requires gradient information.

Adopted feasibility rules in their agent-based memetic algorithm to solve CNOPs. This approach is similar to a GA, and adopts the SBX operator to generate offspring which are subjected to a learning process that lasts up to four life spans. This actually works as a mutation operator whose use is based on a proper selection being made by each individual (agent). The measures used to select an operator are based on the success of each of them to generate competitive offspring. The communication among agents in the population is restricted to the current population. This approach seems to be sensitive to the value of the parameter associated with the communication mechanism. The results obtained were comparable with previously proposed approaches.

The biogeography based optimization (BBO) algorithm. The idea is to add a migration operator inspired on the blend crossover operator used in real-coded GAs. BBO is inspired on the study of distributions of species over time and space and it adopts two variation operators: migration (or emigration) and mutation. A habitat (solution) has a habitat suitability index, HSI (i.e., the fitness function). High-HSI solutions have a higher probability to share their features with low-HSI solutions by emigrating features to other habitats. Low-HSI solutions accept a lot of new features from high-HSI solutions by immigration from other habitats. Feasibility rules are used in this approach as the constraint-handling mechanism. The approach was found to be competitive with respect to PSO-based approaches and one GA-based algorithm. However, no further comparisons against state-of-the-art were reported.

Ali and Kajee-Bagdadi compared feasibility rules with respect to the superiority of feasible points proposed by Powell and Skolnick in a DEbased approach, in which a modified version of the pattern search method was used as a local search operator. They also compared their approach with respect to another based on a GA and found the former to be more competitive. The proposed DE-based approach presented a comparable performance with respect to other DE-based algorithms.

7.2.2. Stochastic Ranking

Stochastic ranking (SR) was originally proposed by Runarsson and Yao. SR was designed to deal with the inherent shortcomings of a penalty function (over and under penalization due to unsuitable values for the penalty factors). In SR, instead of the definition of those factors, a user-defined parameter called P_f controls the criterion used for comparison of infeasible solutions: (1) based on their sum of constraint violation or (2) based only on their objective function value. SR uses a bubble-sort-like process to rank the solutions in the population as shown in Figure 3.

```
Begin
    For i=1 to N
        For j=1 to P-1
            u=random(0,1)
            If (φ(I_j) = φ(I_{j+1}) = 0) or (u < P_f)
                If (f(I_j) > f(I_{j+1}))
                    swap(I_j,I_{j+1})
            Else
                If (φ(I_j) > φ(I_{j+1}))
                    swap(I_j,I_{j+1})
        End For
        If (not swap performed)
            break
    End For
End
```

Figure 3. Stochastic Ranking sort algorithm. I is an individual of the population. φ(Ij) is the sum of constraint violation of individual I_j . f(I_j) is the objective function value of individual I_j.

SR was originally proposed to work with an ES in its replacement mechanism which indeed requires a ranking process. However, it has been used with other NIAs where the replacement mechanism is quite different as in the approach reported by Zhang. In this case, the authors used SR with a DE variant proposed by Mezura-Montes in which more than one trial vector is generated per each target vector. Moreover, the parameter P_f was manipulated by a dynamic parameter control mechanism in order to conveniently decrease it, aiming to favor diversity during the initial generations of the search (infeasible solutions close to the feasible region are maintained) whereas only feasible solutions are kept during the final part of the search. The approach was compared against state-of-the-art algorithms and the results obtained were very competitive while requiring a low number of fitness function evaluations. However, the main disadvantage of this approach is that it requires the definition of the number of trial vectors generated by each target vector.

7.2.3. ε-constrained Method

One of the most recent constraint-handling techniques reported in the specialized is the ε-constrained method proposed. This mechanism transforms a CNOP into an unconstrained numerical optimization problem and it has two main components: (1) a relaxation of the limit to consider a solution as feasible, based on its sum of constraint violation, with the aim of using its objective function value as a comparison criterion, and (2) a lexicographical ordering mechanism in which the minimization of the sum of constraint violation precedes the minimization of the objective function of a given problem. The value of ε, satisfying $\varepsilon > 0$, determines the so-called ε-level comparisons between a pair of solutions $\vec{x}_1$ and $\vec{x}_2$ with objective function values $f(\vec{x}_1)$ and $f(\vec{x}_2)$ and sums of constraint violation $\phi(\vec{x}_1)$ and $\phi(\vec{x}_2)$ as indicated in Equations (6) and (7).

$$(f(\vec{x}_1), \phi(\vec{x}_1)) <_\varepsilon (f(\vec{x}_2), \phi(\vec{x}_2)) \Leftrightarrow \begin{cases} f(\vec{x}_1) < f(\vec{x}_2) & \text{, if } \phi(\vec{x}_1), \phi(\vec{x}_2) \leq \varepsilon \\ f(\vec{x}_1) < f(\vec{x}_2) & \text{, if } \phi(\vec{x}_1) = \phi(\vec{x}_2) \\ \phi(\vec{x}_1) < \phi(\vec{x}_2) & \text{, otherwise} \end{cases} \quad (6)$$

$$(f(\vec{x}_1), \phi(\vec{x}_1)) \leq_\varepsilon (f(\vec{x}_2), \phi(\vec{x}_2)) \Leftrightarrow \begin{cases} f(\vec{x}_1) \leq f(\vec{x}_2) & \text{, if } \phi(\vec{x}_1), \phi(\vec{x}_2) \leq \varepsilon \\ f(\vec{x}_1) \leq f(\vec{x}_2) & \text{, if } \phi(\vec{x}_1) = \phi(\vec{x}_2) \\ \phi(\vec{x}_1) < \phi(\vec{x}_2) & \text{, otherwise} \end{cases} \quad (7)$$

As can be seen, if both solutions in the pairwise comparison are feasible, slightly infeasible (as determined by the ε value) or even if they have the same sum of constraint violation, they are compared using their objective function values. If both solutions are infeasible, they are compared based on their sum of constraint violation. Therefore, if ε = ∞, the ε-level comparison works by using only the objective function values as the comparison criteria. On the other hand, if ε = 0, then the ε-level comparisons<0 and $\leq_0$ are equivalent to a lexicographical ordering in which the minimization of the sum of constraint violation $\phi(\vec{x})$ precedes the minimization of the objective function $f(\vec{x})$, as promoted by the use of feasibility rules.

Takahama and Sakai have an earlier approach called the α-constrained method. In this case, the authors perform α-level comparisons which work in a similar way as those of the ε-constrained method. However, unlike the ε value which represents a tolerance related to the sum of constraint violation, the α value is related to the satisfaction level of the constraints for a given solution. Therefore, the condition to consider the objective function as a criterion in a pairwise comparison is based on the aforementioned satisfaction level of both solutions. If both levels are higher than a $0 \leq \alpha \leq 1$ value, the comparison can be made by using the objective function value, regardless of the full feasibility of the solutions. The main drawback of the α-constrained method with respect to the ε-constrained method is that the first may require user-defined parameters to compute the satisfaction level while the second uses the sum of constraint violation which requires no additional parameters. Nonetheless, in both mechanisms, the careful fine-tuning of α and ε remains as the main shortcoming. The authors have proposed dynamic

mechanisms which have allowed these two algorithms to provide competitive results.

The α-constrained method was coupled to a GA in while the use of the Nelder-Mead method was reported by the same authors. The results obtained by using multiple simplexes allowed the approach to obtain competitive results with respect to those found by SR. Wang and Li adopted the α-constrained method in using DE as their search engine, and improved the results reported in. Also, the ε-constrained method was combined with a hybrid PSO-GA algorithm. The approach considered the reproduction for particles as in a GA with the goal to tackle the premature convergence observed in a version in which the α-constrained method was coupled only to PSO. The hybrid approach was tested only in one benchmark function and two engineering design problems.

A successful attempt to find a more suitable search algorithm for the εconstrained method was reported in where a DE variant (DE/ rand/1/- exp) and a gradient-based mutation operator (acting as a local search engine) were employed. This version obtained the best overall results in a competition on constrained real-parameter optimization in 2006, in which a set of 24 test problems were solved. Gradient-based mutation was applied to newly infeasible generated trial vectors in order to make them feasible. Evidently, the main limitation of this sort of approach is that gradient information must be computed. Also, additional parameters must be fine-tuned by the user in this approach. Finally, it is worth remarking that there seem to be no studies that analyze the role of adopting gradient-based information in an EA used for constrained optimization.

Further improvements have been proposed to the ε-constrained method. In improved the dynamic control for the ε value by using an adaptive approach which allowed a faster decrease in its value if the sum of constraint violation was reduced quickly enough during the search process. This mechanism produced improved results when dealing with CNOPs having equality constraints. However, in this case, there is also an additional user-defined parameter, which is related to the adaptation process.

Additionally, the authors did not analyze the performance of this variant in CNOPs that have only inequality constraints.

In Takahama and Sakai improved their approach by adding a de creasing probability on the use of the gradient-based mutation. They also introduced two new mechanisms to deal with boundary constraints: (1) one based on a reflecting back process for variable values lying outside the valid limits when DE mutation was applied, and (2) another one that consisted in assigning the limit value to a variable lying outside a boundary when the gradient-based mutation was computed. With the aforementioned changes, the authors could obtain feasible solutions for one highly difficult problem known as g22. The main drawback of this improved version was the addition of user-defined parameters for the dynamic mechanism used by the gradient-based mutation operator. A further improved version of the aforementioned algorithm was proposed by Takahama and Sakai in where an archive to store solutions and the ability of a vector to generate more than one trial vector were added. The approach has provided one of the most (if not the most) competitive performance in different sets of test problems. However, the algorithm still depends on the gradient-based mutation to provide such competitive results.

Motivated by its competitive performance, the ε-constrained method has been adopted as a constraint-handling technique in other proposals. This is the case of the jDE algorithm proposed by Brest in which the authors propose to self-adapat the parameters of DE using stochastic values. In a version called ε-jDE the ε-constrained method was one of the improvements proposed, besides the use of additional DE variants and a reduction scheme of the population size. Brest also added a novel way to adapt the ε value, but additional user-defined parameters were introduced. However, the results obtained by ε-jDE were highly competitive in a set of 24 well-known benchmark problems. An improved version called jDEsoco was proposed in where an ageing mechanism to replace those solutions stagnated in a local optimum was added. Moreover, only the 60% of the population was compared by the ε-constrained method and the remaining 40% was compared by

only using the objective function value. The results were improved but two parameters, the population ratio and an ageing probability were added to the algorithm.

Zeng employed the ε-constrained method with a ε variation process based on the dynamic decrease mechanism originally proposed. A crossover operator biased by the barycenter of the parents, plus a uniform mutation were used as variation operators. The approach was tested in 24 test problems and the results were found to be competitive with respect to the ε-constrained DE. An improved version of this approach was proposed by Zhang where a gradient-based mutation similar to the one proposed by Takahama and Sakai was added. The results obtained by this approach were compared with respect to those obtained by the ε-constrained DE. The approach added parameters related with the variation of the ε value as well as those required by the gradient-based mutation.

The ε-constrained method within the ABC algorithm. Additionally, a dynamic mechanism to decrease the tolerance for equality constraints was considered. The results obtained outperformed those reported by a ABC version in which feasibility rules were used as the constraint-handling technique. However, this approach showed premature convergence in some test problems having high dimensionality.

7.2.4. Novel Penalty Functions

In spite of the fact that the two types of constraint-handling techniques discussed avoid the use of a penalty function, there are proposals based on such penalty functions which provide very competitive results. Here, we will briefly review the most representative work in this direction.

Xiao used the so-called KS function in a static penalty function to solve CNOPs. However, even when the approach was competitive in some test problems, it was clearly outperformed in others.

Deb and Datta revisited the static penalty function by proposing a method to compute a suitable value for a single penalty factor,

assuming the normalization of the constraints. As a first step a bi-objective problem was solved by a multi-objective evolutionary algorithm (MOEA). The first objective was the original objective function while the second was the sum of constraint violation $\varphi = 0$. Furthermore, φ was restricted by a tolerance value (in a similar way as the ε-constrained method but with a fixed value in this case). The tolerance value was determined by a userdefined parameter based on the number of constraints of the problem. After a certain number of generations (also defined by the user), a cubic curve to approximate the current obtained Pareto front was generated by using four points whose φ values were a small tolerance. The penalty factor was then defined by calculating the corresponding slope at $\varphi = 0$. After that, a traditional static penalty function was used to solve the original CNOP by using a local search algorithm (Matlab's fmincon() procedure was used by the authors) using the solution with the lowest φ value from the population of the MOEA as the starting point for the search. The termination criterion for the local search algorithm was the feasibility of the final solution combined with a small tolerance for the difference between objective function values of the starting point and the final one. The approach was tested on a set of six benchmark problems in which it obtained competitive results, while requiring a significant lower number of evaluations with respect to those reported by other state-of-the-art NIAs. The approach, however, requires the calibration of the MOEA as well as a tolerance value for the constraint related to the sum of constraint violation. Additionally, it also requires the number of generations to define the interval of use for the local search and, finally, the tolerance for the termination criterion of the local search. It is worth noting that this approach considered only inequality constraints. In Datta and Deb extended their approach to deal with equality constraints, too. The extension consisted in two main changes: (1) the punishment provided by the penalty value obtained by the bi-objective problem was increased if the local search failed to generate a feasible solution and (2) the small tolerance used for choosing the four points employed to approximate the cubic curve was relaxed. Both changes were motivated by the difficulties to generate feasible solutions caused

by the presence of equality constraints. The results obtained in eight well-known test problems were highly competitive with respect to two state-of-the-art approaches based on PSO and DE.

Tasgetiren and Suganthan proposed the use of a dynamic penalty function coupled with a multi-population differential evolution algorithm. In this approach, the authors allowed a user-defined number of sub-populations to evolve independently. However, the selection of the solutions to compute the differential mutation could be made by considering all sub-populations. Furthermore, a regrouping process, similar to a recombination operator among best solutions in each sub-population, was carried out after a number of generations, defined by the user. The approach was tested on 24 test problems, and the authors reported a high sensitivity of their approach to the parameters related with the severity of the penalty.

Farmani and Wright proposed a two-parts adaptive penalty function in which no penalty factors need to be defined by the user. The first part increases the fitness of the infeasible solutions with a better value of the objective function with respect to the best solution in the current population. The best solution can be the feasible solution with the best objective function value. However, if no feasible solutions are present in the population, the best solution is the infeasible solution with the lowest sum of constraint violation. This first part of the penalization focuses on promoting diversity in promising regions of the search space, regardless of their feasibility. The second part modifies the fitness values of the worst infeasible solutions (those with the highest sum of constraint violation and a poor objective function value) aiming to make them similar to the fitness of the solution with the worst value of the objective function. The aim is to generate more solutions in the boundaries of the feasible region but with better values of the objective function. In spite of its lack of user-defined penalty factors, the approach was computationally expensive, since it required more than one million evaluations to provide competitive results in a set of 11 test problems.

Puzzi and Carpinteri explored a dynamic penalty function based on multiplications instead of summations in a GA-based approach. However, this approach performed well in problems having only inequality constraints.

Tessema and Yen used the number of feasible solutions in the current population to determine the penalty value assigned to infeasible solutions in a two-penalty based approach. This parameterless penalty function allows, based on the feasibility of solutions in the population, to favor slightly infeasible solutions having a good objective function value, as promoted. This is done in the selection process by assigning such solutions a higher fitness value. The approach obtained competitive results in 22 test problems. However, the number of evaluations required was higher (500, 000) than that required by other state-of-the-art approaches (they require around 250, 000 evaluations). Furthermore, three mutations operators (which require three mutation probabilities defined by the user) are required to maintain the explorative capabilities of the approach.

Mani and Patvardhan explored the use of an adaptive penalty function in a two-population-GA-like-based approach in which the first population evolves by using a parameter-free adaptive penalty function based on the objective function and the constraint violation of the best solution available so far in the population. The other population evolves based on feasibility rules. Then, both populations exchange their best solutions plus an additional percentage of randomly chosen solutions. The approach was tested on a set of test problems. However, the approach required parameters related to the migration process as well as the variation operators, as well as a local search mechanism based on gradient information.

In an analogous way as Coello used co-evolution to optimize penalty factors to solve CNOPs by using two-nested GAs, He used two PSO algorithms instead. Their approach was used to solve a set of engineering design problems and the results were encouraging. However, as in the approach using GAs, this one

requires the definition of parameter values for the two PSO algorithms.

Wu proposed an AIS which combines the metaphor of clonal selection with idiotypic network theories. To deal with CNOPs, an adaptive penalty function was defined to assign its affinity to each antibody. Different operators based on the clonal selection principle, affinity maturation and the bone marrow operator were applied to generate new solutions. The approach was tested on four benchmark nonlinear programming problems and four generalized polynomial programming (GPP) problems.

7.2.5. Novel special operators

Leguizam´on and Coello Coello proposed a boundary operator based on conducting a binary search between a feasible and an infeasible solution. Furthermore, three strategies to select which constraint (if more than one is present in a CNOP) is analyzed. The search algorithm was an ACO variant for continuous search spaces. The approach provided highly competitive results, mostly, as expected, in problems having active constraints. However, it was outperformed in others. The main disadvantage of the approach is the need of an additional constraint-handling technique (a penalty function was used in this case) to deal with solutions which are on the boundary of the constraint treated but violate other constraints. Furthermore, no other search algorithms which are more popular in the solution of CNOPs (e.g., DE, ES) have been coupled to this proposed boundary operator.

Huang proposed a boundary operator in a two-population approach. The first population evolves by using DE as the search engine, based only on the objective function value (regardless of feasibility). The second population stores only feasible solutions and the boundary operator uses solutions from both populations to generate new solutions, through the application of the bisection method in the boundaries of the feasible region. Furthermore, the Nelder-Mead simplex method was used as a local search operator applied to the best feasible solutions. Unlike Leguizam´on and

Coello's proposal, this approach does not require an additional constraint-handling technique, but a feasible solutions is needed at the beginning of the process. The approach was tested only in a few problems having only inequality constraints and it required different parameter values for each test problem, showing some sensitivity to them.

The Constraint Quadratic Approximation (CQA), which is a special operator designed to restrict an evolutionary algorithm (a GA in this case) to sample solutions inside an object with the same dimensions of the feasible region of the search space. This is achieved by a second-order approximation of the objective function and an equality constraint, which is updated at each generation. A subset of solutions from the population was used to build the quadratic approximations. The operator was applied based on a number of generations defined by the user. Moreover, a static penalty function was used to guide the GA search and the equality constraint was transformed into two inequality constraints by using a small ◎ tolerance. The approach was tested on a small set of problems but it could only deal with one quadratic equality constraint. With the aim of solving CNOPs with more than one equality constraint, Peconick proposed the Constraint Quadratic Approximation for Multiple Equality Constraints (CQA-MEC). This was achieved by an iterative projection algorithm which is able to find points satisfying the approximated quadratic constraints with a low computational overhead. However, CQA-MEC still requires the static penalty function to work as its predecessor (CQA). Araujo extended the approaches to deal with multiple inequality constraints by using an special operator in which the locally convex inequality constraints are approximated by quadratic functions, while the locally non-convex inequality constraints are approximated by linear functions. The dependence of the static penalty function remains in this last approach.

Ullah proposed an agent-based memetic algorithm to solve CNOPs, in which the authors adopt a special local operator for equality constraints, which is one of five life span learning processes. After a selection process in which pairs of agents (i.e., solutions) are

chosen based on their fitness and location in the search space, the SBX operator is applied. Thereafter, the special operator for equality constraints is applied to some individuals in the population as follows: the satisfaction of a randomly chosen equality constraint is verified for a given solution. If it is not satisfied, a decision variable, also chosen at random, is updated with the aim to satisfy it. If the constraint is indeed satisfied, two other variables are satisfied in such a way that the constraint is still satisfied (i.e., the constraint is sampled). This special operator is only applied during the early stages of the search because it reduces the diversity in the population. These processes are applied based on their success. The approach was tested on a set of benchmark problems with equality constraints and the results were promising. However, the approach requires additional parameters to be defined by the user (e.g., the number of generations during which the operator must be applied, the number of decision variables to be updated in the equality constraint). In fact, the authors do not provide any guidelines regarding the way in which these parameters must be tuned.

Lu and Chen proposed an approach called self-adaptive velocity particle swarm optimization (SAVPSO) to solve CNOPs. This approach relies on an analysis based on three elements: (1) the position of the feasible region with respect to the whole search space, (2) the connectivity and the shape of the feasible region, and (3) the ratio of the feasible region with respect to the search space. As a result of this analysis, the velocity update formula was modified in such a way that each particle has the ability to selfadjust its velocity according to the aforementioned features of the feasible region. The fitness of a solution is assigned based on its feasibility: feasible solutions are evaluated by their objective function value, while infeasible solutions are evaluated by their sum of constraint violation. The approach was tested on a set of 13 benchmark problems. The approach, however, showed some sensitivity to some of its parameters.

Spadoni and Stefanini transformed a CNOP into an unconstrained search problem by sampling feasible directions instead of

solutions of a CNOP. Thereafter, three special operators, related to feasible directions for box constraints, linear inequality constraints, and quadratic inequality constraints, are utilized to generate new solutions by using DE as the search engine. The main contribution of the approach is that it transforms a CNOP into an unconstrained search problem without using a penalty function. However, it cannot deal with nonlinear (either equality or inequality) constraints.

Modified variation operators in NIAs in such a way that the recombination of feasible and infeasible solutions led to the generation of more feasible solutions. An adaptive mechanism to maintain infeasible solutions was added to the approach. This latter version was specifically based on DE's variation operators.

7.2.6. Multi-objective concepts

In spite of the fact that empirical evidence has suggested that multi objective concepts are not well-suited to solve CNOPs, there are highly competitive constraint-handling techniques based on such concepts.

Motivated by the idea of keeping suitable infeasible solutions Ray proposed the Infeasibility Driven Evolutionary Algorithm (IDEA) whose replacement process requires the definition of a proportion of infeasible solutions to remain in the population for the next generation. IDEA works in a similar way as NSGA-II. Nonetheless, an additional objective, besides the original objective function, is added. This objective consists on the constraint violation measure, whose value is computed as follows: each individual in the population has a rank for each constraint of the CNOP being solved and each rank value depends on the constraint violation value for such solution (lower values are ranked higher because they represent a smaller violation for a constraint). If a solution satisfies the constraint, a zero rank is assigned to it. After each solution is ranked for each constraint, the violation measure is computed as the sum of ranks per solution. After the offspring are generated, the union of parents and offspring is split in two sets,

one with the feasible solutions and the other with the infeasible ones. Non-dominated sorting is used to rank both sets separately and, based on the proportion of desired feasible solutions, they are chosen first from the infeasible set, the best ranked feasible solutions are chosen. IDEA is able to work with CNOPs and also with constrained numerical multi-objective optimization problems (CNMOPs). However, its performance has been more competitive when solving CNMOPs. The usage of local search, sequential quadratic programming in this case, was added to IDEA in the so-called Infeasibility Empowered Memetic Algotrithm (IMEA). The approach was tested in eigtheen scalable test problems and its performance improved with respect to the original IDEA when solving CNOPs. However, the local search algorithm adopted requires gradient calculation.

Reynoso-Meza proposed the spherical-pruning multi-objective optimization differential evolution (sp-MODE) to solve CNOPs, which were transformed into three-objective optimization problems, where the first objective was the original objective function, the second objective was the sum of constraint violation for inequality constraints and the third objective was the sum of constraint violation for equality constraints. An external archive was used to store non-dominated solutions. The sphere-pruning operator aims to find the best trade-off between feasibility and the optimization of the objective function. The approach required the definition for some parameter values depending of the number of constraints. However, the sphere-pruning operator might be an interesting operator to be applied in some parts of the search.

Wang proposed the use of Pareto dominance in a Hybrid Constrained EA (HCOEA) to solve a CNOP which was transformed into a bi-objective optimization problem. In this case, the first objective is the original objective function while the second one is the sum of constraint violation. A global search carried out by an EA is coupled to a local search operator based on a population division scheme and on the use of the SPX operator. In both cases, Pareto dominance is the criterion adopted to select solutions. The approach was tested in 13 benchmark problems and the results

were found to be competitive with respect to four state-of-the-art algorithms. However, the approach requires the definition of two crossover probabilities (one for the global search and another for the local search) as well as the number of subsets in which the population will be divided. HCOEA showed some sensitivity to this last parameter.

Wang proposed an steady state EA to solve a CNOP which was also transformed into a bi-objective problem. At each generation, a set of offspring solutions are generated by applying orthogonal crossover to a randomly chosen set of solutions in the current population. After that, the non-dominated solutions obtained from the set of offspring are chosen. If there are no feasible offspring, two randomly chosen solutions from the set of parents will be replaced by the offspring which dominate them. Alternative, solutions can also be chosen if they have a lower sum of constraint violation. Furthermore, the individual with the lowest sum of constraint violation will replace the worst parent in the population. If there are feasible offspring, based on a user-defined probability, two randomly chosen parents will be replaced by two offspring which dominate them. Otherwise, the worst parent, based on feasibility rules, will be replaced by one offspring. After the steady state replacement, all solutions are affected by an improved version of the BGA mutation operator based on a user-defined probability. The approach was tested in a set of 11 test problems and showed competitive results in some of them, but premature convergence was observed in others.

Wang in their adaptive trade-off mode (ATM) evolution strategy (ATMES), divided the search in three phases based on the feasibility of solutions in the population: (1) only infeasible solutions, (2) feasible and infeasible solutions, and (3) only feasible solutions. Owing to the fact that the CNOP was transformed into a bi-objective problem, the selection in the first phase was based on Pareto dominance. From the Pareto front obtained, the solutions were ranked in ascending order based on the sum of constraint violation and the first half was chosen to survive for the next generation and was deleted from the set. The process was

repeated until the desirable number of solutions was achieved. The second phase was biased by a fitness value which is adapted based on the percentage of feasible solutions in the population. The last stage was biased only by the objective function value. The approach provided competitive results in 13 test problems. However, ATMES required some parameters related to the tolerance for equality constraints and the step size employed by the ES used as the search engine. This same ATM was coupled by Wang with a NIA in which the offspring generation was as follows: An offspring was generated by one of two variation operator: (1) simplex crossover or (2) one of two mutations (uniform mutation or improved BGA mutation). The approach, besides being tested on a set of 13 benchmark problems, was used to solve some engineering design problems. The results obtained by the authors were found to be very competitive, but some cases of premature convergence were reported. Another improvement to the ATM, which is based on a shrinking mechanism proposed by Hern´andez-Aguirre was proposed. This approach, called Accelerated ATM (AATM), outperformed both the original ATM and the approach proposed by Hern´andez-Aguirre. However, additional parameters (which are required by the shrinking mechanism) were introduced by the authors. The ATM was coupled with DE in a recent approach showing an improvement in the results with respect to versions of the same algorithm. Liu used the ATM in an EA but with two main differences: (1) good point set crossover was used to generate offspring and (2) feasibility rules were the criteria to select solutions in the second stage of the ATM (at which there are feasible and infeasible solutions in the current population). The approach was tested in some benchmark problems. However, the performance of the proposed crossover operator was not found to be clearly better with respect to the version of this approach reported.

Gong and Cai used Pareto dominance in the many-objective space defined by the constraints of a problem as a constraint-handling mechanism in a DE-based approach. An orthogonal process was employed for both, generating the initial population and for applying crossover. Furthermore, the ◎-dominance

concept was adopted to update an external archive in which the non-dominated solutions found during the search were stored. Orthogonal crossover was applied after DE generated the offspring population. In fact, an intermediate child population was designed to store the offspring which were non-dominated with respect to their parents. The aim is to perform a non-dominance checking on the union of the parent population and the offspring population as a replacement mechanism at the end of each generation of the algorithm. Although the approach provided competitive results in a set of 13 test problems, the contribution of each of the additional mechanisms adopted is not clear. Additionally, no information is provided regarding the fine-tuning required for the parameters required by this approach.

Li et solved a CNOP which was also transformed into a bi objective optimization problem by using a PSO algorithm in which Pareto dominance was used as a criterion in the p best update process and in the selection of the local-best leaders in a neighborhood. In case of ties, the sum of constraint violation worked as a tie-breaker. A mutation operator was also added to keep the approach from converging prematurely. Additionally, a small tolerance was used to consider as feasible to solutions that were slightly infeasible (this is similar to the ε-constrained method but with a fixed value instead of a dynamic one). The approach was tested only on three engineering design problems.

Venter and Haftka also transformed a CNOP into a bi-objective optimization problem and used PSO as their search engine. However, the leader selection was based most of the time on the sum of constraint violation, while the rest of the time the criterion was one of the three following choices: (1) the original objective function, (2) the crowding distance or (3) Pareto dominance. The approach was tested on several benchmark problems and some engineering design problems.

Wang used a hybrid selection mechanism based on Pareto dominance and tournament selection into a Adaptive Bacterial Foraging Algorithm (ABFA) to solve CNOPs. The approach uses the so-called good nodes set method to initialize the population, to

perform crossover and to spread similar individuals throughout the search space. The approach was tested in a set of benchmark problems and in some engineering design problems, providing competitive results in both cases.

7.2.7. Ensemble of constraint-handling techniques

Many Optimization problems in science and engineering involve constraints. The presence of constraints reduces the feasible region and complicates the search process. Evolutionary algorithms (EAs) always perform unconstrained search. When solving constrained optimization problems, they require additional mechanisms to handle constraints. In the several constraint handling techniques have been proposed to be used with the EAs.

When solving constrained optimization problems, solution candidates that satisfy all the constraints are feasible individuals while individuals that fail to satisfy any of the constraints are infeasible individuals. One of the major issues in constraint optimization is how to deal with the infeasible individuals throughout the search process. One way to handle it is to completely disregard infeasible individuals and continue the search process with feasible individuals only. This approach may be ineffective as EAs are probabilistic search methods and potential information present in infeasible individuals can be wasted. If the search space is discontinuous, then the EA can also be trapped in one of the local minima. Therefore, different techniques have been developed to exploit the information in infeasible individuals. Michalewicz and Schoenauer grouped the methods for handling constraints within EAs into four categories: preserving feasibility of solutions, penalty functions, make a separation between feasible and infeasible solutions, and hybrid methods. A constrained optimization problem can also be formulated as a multi objective problem, but it is computationally intensive due to nondomination sorting.

According to the no free lunch (NFL) theorem, no single state-of-the-art constraint handling technique can outperform all others on every problem. Hence, solving a particular constrained problem

requires numerous trial-and-error runs to choose a suitable constraint handling technique and to fine tune the associated parameters. This approach clearly suffers from unrealistic computational requirements in particular if the objective function is computationally expensive or solutions are required in real-time. In this an ensemble of constraint handling techniques (ECHT) with four constraint handling techniques is proposed as an efficient alternative to the trial-and-error-based search for the best constraint handling technique with its best parameters for a given problem. In ECHT, each constraint handling technique has its own population and each function call is efficiently utilized by each of these populations. Different EAs such as differential evolution (DE) particle swarm optimizer, evolution strategies, evolutionary programming (EP) and others have been used to solve constrained optimization problems. In addition, EP and ES are similar. Recently the usage of DE to solve constrained problems is also gaining importance. Being a general concept, the ECHT can be realized with any of the existing EAs.

7.3. APPROACHES TO HANDLING CONSTRAINTS

In the discussion so far, we have not considered the nature of the domains of the variables. In this respect there are two extremes: they are all discrete or all continuous. Continuous CSPs are rather rare, so by default a CSP is discrete. For COPs this is not the case as we have discrete COPs (combinatorial optimization problems) and continuous COPs as well. Much of the evolutionary on constraint handling is restricted to one of these cases, but in fact the ways for handling constraints are practically identical – at least at the conceptual level. Therefore the following treatment of constraint handling methods is general, and we note simply that the presence of constraints will divide the space of potential solutions S into two or more disjoint regions, the feasible region (or regions) F containing those candidate solutions that satisfy the given constraints, and U, the infeasible region containing those that do not.

7.3.1. Penalty Functions

Penalty functions modify the original fitness function $f(\overline{x})$ applied to a candidate solution $\overline{x}$ such that $f'(\overline{x}) = f(\overline{x}) + P(d(\overline{x}, F))$, where $d(\overline{x}, F))$ is a distance metric of the infeasible point to the feasible region F (this might be simply a count of the number of constraints violated). The penalty function P is zero for feasible solutions, and it increases with distance from the feasible region (for minimisation problems).

For our knapsack problem one simple approach is to calculate the excess weight $e(\overline{x}) = \sum_i x(i) \cdot c(i) - C_{max}$, and then use the penalty function: where the fixed weight w is large enough that feasible solutions are preferred.

It is important to note that this approach assumes that it is possible to evaluate an infeasible point; although in this example it is, for many others this is not the case. This discussion is also confined to exterior penalty functions, where the penalty is only applied to infeasible solutions, rather than interior penalty functions, which apply penalties to all solutions based on distance from the constraint boundary in order to encourage exploration of this region.

The conceptual simplicity of penalty function methods means that they are widely used, and they are especially suited to problems with disjoint feasible regions, or where the global optimum lies on (or near) the constraint boundary. However, their successful use depends on a balance between exploration of the infeasible region and not wasting time, which places a lot of emphasis on the form of the penalty function and the distance metric.

If the penalty function is too severe, then infeasible points near the constraint boundary will be discarded, which may delay, or even prevent, exploration of this region. Equally, if the penalty function is not sufficient in magnitude, then solutions in infeasible regions may dominate those in feasible regions, leading to the algorithm spending too much time in the infeasible regions and possibly

stagnating there. In general, for a system with m constraints, the form of the penalty function is a weighted sum.

$$P(d(\overline{x}, F)) = \sum_{i=1}^{m} w_i \cdot d_i^{\kappa}(\overline{x})$$

where κ is a user-defined constant, often taking the value 1 or 2, and as the distance metrics $d_i(\overline{x})$ from the point $\overline{x}$ to the boundary for constraint i may be a simple binary value according to whether the constraint is satisfied, or a metric based on cost of repair.

Many different approaches have been proposed, and a good review is given in, where penalty functions are classified as constant, static, dynamic, or adaptive.

Static Penalty Functions

Three methods have commonly been used with static penalty functions, namely extinctive penalties (where all of the w_i are set so high as to prevent the use of infeasible solutions), binary penalties (where the value d_i is 1 if the constraint is violated, and zero otherwise), and distance-based penalties.

It has been reported that, of these three, the latter give the best results, and the contains many examples of this approach. This approach relies on the ability to specify a distance metric that accurately reflects the difficulty of repairing the solution, which is obviously problem dependent, and may also vary from constraint to constraint. The usual approach is to take the square of the Euclidean distance (i.e., set $\kappa = 2$) .

However, the main problem in using static penalty functions remains the setting of the values of w_i. In some situations it may be possible to find these by experimentation, using repeated runs and incorporating domain-specific knowledge, but this is a time-consuming process that is not always possible.

Dynamic Penalty Functions

An alternative approach to setting fixed values of w_i by hand is to use dynamic values, which vary as a function of time. A typical approach is that of, in which the static values w_i were replaced with a simple function of the form $s_i(t)=(w_i t)^{\alpha}$, where it was found that for best performance $\alpha \in \{1, 2\}$. Although this approach is possibly less brittle as a result of not using fixed (possibly inappropriate) values for w_i, the user must still decide on the initial values.

An alternative, which can be seen as the logical extension of this approach, is the behavioural memory algorithm of. Here a population is evolved in a number of stages – the same number as there are constraints. In each stage i, the fitness function used to evaluate the population is a combination of the distance function for constraint i with a death penalty for all solutions violating constraints j that different results may be obtained, depending on the order in which the constraints are dealt with.

Adaptive Penalty Functions

Adaptive penalty functions represent an attempt to remove the danger of poor performance resulting from an inappropriate choice of values for the penalty weights w_i. A second approach is that of, in which adaptive scaling (based on population statistics of the best feasible and infeasible raw fitnesses yet discovered) is coupled with the distance metrics for each constraint based on the notion of "near feasible thresholds". These latter are scaling factors for each distance metric, which can vary with time.

The Stepwise Adaptation of Weights (SAW) algorithm of can be seen as a population-level adaptation of the search space. In this method the weights w_i are adapted according to a simple heuristic: if the best individual in the current population violates constraint i, then this constraint must be hard and its weight should be increased. In contrast to the adaptive mechanisms, the updating function is much simpler. In this case a fixed penalty increment Δw is added to the penalty values for each of the constraints

violated in the best individual of the generation at which the updating takes place. This algorithm was able to adapt weight values that were independent of the EA operators and the initial weight values, suggesting that this is a robust technique.

7.3.2. Repair Functions

The use of repair algorithms for solving COPs with EAs can be seen as a special case of adding local search to the EA. In this case the aim of the local search is to reduce (or remove) the constraint violation, rather than to simply improve the value of the fitness function, as is usually the case.

The use of local search has been intensively researched, with attention focusing on the benefits of so-called Baldwinian versus Lamarckian learning. In either case, the repair algorithm works by taking an infeasible point and generating a feasible solution based on it. In the Baldwinian case, the fitness of the repaired solution is allocated to the infeasible point, which is kept, whereas with Lamarckian learning, the infeasible solution is overwritten with the new feasible point. Although the Baldwin vs. Lamarck debate has not been settled within unconstrained learning, many COP algorithms reach a compromise by introducing some stochasticity, for example Michalewicz's GENOCOP algorithm uses the repaired solution around 15% of the time.

For our knapsack example, a simple repair method is to change some of the gene values in $\overline{x}$ from 1 to 0. Although this sounds simple, this example raises some interesting questions. One of these is the replacement question just discussed; the second is whether the genes should be selected for altering in a predetermined order, or at random. In it was reported that using a greedy deterministic repair algorithm gave the best results, and certainly the use of a nondeterministic repair algorithm will add noise to the evaluation of every individual, since the same potential solution may yield different fitnesses on separate evaluations. However, it has been found by some authors that the addition of noise can assist the GA in avoiding premature convergence. In practice it is likely that the

best method is not only dependent on the problem instance, but on the size of the population and the selection pressure.

Although the knapsack example is fairly simple, in general defining a repair function may be as complex as solving the problem itself. One algorithm that eases this problem (and incidentally uses stochastic repair), is Michalewicz's GENOCOP III algorithm for optimisation in continuous domains. This works by maintaining two populations, one Ps of so-called search points and one P_r of 'reference points', with all of the latter being feasible. Points in P_r and feasible points from Ps are evaluated directly. When an infeasible point is generated in P_s it is repaired by picking a point in P_r and drawing a line segment from it to the infeasible point. This is then sampled until a repaired feasible point is found. If the new point is superior to that used from P_r, the new point replaces it. With a small probability (which represents the balance between Lamarckian and Baldwinian search) the new point replaces the infeasible point in P_s. It is worth noting that although two different methods are available for selecting the reference point used in the repair, both are stochastic, so the evaluation is necessarily noisy.

7.3.3. Restricting Search to the Feasible Region

In many COP applications it may be possible to construct a representation and operators so that the search is confined to the feasible region of the search space. In constructing such an algorithm, care must be taken in order to ensure that all of the feasible region is capable of being represented. It is equally desirable that any feasible solution can be reached from any other by (possibly repeated) applications of the mutation operator. The classic example of this is permutation problems.

For our knapsack problem, we could imagine the following operators. A randomised initialisation operator might construct solutions by starting with an empty set x(i)=0, ∀i and randomly picking elements i to flip the gene value from to 1 until adding the next value chosen would violate the cost constraint. This would give an initial population where the excess cost $e(\overline{x})$ was negative

for each member. For recombination, we could apply a slightly modified one-point crossover. For any given pair of parents, first we generate a random permutation of the values {1,,...,n−1} in which to consider the potential crossover points. In that order we consider the pairs of offspring created, accepting the first pair that is feasible. For mutation we apply bitwise mutation, accepting any move that changes a gene from 1 to 0, but only those from 0 to 1 that do not created excess cost. Again we might choose to do this in a random order to remove bias towards selecting items at the start of our representation.

It should be noted that this approach to solving COP, although attractive, is not suitable for all types of constraints. In many cases it is difficult to find an existing or design a new operator that guarantees that the offspring are feasible. Although one possible option is simply to discard any infeasible points and reapply the operator until a feasible solution is generated, the process of checking that a solution is feasible may be so time consuming as to render this approach unsuitable. However, there remains a large class of problems where this approach is valid and with suitable choice of operators can be very successfully applied.

7.3.4. Decoder Functions

Decoder functions are a class of mappings from the genotype space S′ to the feasible regions F of the solution space S that have the following properties:

- Every $z \in S$ must map to a single solution $s' \in F$.
- Every solution $s' \in F$ must have at least one representation $s' \in S'$.
- Every $s' \in F$ must have the same number of representations in S′ (this need not be 1).

Such decoder functions provide a relatively simple way of using EAs for this type of problem, but they are not without drawbacks. These are centred around the fact that decoder functions generally introduce a lot of redundancy into the original genotype space.

This arises when the new mapping is manyto-one, meaning that a number of potentially radically different genotypes may be mapped onto the same phenotype, and only a subset of the phenotype space can be reached.

Considering the knapsack example, a simple approach would leave the genotype, initialisation and variation operators unchanged. When constructing a solution, the decoder function could start at the left hand end of the string and interpret a 1 as take this item if possible ... If the cost limit is reached after considering, say, j of the n genes, then it is irrelevant what values the rest take, and so 2^{n-j} strings all map onto the same solution.

In a few cases it may be possible to devise a decoder function that permits the use of relatively standard representation and operators while preserving a one-to-one mapping between genotype and phenotype. One such example is the decoder for the TSP problem proposed by Grefenstette, which is well described by Michalewicz in. In this case a simple integer representation was used with each gene $a_i \in \{1,...,n+1-i\}$. This representation permits the use of common crossover operators and a bitwise mutation operator that randomly resets a gene value to one of its permitted allele values. The outcome of both of these operators is guaranteed to be valid. The decoder function works by considering an ordered list of cities, ABCDE, and using the genotype to index into this.

For example, with a genotype <4, 2, 3, 1, 1> the first city in the constructed tour is the fourth item in the list, i.e., D. This city is then removed from the list and the second gene is considered, which in this case points to B. This process is continued until a complete tour is constructed: <4, 2, 3, 1, 1> → DBEAC.

Although the one-to-one mapping means that there is no redundancy in the genotype space, and it permits the use of straightforward crossover and mutation operators, the complexity of the mapping function means that a small mutation can have a large effect, e.g., <3, 2, 3, 1, 1> → CBDAE. Equally, it can be easily shown that recombination operators no longer respect and propagate all features common to both solutions. Thus if the two

solutions <1, 1, 1, 1, 1>→ ABCDE and <5, 1, 2, 3, 1> → EACDB, which share the common feature that C occurs in the third position and D in the fourth undergo 1-point crossover between the third and fourth loci, the solution <5, 1, 2, 1, 1>→ EACBD is obtained, which does not possess this feature. If the crossover occurs in other positions, the edge CD may be preserved, but in a different position in the cycle.

In both of the examples given, the complexity of the genotype–phenotype mapping makes it very difficult to ensure locality and makes the fitness landscape associated with the search space highly complex, since the potential effects in fitness of changes at the left-hand end of the string are much bigger than those at the right-hand end. Equally, it can become very difficult to specify exactly the common features the recombination operators are supposed to be preserving.

7.4 APPLICATION EXAMPLE: GRAPH THREE-COLOURING

We illustrate the approaches outlined via the description of two different ways of solving a well-known CSP problem, graph three-colouring. This is an abstract version of colouring a political map so that no two adjacent areas (counties, states, countries) have the same colour. We are given a graph G = {v, e} with n = |v| vertices and m = |e| edges connecting some pairs of the vertices. The task is to find, if possible, an assignment of one of three colours to each vertex so that there are no edges in the graph connecting same-coloured vertices.

7.4.1. Indirect Approach

We begin by illustrating an indirect approach, transforming the problem from a CSP to a FOP by means of penalty functions. The most straightforward representation is using ternary strings of

length n = |v|, where each variable stands for one node, and the integers 1, 2, and 3 denote the three colours.

Using this standard GA representation has the advantage that all standard variation operators are immediately applicable. We now define two objective functions (penalty functions) that measure the amount of 'incorrectness' of a chromosome. The first function is based on the number of 'incorrect edges' that connect two nodes with the same colour, while the second relies on counting the 'incorrect nodes' that have a neighbour with the same colour. For a formal description let us denote the constraints belonging to the edges as ci (i = {1,...,m}), and let C^i be the set of constraints involving variable v_i (edges connecting to node i). Then the penalties belonging to the two options described can be expressed as follows:

$$f_1(\bar{s}) = \sum_{i=1}^{m} w_i \times \chi(\bar{s}, c_i),$$

where $\chi(\bar{s}, c_i) = \begin{cases} 1 & \text{if } \bar{s} \text{ violates } c_i, \\ 0 & \text{otherwise.} \end{cases}$

Respectively,

$$f_2(\bar{s}) = \sum_{i=1}^{n} w_i \times \chi(\bar{s}, C^i),$$

where $\chi(\bar{s}, C^i) = \begin{cases} 1 & \text{if } \bar{s} \text{ violates at least one } c \in C^i, \\ 0 & \text{otherwise.} \end{cases}$

Note that both functions are correct transformations of the constraints in the sense that for each $\bar{s} \in S$ we have that $\phi(\bar{s})$= true if and only if fi $f_i(\bar{s})$=0 (i = 1, 2). The motivation to use weighted sums in this example, and in general, is that they provide the possibility of emphazising certain constraints (variables) by giving them a higher weight. This can be beneficial if some constraints are more important or known to be harder to satisfy. Assigning them a higher weight gives a higher reward to a chromosome, hence the EA naturally focuses on these. Setting the weights can be done manually by the user, but can also be done by the EA itself on-the-fly as in the stepwise adaptation of weights (SAW) mechanism.

Now the EA for the graph three-colouring problem can be composed from standard components. For instance, we can apply a steady-state GA with population size 100, binary tournament

selection and worst fitness deletion, using random resetting mutation with $p_m = 1/n$ and uniform crossover with $p_c = 0.8$. Notice that this EA really ignores constraints; it only tries to minimise the given objective function (penalty function).

7.4.2. Mixed Mapping Direct Approach

For this problem, two of the direct approaches would be extremely difficult, if not impossible, to implement. Specifying either an initialization operator, or a repair function, to create valid solutions would effectively mean solving the problem, and since it is thought to be NP-complete it is unlikely that there is a polynomial time algorithm that could accomplish either of these.

However, we now present another EA for this problem, illustrating how constraints can be handled by a decoder. The main idea is to use permutations of the nodes as chromosomes. The phenotype (coloring) belonging to a genotype (permutation) is determined by a procedure that assigns colors to nodes in the order they occur in the given permutation, trying the colors in increasing order (1,2,3), and leaving the node uncolored if all three colors would lead to a constraint violation. Formally, we shift from the search space of all colorings $S = \{1, 2, 3\}^n$ to the space of all n-long permutations $S' = \{\bar{s} \in \{1, \ldots, n\}^n \mid s_i \neq s_j \quad i, j = 1, \ldots, n\}$, and the coloring procedure (the decoder) is the mapping from S′ to S′. At first glance this might not seem like a good idea as we still have constraints in the transformed problem – those that define the property of being a permutation in the definition of S′. However, we know from Sect. 4.5 that working in a permutation space is easy, as there are many suitable variation operators keeping the search in this space. In other words, we have various operators preserving the constraints defining this space.

An appropriate objective function for this representation can simply be defined as the number (weighted sum) of nodes that remain uncolored after decoding. This function also has the property that an optimal value (0) implies that all constraints are satisfied, i.e., all nodes are colored correctly. The rest of the EA

can again use off-the-shelf components: a steady-state GA with population size 100, binary tournament selection and worst fitness deletion, using swap mutation with $p_m = 1/n$ and order crossover with $p_c = 0.8$.

Looking at this solution at a conceptual level we can note that there are two constraint-handling issues. Primary constraint-handling concerns handling the constraints of the original problem, the graph three-coloring CSP. This is done by the mapping approach via a decoder. However, the transformed search space S′ in which the EA has to work is not free, rather it is restricted by the constraints defining permutations. This constitutes the secondary constraint handling issue that is solved by a (direct) preserving approach using appropriate variation operators.

REFERENCES

1. A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, Springer, Berlin, Germany, 2003.
2. C. A. Coello Coello, "Treating constraints as objectives for single-objective evolutionary optimization," Engineering Optimization 2000.
3. C. A. Coello Coello, "Use of a self-adaptive penalty approach for engineering optimization problems," Computers in Industry, 2000.
4. K. Deb, "An efficient constraint handling method for genetic algorithms," Computer Methods in Applied Mechanics and Engineering, 2001.
5. Kramer, A. Barthelmes, and G. Rudolph, "Surrogate constraint functions for CMA evolution strategies," in Proceedings of the 32nd German Annual Conference on Artificial Intelligence (KI '09), pp. 169–176, Paderborn, Germany, September 2009.
6. O. Kramer and H.-P. Schwefel, "On three new approaches to handle constraints within evolution strategies," Natural Computing, 2006.
7. O. Kramer, "Premature convergence in constrained continuous search spaces," in Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN '08), pp. 62–71, Springer, Dortmund, Germany, 2008.
8. O. Kramer, C.-K. Ting, and H. Kleine Büning, "A mutation operator for evolution strategies to handle constrained problems," in Proceedings of the 7th Conference on Genetic and Evolutionary Computation (GECCO '05), pp. 917–918, Washington, DC, USA, June 2005.
9. O. Kramer, C.-K. Ting, and H. Kleine Büning, "A new mutation operator for evolution strategies for constrained problems," in Proceedings of the IEEE Congress on Evolutionary Computation (CEC '05), pp. 2600–2606, Edinburgh, UK, September 2005.

10. O. Kramer, S. Brugger, and D. Lazovic, "Sex and death: towards biologically inspired heuristics for constraint handling," in Proceedings of the 9th Conference on Genetic and Evolutionary Computation (GECCO '07), pp. 666–673, ACM Press, London, UK, July 2007.
11. T. P. Runarsson, "Approximate evolution strategy using stochastic ranking," in Proceedings of the IEEE Congress on Evolutionary Computation (CEC '06), pp. 2760–2767, IEEE, Vancouver, Canada, July 2006.

INDEX

N

P

Q

R

S

T

V

Evolutionary Computing

Evolutionary computing is particularly suited to the adaptation (learning) of neural and fuzzy systems. Evolutionary computing is a versatile problem solver inspired by natural evolution. It models the critical elements of biological evolution and investigates the space of solution through gene inheritance, mutation and selection of the most suitable candidate solutions. Evolutionary computing is a significant field of study for adaptation and optimization. The approach actually originated from the Darwin concept of natural selection, also known as the survival of the fittest. Evolutionary computing has seen a significant increase in both theoretical and industrial applications over the last decade. Its scope has grown beyond its original sense of "biological evolution" to a broad range of nature-inspired computational algorithms and techniques, covering evolutionary, neural, ecological, social and economic computing, etc., in a unified context. In the Darwinian model, information gained by an individual cannot be transmitted to its genome and consequently passed on to the next generation. The synthesis of learning and evolution, represented by evolving neural networks, is more adaptable to a changing world. The interaction of learning with evolution accelerates evolution, which can take the form of the Lamarckian evolution or be based on the Baldwin effect. The Lamarckian strategy enables the inheritance of inherited traits in the genetic code of an individual's life so that the offspring will inherit its characteristics. Today, many research topics in evolutionary computing are not inherently "evolutionary".
The current book provides an overview of some recent advances in evolutionary computation. It concentrates on evolutionary computing, which is viewed as one of the most promising paradigms of computational intelligence. It covers a wide range of topics in optimization, learning and design using evolutionary approaches and techniques, and theoretical results in the computational time complexity of evolutionary algorithms. The dialects of evolutionary algorithms include genetic algorithms, evolutionary strategies, genetic programming, particle swarm optimization, ant colony optimization, artificial immune systems, estimation of distribution algorithms, differential evolution, and memetic algorithms. These evolutionary methods have proven their success on various hard and complex optimization problems. During this time, new metaheuristic optimization approaches, like evolutionary algorithms, genetic algorithms, swarm intelligence, etc., were being developed and new fields of usage in artificial intelligence, machine learning, combinatorial and numerical optimization, etc., were being explored. Some issues related to future development of evolutionary computation are also discussed. Presenting some new theoretical as well as practical aspects of evolutionary computation, this book will be of great value to undergraduates and graduate students in computer science.

Luciana Rocha, PhD, is currently an Associate Professor in the Department of Computer Science and Numerical Analysis. She has written several research papers and articles on algorithm development, optimization and metaheuristics, applications of soft computing methods in engineering, artificial neural networks, and computational solid mechanics.

ISBN 978-1-98467-775-4

9 781984 677754 00000

BIBLIOTEX
Digital Library