Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Structured population in evolutionary algorithms

Tomáš Dulava

Supervisor: Ing. Petr Pošík, Ph.D.
Field of study: Open Informatics
Subfield: Computer Games and Graphics
August 2021

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Ing. Petr Pošík, Ph.D. The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, August 13, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 13. srpna 2021

Abstract

Evolutionary algorithms belong to popular optimization methods. They are able to provide satisfactory solutions even for hard tasks. But even these algorithms have their limitations. A common problem we can encounter within evolutionary algorithms is so-called premature convergence which is that the population converged to a local optimum, and no more improvements could be made. There are various methods that are designed to deal with premature convergence, and one of them is using structured populations. These structures modify the behavior of evolutionary algorithms in order to maintain diversity of the population. The goal of this thesis is to compare classical unstructured evolutionary algorithm with some structured algorithms, specifically ALPS: Age-Layered Population Structure, MAP-Elites: Multi-Dimensional Archive of Phenotypic Elites and cellular evolutionary algorithm, in order to find out if using structured populations is really beneficial.

Keywords: black-box optimization, premature convergence, evolutionary algorithm, structured population, ALPS, Age Layered population structure, MAP-Elites, Mutli-dimensional archive of phenotypic Elites, cellular model, diffusional evolutionary algorithm, TSP, Travelling Salesperson Problem

Supervisor: Ing. Petr Pošík, Ph.D.

Abstrakt

Jednou z populárních optimalizačních metod jsou evoluční algoritmy, které dokáží poměrně efektivně řešit i velmi složité úlohy. Ovšem i tyto algoritmy mají svoje limity. Jedním z běžných problémů, se kterým se můžeme u evolučních algoritmů setkat, je předčasná konvergence, kdy algoritmus konvergoval do lokálního optima a není již schopen dalšího zlepšení. Jednou z možností, jak se s tímto problémem vypořádat, je použití struktur při správě populace evolučního algoritmu. Použití struktury určitým způsobem modifikuje chování evolučního algoritmu a pomáhá tak zachování diverzity populace. Cílem této práce je porovnání klasického evolučního algoritmu a několika vybraných evolučních algoritmů využívajících strukturovanou populaci. Konkrétně jsou to algoritmy ALPS: Age-Layered Population Structure, MAP-Elites: Multi-Dimensional Archive of Phonetypic Elits a celulární evoluční algoritmus. Otázkou tedy je, zda-li použití strukturované populace opravdu pomáhá evolučnímu algoritmu nalézt lepší řešení, respektive za jakých podmínek.

Klíčová slova: black-box optimalizace, evoluční algoritmus, předčasná konvergence, strukturovaná populace, ALPS, Age Layered population structure, MAP-Elites, Mutli-dimensional archive of phenotypic Elites, celulární model, difůzní evoluční algoritmus, TSP, problém obchodního cestujicího

Překlad názvu: Struktura populace v evolučních algoritmech

Contents

Figures

Chapter 1

Introduction

In almost every domain, there is a need to optimize some problems and find good solutions. For many problems, there are designed sophisticated optimization methods. Effective for particular problems yet not always applicable on other ones. What to do when it is not clear what optimization method should be used? What to do when we even do not know what the optimization function looks like? What to do when the optimal solution changes over time or is noisy? What to do if there are too many candidate solutions for a problem, and we can not test all of them in a reasonable time? A possible answer to all questions above is to use evolutionary optimization algorithms.

Firstly, we will take a look at optimization in general. What is it? What is important to keep in mind when optimizing some problem? Then we will talk about evolutionary algorithms. What are they? How they work? Then we will discuss particular evolutionary algorithms, and finally, based on conducted experiments, we will answer the following questions: "Can be structured population beneficial for evolutionary algorithms? Under what conditions is it so?"

1.1 Optimization

Optimization is, generally speaking, the selection of the best element from a set of available alternatives. Typically, an optimization problem is defined by[1]:

- A set of available solutions from some set:
  $X \subseteq U$
- An objective function that we want to minimize[1]:
  $f : X \to \mathbb{R}$
- A set of constraints that a candidate solution must comply with
  $g_i(x_1, ..., x_n) \leqq 0, \quad i = 1, ..., m$

[1]If we want to maximize some function we can easily rewrite it as minimization because $max_{x \in X} f(x) = min_{x \in X} - f(x)$

$$f_i(x_1, ..., x_n) = 0, \quad i = 1, ..., n$$

The solution of an optimization problem is $x^* \in X$ such that $f(x^*) = \min_{x \in X} f(x)$ and $f(x^*)$ is an optimal value. The x^* is also called global minimum. Besides global optimum, there is also so-called local optimum. That is the best solution in the immediate vicinity. Global optimum is always also local minimum, but it does not hold the other way round. The difference between global and local optimum is for better idea depicted in Figure 1.1.

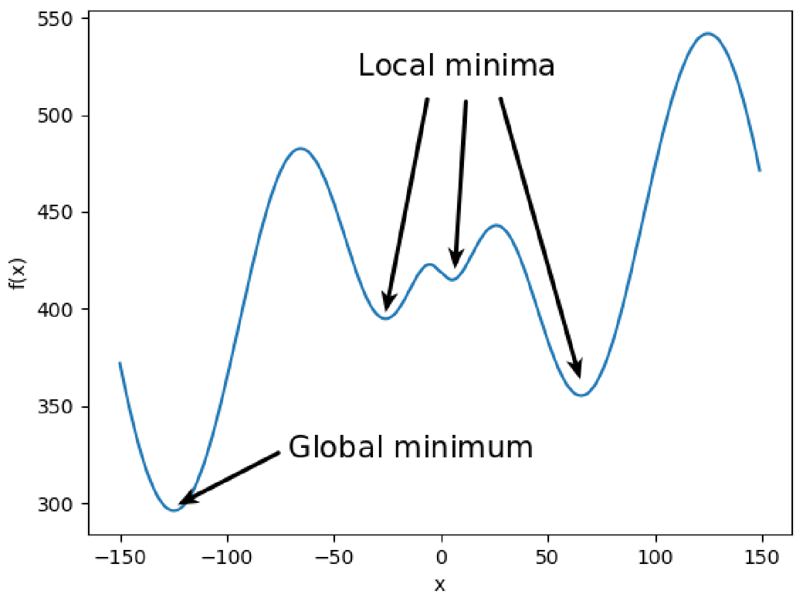


Figure 1.1: The demonstration of difference between global optimum and local optimum on the Schwefel function.

According to this image of optimization, an example of a simple optimization problem could be as follows:

Find $x \in R$ that minimizes function $f(x) = x^2$, such that $x \geqslant 0$ and $x \leqslant 10$
Solution: $x^* = 0, f(x^*) = 0$

In this particular case the solution is obvious, but that is not matter of course for all optimization problems. Some optimization problems require a complicated analytical way of solving and for many optimization problems an analytical method does not even exist.

Another obstacle we can encounter when solving optimization problems could be a lack of information of the optimization problem when we do not know what the objective function looks like. For instant, when the evaluation of a candidate solution is based on some simulation. This type of optimization is called black-box optimization.

A question arises what to do when we need to find a solution of the black-box optimization problem or optimization problem where no low-cost or analytic solution is known. Popular methods that are suitable for solving this class of problems are evolutionary algorithms.

1.2 Evolutionary algorithm

Evolutionary algorithms are popular optimization methods that are especially valued for their robustness, effectiveness and simple implementation. Evolutionary algorithms (EA) are stochastic algorithms inspired by natural evolution. It is worth taking a look at the evolution in biology for better understanding of EAs because the terminology and basic mechanisms comes from it.

1.2.1 Evolution in biology

Imagine some living organism, a cat for instant. The collection of all its observable traits is called its *phenotype*. The height of the cat, its weight, color, etc. are features that contributes to its phenotype. Then we have a *genotype* of an organism. That is the collection of all its *genes*. We can think about it like a set of information what the organism should look like. We usually do not observe the genotype of an organism directly, but we can observe its influence on its phenotype. So then we can observe correlation between presence of some gene in the genotype and the phenotype of an organism. But the phenotype of an organism is not fully destined only by its genotype but also by effect of an environment and what the organism experienced.

With this in mind there could be potentially two organisms with the same genotype but with different phenotypes as well as organisms with different genotypes but the same phenotype.

Different individuals have different phenotype, different traits and thus their successfulness in survival and reproducing also vary. How good traits an individual has reflects its *fitness* value. Simply put, the better traits an individual has, for its environment and situation, the more successful it is, and thus the better fitness it has.

When two individuals mate with each other, they reproduce offspring that inherit a combination of genes from their parents. This way an individual with unique genotype can be produced, but still based on a genotype of its parents. Besides that, an individual might be affected by mutation, when its genotype is changed.

Now there is a question, how the evolution works? Let's have a population of individuals with various genotypes and phenotypes and an environment with limited resources. Those with great fitness values will have tendency to produce more offspring than those with low fitness value, and thus, completely naturally, in the population will be more and more individuals with high fitness value.

The concept of evolution introduced here is very simplified and not absolutely correct. Biological evolution is a complicated and complex field of study, and it is not the object of this thesis to fully describe it here. Yet, the concept introduced here is well known, simple and useful for our purposes.

1.2.2 Terminology

Now it is a time to combine our basic concept of biological evolution with optimization.

- *Genotype*:
  The mere representation of a candidate solution. It could be a binary string, real value vector, tree graph, etc.
- *Phenotype*:
  The actual meaning of the genotype. For instance, genotype represented as a permutation of integers could actually mean an order of execution of some tasks.
- *Fitness*:
  Quality of the candidate solution. It is assigned based on the objective function. The better fitness, the more promising candidate solution.
- *Individual*:
  The candidate solution with its fitness value so basically genotype+fitness.

1.2.3 Mechanisms

The simplified workflow of EA looks as follows. At the beginning of the algorithm, a set of random candidate solutions is generated. Then each candidate solution is evaluated with respect to the objective function we want to optimize. This way, each candidate gets its fitness value that represents the quality of the candidate. Then an evolution of this population of candidate solutions takes place, and this evolution lasts until termination conditions are met. The terminal condition is usually number of evaluations the algorithm can make, but it also could be a time limit, detection of sufficiently good solution, etc. The evolution runs in a cycle and this cycle consists of 4 stages:

- *Selection*:
  We select parents from our population, usually individuals with great fitness.

- *Mutation and crossover*:
  By mutation, we individually modify parents to create new candidate solutions. By crossover, we combine parents to create new candidate solutions.
- *Evaluation*:
  In evaluation stage, we evaluate all newly created candidate solutions with respect to our objective function.
- *Replacement*:
  In replacement, we create new population based on the old one and newly created individuals.

When the terminal conditions are met, the algorithm returns the best individual in the population.

A simplified description of how EA works is depicted in the following pseudo-code:

Algorithm 1 Illustration of simple evolutionary algorithm

```
1: procedure SIMPLE EVOLUTIONARY ALGORITHM
2:     $population$ ← initialization of population
3:     evaluate $population$
4:     while the conditions for termination are not met do
5:         $parents$ ← select individuals from $population$
6:         $offspring$ ← make mutation and crossover of $parents$
7:         evaluate $offspring$
8:         $population$ ← make replacement with $population$ and $offspring$
9:     return the best individual of $population$
```

There are, of course, various strategies and methods for individual parts of EA such as initialization, selection, mutation, crossover and replacement. Very important is selection pressure. Either selection or replacement have to operate in favor of individuals with better fitness, otherwise the algorithm have no mechanism to generate better individuals.

Even though that evolutionary algorithms are powerful solving techniques, they have their limitations. One of the problems we can encounter is so called *premature convergence*.

1.2.4 Premature convergence

What it actually means is that the population converged to some local optimum and no more improvements could be made. The population prematurely lost its diversity and is unable to explore other parts of the fitness landscape[2], as depicted in Figure 1.2. This usually happens when the selective pressure is too big or when the population size and thus the initial diversity is too low.

There are various techniques and methods to prevent this state.

- Start with bigger population size:
  This approach could really help to prevent premature convergence, but when we need to process bigger population it also takes more time. If we use the number of evaluations, number of generation or time as terminal condition it can happen that algorithm prematurely ends without good solution. To estimate right population size is a hard task in general.

- Make the selective pressure weaker:
  With this approach we do not lose the diversity of the population so quickly, so it can to help to prevent prematurely convergence, but another problem can arise and that it *stagnation*. It basically means that the selective pressure is too weak, and that leads to ineffective search when individuals are not encouraged to explore other parts of the fitness landscape.

- Higher mutation rate:
  This way, we increase diversity of the population during the run, but if it is too big offspring unlike to their parent are made and this leads to random search like algorithm that is also ineffective. Besides that, with high mutation rate, exploring of narrow peaks of fitness landscape becomes also very hard.

- Structured population: Set population to structure usually means to introduce some restriction on selection or replacement operator, but the effect of structure can go beyond that, depending on the particular structure.

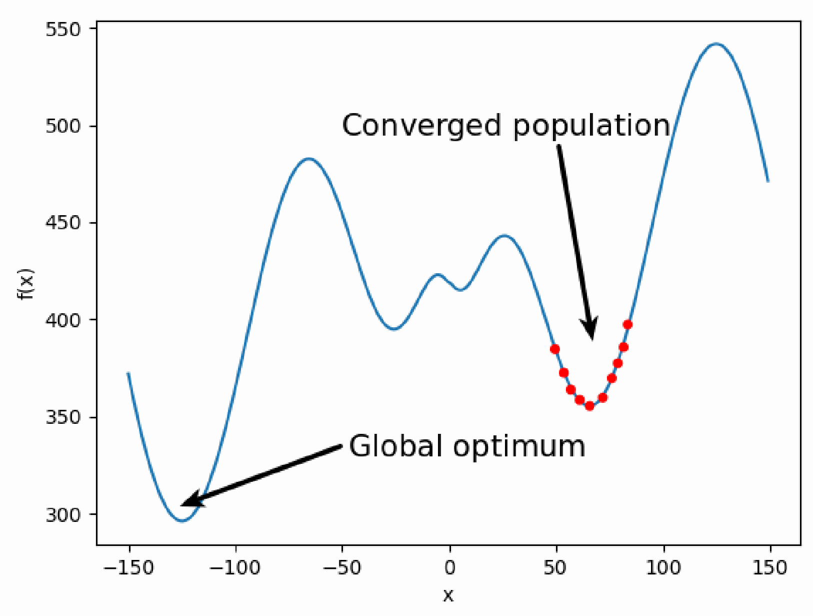


Figure 1.2: Example of premature convergence on Schwefel function. Population converged to local optimum and no more improvements could be made. The Global optimum is unreachable.

Chapter 2
Algorithms

There have been proposed many evolutionary algorithms with structured population. There is a question whether using some structure can really improve the ability of evolutionary algorithm to find better solution. Why it could be so?

When classical evolutionary algorithm without structured population is used, there is not any restriction for breeding. The selection operator chooses from the whole population of individuals, and the same holds for the replacement operator. Using structured population basically means to restrain somehow these operators.

Now, take a look at algorithms we are going to compare.

2.1 Simple evolutionary algorithm (EA)

Just a simple evolutionary algorithm with unstructured population. Its pseudo-code is depicted in Alg. 2.

2.2 ALPS: Age-Layered Population Structure

Age-Layered Population Structure[3] is an evolutionary algorithm designed to prevent premature convergence. There are many other approaches to face this problem, but they are not usually suitable for complicated representations, and they are limited to discovering solutions that are within the basin of attraction of the initial population. ALPS deals with this problem by introducing newly generated individuals in regular intervals and incorporating them into the population. The population is structured into layers based on the age of individuals. Pseudo-code of ALPS is depicted in Alg. 3.

2.2.1 Age

ALPS introduces a new measure of age. Each entity in ALPS has an age that measures how long its genetic material has been evolving. Thus, a

Algorithm 2 Pseudo-code of simple EA

```
 1: procedure SIMPLE EA
 2:     population ← initialize population
 3:     while terminal conditions are not met do
 4:         population ← make generation from population
 5:     return best individual of population

 1: procedure MAKE GENERATION FROM population
 2: selection:
 3:     elites ← find k best from population
 4:     individuals ← select n-k from population
 5: crossover:
 6:     for ind1, ind2 in individuals do
 7:         if random < crossover probability then
 8:             ind1, ind2 ← breed ind1 with ind2
 9: mutation:
10:     for ind in individuals do
11:         if random < mutation probability then
12:             ind ← mutate ind
13: return:
14:     return elites with individuals
```

newly generated individual has age 0. An individual created by mutation or crossover has age 1 plus the age of the older parent.

2.2.2 Layers

Based on the age, ALPS divides population into layers. Each layer has a maximum capacity of individuals and a maximum age that an individual can have in the layer.

The initial layer has a special function. Individuals in this layer are replaced by newly generated individuals in regular intervals, and thus this layer is the source of new genetic material.

Every individual can breed only with individuals from the same layer or from the previous layer. Thus, younger individuals have opportunity to find their own optimum and are not dominated by older individuals. This should ensure effective exploration of fitness landscape by newly generated individuals.

If an individual became too old for its current layer, it is compared against every individual in the older layer. If it is better than least one of them it replaces the worst one, otherwise it is discarded. This way, every individual is guaranteed to stay in the population forever only if it is the global optimum, otherwise it will be eventually replaced by a better individual.

If an individual became too old for its current layer, it is compared against the worst individual in the older layer. If it is better, it replaces it, otherwise it is discarded.

2.2.3 Age-gap

Age-gap is an important parameter in the settings of ALPS. It determines the interval in which new individuals are generated, as well as the age limits for layers in the structure.

2.2.4 Domain

ALPS is not restricted to any particular representation of individuals and thus could be widely used.

Algorithm 3 Pseudo-code of ALPS

```
1: procedure ALPS
2:     structure ← initialize structure
3:     while terminal conditions are not met do
4:         for layer in structure do
5:             process layer
6:     return best individual of structure
1: procedure PROCESS layer
2:     if layer is active then
3:         if layer is initial layer then
4:             population of layer ← initialize population
5:         for gen ← 1, 2,...,AGE-GAP do
6:             population of layer ← make generation from populatin of layer
7:             move old individuals from current layer to older layer
8:     else if layer is ready to be activated then
9:         activate layer
```

2.3 MAP-Elites: Multi-dimensional Archive of Phenotypic Elites

Evolutionary algorithms typically return one fittest solution. MAP-Elites[5] belong to the family of algorithms called Quality-Diversity[7] algorithms, and this type of algorithms have a bit different ambitions. They return a set of phenotypically different individuals, yet with good quality. The main goal of Quality-Diversity optimization is to illuminate the search space, and find out how high-performing solutions are distributed throughout the search space. In result, it provides us with a holistic view over the problem.

2.3.1 Feature space

In traditional evolutionary algorithm we work with individuals only in the genotypic space. But in Quality-Diversity optimization we work also with phenotypic space. This space forms dimensions of interest in some features of individuals. Although the genotypic space can be very large or also infinite, the phenotypic space is usually quite small in terms of number of dimensions. For example, in optimization of robot movement the fitness value could be the speed of the robot and the features could be its weight and height even though the robot itself can be described by a very long chromosome. Then, we are interested in the fittest robot throughout various combinations of height and weight.

The feature space is discretized thus it can be described as a multidimensional grid. A particular position in the feature space is usually called as cell.

The problem is that we can not search explicitly in the feature space, but we have to search in the genotype space. Therefore, we need some mapping from genotype space to the feature space, and this mapping function is called a feature descriptor.

2.3.2 Feature descriptor

The feature descriptor takes genotype as the input and return some position in the feature space as an output. The mapping may be direct from the genotype, but also it could be a result of some more complicated operation that is not easily determined by the genotype. There is no guarantee that each cell in the feature space can be filled. It is also possible that various different genotypes are mapped into the same cell. In other words, the genotype-phenotype function is not guaranteed to be injective nor surjective[8].

2.3.3 Workflow

The workflow of the algorithm looks as follows. First a random population of individuals is generated, then based on the feature descriptor their positions in the feature space are found. If the position is empty or an individual with worse fitness is present here, they are placed here. At the same time, an individual is present in a cell unless a better individual with the same feature values is found, therefore the feature space really works as an archive of phenotypic elites. The pseudo-code of the algorithm is depicted in Alg. 4.

2.3.4 Properties

Because the MAP-Elites belong to Quality-Diversity algorithms, it returns not only the fittest solution but the whole set of high-performing solutions throughout the feature space. This way it illuminates the relation between performance and fitness of individuals, and gives holistic view of the problem.

Even though the MAP-Elites is not focused solely on the fittest solution, but also is focused on the high phenotypic diversity of the solutions it may be surprising but the MAP-Elites finds even better solution than classical evolutionary algorithm on some problems, especially on the very deceptive or complex ones[5][6].

2.3.5 Domain

MAP-Elites is primary designed for problems, where we are interested in some phenotypic qualities of individuals, and we want to know about relations between these qualities and the fitness value of the individual. Nonetheless, the MAP-Elites could be widely used on any type of problem for which one can design reasonable feature descriptors.

Algorithm 4 Pseudo-code of MAP-Elites

```
 1: procedure MAP-ELITES
 2:     structure ← initialize structure
 3:     population ← initialize population
 4:     for individual in population do
 5:         add individual to structure
 6:     while terminal conditions are not met do
 7:         parents ← select parents from structure
 8:         offspring ← make offspring from parents
 9:         for individual in offspring do
10:             add individual to structure
11:     return best individual of structure
 1: procedure ADD individual TO structure
 2:     coordinates ← feature descriptor applied on individual
 3:     if structure is empty on the cordinates then
 4:         individual is placed on the coordinates of the structure
 5:     else
 6:         rival ← entity on the coordinates of the structure
 7:         if fitness of individual is better than fitness of rival then
 8:             individual replaces rival
```

2.4 Cellular model

Cellular model[4] is a well known type of evolutionary algorithm with structured population. Individuals are spatially distributed, and an individual can interact only with individuals that are sufficiently close to him. Because of this, even superior genes are spread slowly and gradually diffuse throughout the population. In result, the algorithm has more explorative behavior, and it is more resistant against getting stuck in a local optimum.

The individuals are spatially distributed, and an individual can interact only with a subset of the population. This behavior could be demonstrated as a graph, where nodes represent individuals and edges between individuals represent whether individuals can "see" each other. The neighborhood of an individual is the set of other individuals it is connected to.

We can encounter various types of graphs. For example, in Figure 2.1 we can see that all individuals are connected, and thus all individuals can interact with each other. This is a trivial case with no restriction in interaction between individuals, and actually this is the case for simple evolutionary algorithm. A more interesting case in shown in Figure 2.2, where some of the edges are missing, and thus the algorithm will converge more slowly.

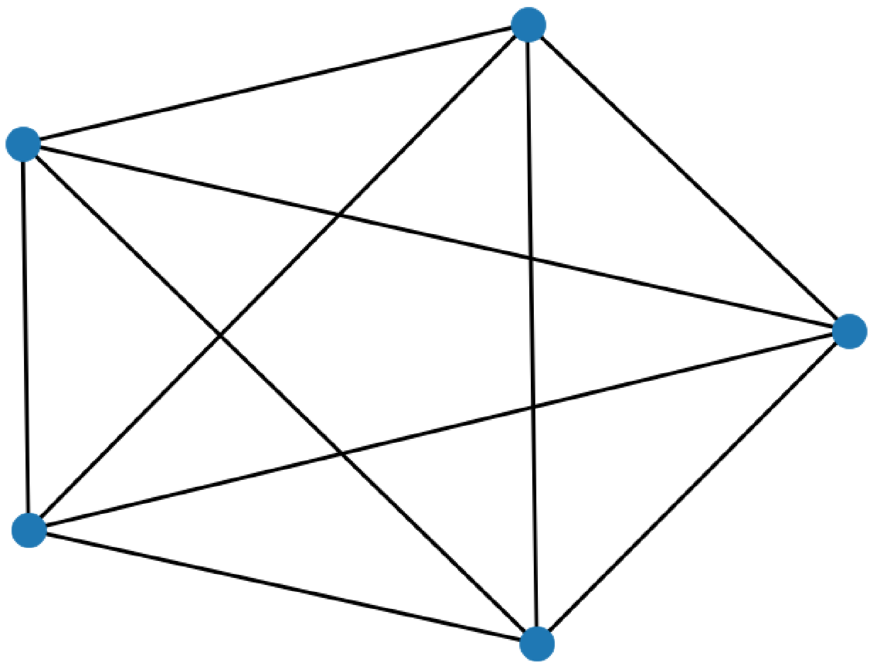


Figure 2.1: Completely connected graph with 5 nodes

To represent structured population arbitrary graph could be used, but for cellular model lattice graph is usually used.

2.4.1 Lattice

The lattice graph, also called a mesh graph or grid graph, is a regular and connected graph. It basically means that each individual has a neighborhood of the same size and between every two nodes exists a series of edges the nodes are connected with. An example of 1-dimensional lattice graph is depicted in Figure 2.3 and is called ring. Another example is shown as Figure 2.4. It is a 2-dimensional lattice graph called torus with rectangular spaces in the graph.

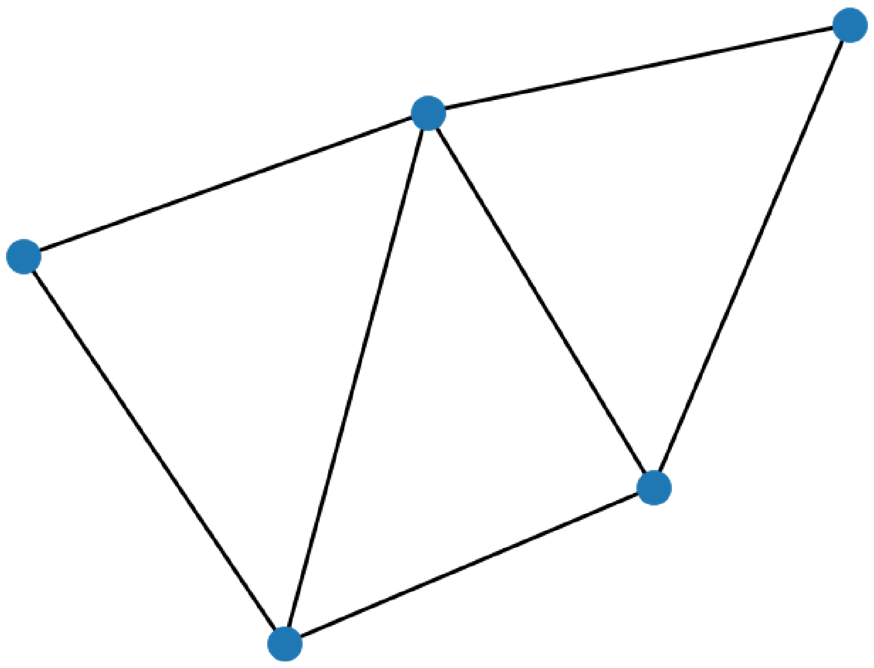


Figure 2.2: Incomplete graph with 5 nodes

We can also think about the torus as a grid wrapped around on itself. In some cases this point of view is quite useful. Although lattices with arbitrary number of dimensions could be used, typically 1-dimensional or 2-dimensional lattices are used.

The graph insight view here is very simplified, but it is sufficient for our needs. For more information about graphs, see [4]

2.4.2 Neighborhood

When an individual is about to be processed, only its neighborhood is taken into account. There are various types of neighborhood but among the most popular ones is so-called von Neumann neighborhood.

Let't take a look on the von Neumann neighborhood on the rectangular grid. Neighborhood is defined by its range r which is typically 1. To the neighborhood of some node belong all nodes of the graph that are achievable by maximally r edges from the node. Examples are provided in Figure 2.5

2.4.3 Update

Each individual in the population is processed independently. The lattice can be processed in two ways.

- *Asynchronous:*
  Individuals are processed sequentially.

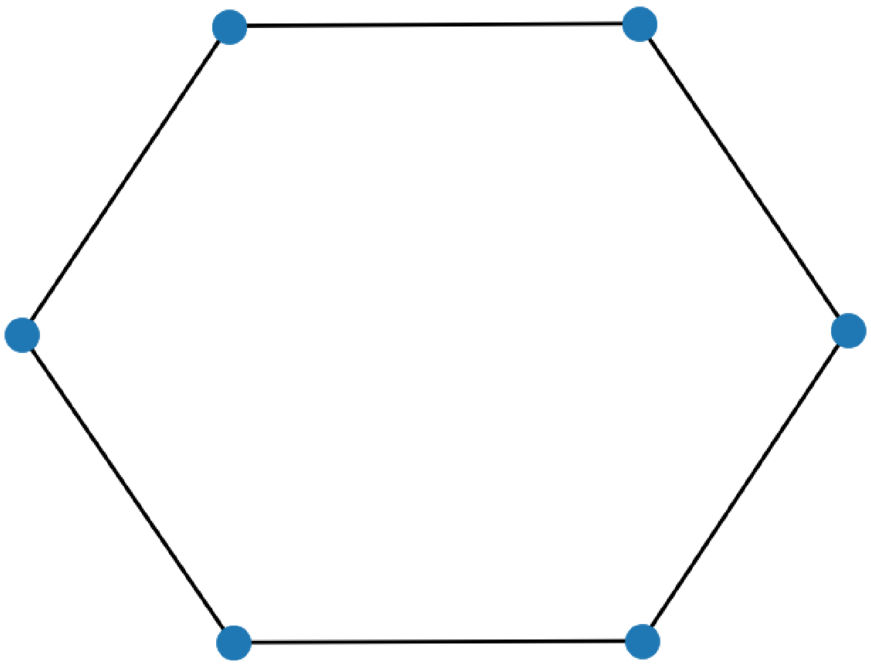


Figure 2.3: A 1-dimensional lattice graph called ring

- *Synchronous:*
  All individuals are processed in parallel.

When an individual is processed, its neighborhood is found first. This is the subset of the population that we take into account. From the neighborhood we select parents and create offspring as usual. If an offspring with better fitness than the individual is produced then the better offspring replaces the individual, otherwise the individual remains unchanged and the process of another individual takes place. Pseudo-code of cellular model is shown in the Alg. 5.

2.4.4 Domain

Cellular model has no restriction on particular representation of individuals neither on any type of problems thus could be widely used.

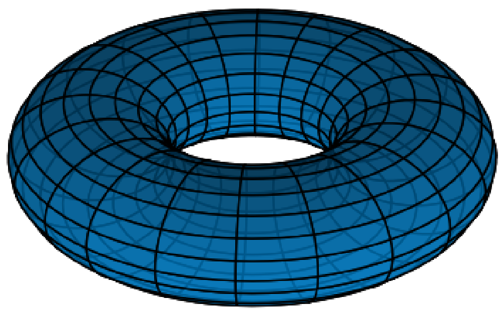
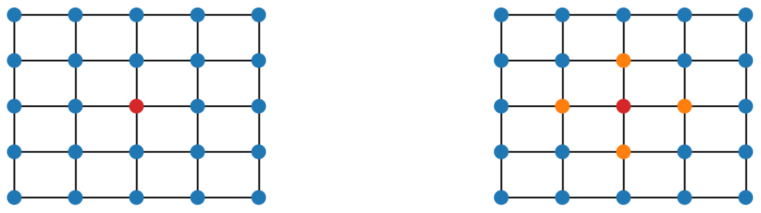
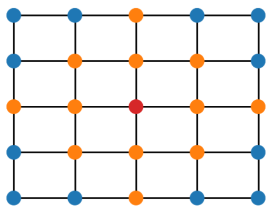


Figure 2.4: A 2-dimensional lattice graph called torus



(a) With radius 0 only the node belongs to its neighborhood

(b) With radius 1 there are totally 5 nodes in the neighborhood



(c) With radius 2 there are 13 nodes in the neighborhood

Figure 2.5: Examples of von Neumann neighborhood on the grid graph for different radii. The target node is red and by orange are colored other nodes that belong to its neighborhood

Algorithm 5 Pseudo-code of Cellular model

```
1: procedure CELLULAR MODEL
2:     $structure$ ← initialize $structure$ with random population
3:     while terminal conditions are not met do
4:         for $individual$ in $structure$ do
5:             $neighbors$ ← find neighborhood of $individual$
6:             $parents$ ← select parents from $neighbors$
7:             $ofspring$ ← make offspring from $parents$
8:             $rival$ ← select best from $offspring$
9:             if fitness of $rival$ is better than fitness of $individual$ then
10:                $rival$ replaces $individual$
11:    return best individual of $structure$
```

Chapter 3

Testing problems

To compare our algorithms and thus analyze effectivity of structured populations I tested algorithms on several continuous and discrete problems. All instances of discrete problems tested belong to Travel Salesperson Problem domain.

3.1 Continuous problems

In continuous problems we distinguish unimodal problems and multimodal problems. The difference between them is that an unimodal problem has only one local optimum (which is also a global optimum) whereas a multimodal problem has more local optima. Multimodal problems are generally harder to optimize, because there is a great chance to get stuck in a local optimum. In our set of testing functions are presented instances of unimodal functions as well as multimodal functions.

In order to test algorithms on continuous problems, the Comparing Continuous Optimizers platform (COCO) was used [12]. There are 24 functions in the benchmark of COCO[13] and for each function there are 15 instances of it.

All functions were tested in the interval [-5, 5] in every dimension and for the following number of dimensions: 2, 5, 10, 40.

3.2 Traveling Salesperson Problem (TSP)

This is a famous and intensively studied optimization problem that is defined as follows:
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"[9]. A simple example of TSP problem is depicted in Figure 3.1.

I tested algorithms on several instances of TSP from TSPLIB[14]. Here I present description of problems[15][16] as well as fitness of optimal solutions[17].

- gr17:

- Number of cities: 17
- Optimal value: 2085

- gr21:
  - Number of cities: 21
  - Optimal value: 2707
- gr24:
  - Number of cities: 24
  - Optimal value: 1272
- gr48:
  - Number of cities: 48
  - Optimal value: 5046

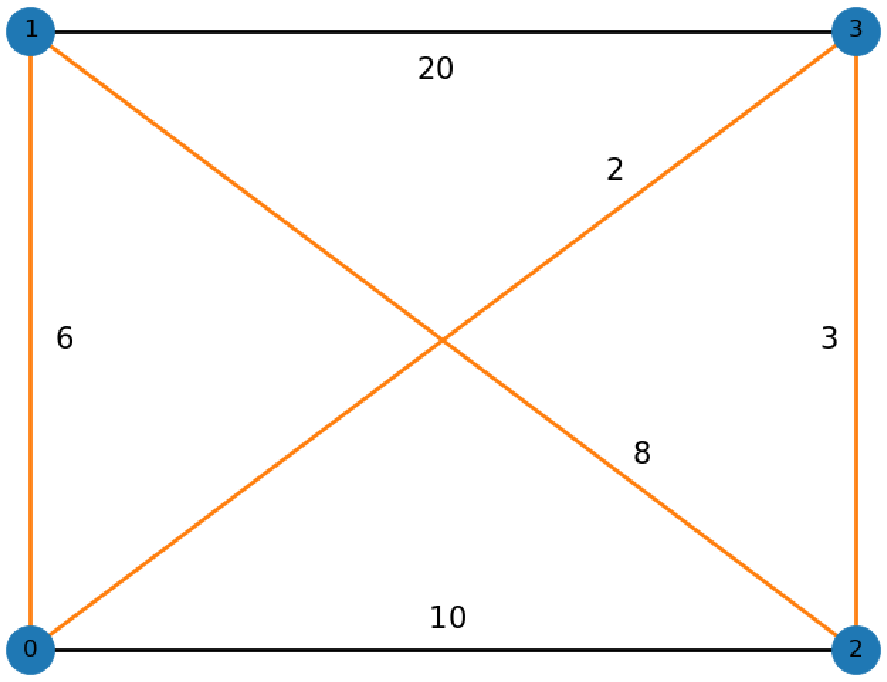


Figure 3.1: A simple example of TSP problem. By orange color is depicted the optimal solution.

Chapter 4

Configuration

There are many parameters to set in evolutionary algorithms. Some of them have greater impact on the effectivity of the algorithm then others, but all of them have to be set to some reasonable value to make the algorithm effective. The problem is that there is no one universal configuration that is best for all algorithms and problems. The optimal configuration for particular algorithm and problem varies and especially for black-box optimization it is very difficult to set some parameters right. Moreover, the more sophisticated algorithm we have, the more parameters it usually has and the higher the risk of setting some parameters suboptimally with negative effect on the algorithm[1].

Although all parameters matters, some of them are more important than others. An essential parameter is population size. If population is to small the algorithm can prematurely converge as already mentioned in the section 1.2.4 about premature convergence and if population is too big it might have not enough time to properly explore the search space and to find optimal solution.

Therefore, population size is the only parameter that is not set, and I conducted all experiments for following population sizes:

- 20
- 50
- 100

We will see how the population size really affects the effectivity of the algorithms, espeially how different algorithms are effective in dealing with small population size and thus low initial diversity.

Setting all parameters to optimal value would be very time demanding, and it is not quite a goal of this thesis. Therefore, other parameters were set to reasonable values or values based on simple experiments that are not discussed.

[1]There is a way to deal with bad configuration of parameters in evolutionary algorithms. Besides evolution of individuals, we can also have evolution of configurations. This idea opens the door to many interesting possibilities that are not concerned in this thesis though.

4.1 Shared configuration

To make the algorithms comparable and to see how the structure of population affects their performance I set them the same operators for mutation and crossover, function for generation of new individuals and the number of evaluations is also the same.

- Number of evaluations: 10,000

4.1.1 Continuous problems

- Generation of individuals
  The function creates a new individual with a random value in the interval of the problem [-5, 5] on each position of the chromosome.
- Mutation
  - Function: Gaussian mutation
  - Probability: 0.2
  - Mean: 0
  - Standard deviation: 0.1
  - Independent probability for each attribute to be mutated: 0.1

  Distribution of this function is depicted in the Figure 4.1
- Crossover
  - Function: Arithmetic crossover
  - Probability: 0.5

  With this crossover new individual is created as weighted average of parents gene-wise. So each gene of the new individual is created independently as $g_{new} = r \cdot g_1 + (1 - r) \cdot g_2$, where g1, g2 are genes of parents on the same position of the chromosome and $r \in (0, 1)$. Arithmetic crossover is depicted in Figure 4.2

4.1.2 Traveling salesperson problem

- Generation of individuals
  The function generates a random permutation of length n, where n is a dimension of particular instance.
- Mutation
  - Function: Twors mutation[10]
  - Probability: 0.2

  Swap the position of two randomly chosen genes.

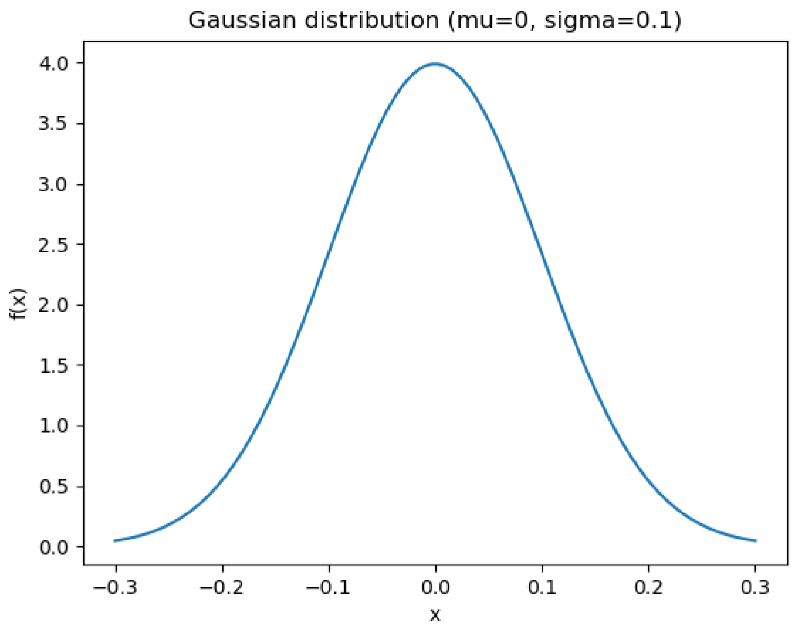


Figure 4.1: Gaussian function with μ=0 and σ=0.1. The $f(x)$ is not with the probability in a good ratio.

- Crossover
  - Function: Partially matched crossover[11]
  - Probability: 0.5

  With this crossover new individual is formed from two parts. The first one is copied from a parent. The second part is created based on the second parent so that it is as similar to it as possible.

4.2 Simple evolutionary algorithm and ALPS

These algorithms have some parameters shared because after all, the ALPS behaves as EA within a layer.

- Selection
  - Function: Tournament selection
  - Tournament size: 3
- Elitism
  - Number of elites: 3
- Replacement
  - Function: Generational

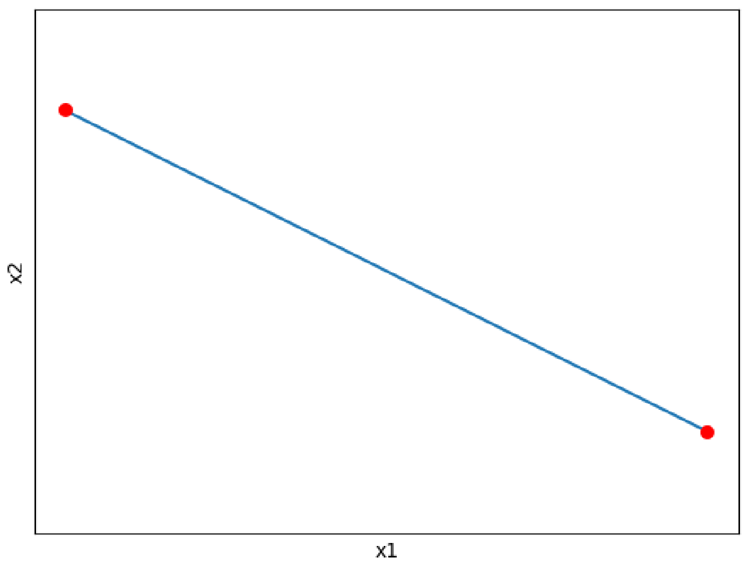


Figure 4.2: Arithmetic crossover in two-dimensional space. Red dots represents parents and all their possible offspring are represented by blue line (red dots including)

Besides that, ALPS have some other parameters related to its structure.

- Number of layers: 5
- Age gap: 20
- Max age for a layer: polynomial function was used, thus age limits were
    - layer 1 : 20
    - layer 2 : 40
    - layer 3 : 180
    - layer 4 : 320
    - layer 5 : no limit

4.3 MAP-Elites

In MAP-Elites we have to discuss two parameters that are not covered yet. The selection operator and the feature descriptor.

4.3.1 Selection

The most easy way to select parents from the population is to select them randomly. Some of the more sophisticated selection operator as tournament selection are not very suitable for MAP-Elites because they do not encourage the algorithm to explore less promising parts of the fitness landscape. Therefore curious selection was proposed.

Every individual has also so-called curious value. At the beginning of the algorithm, all individuals has the curious value equal to 0. Individuals are selected proportionally to their curious value. If selected individuals created offspring that survived and was added to the structure, their curious value increases, otherwise decreases. This way, the attention is primary put into individuals that proved themselves in creating good offspring.

4.3.2 Feature descriptor for continuous problems

What to use as a feature descriptor in functions we know nothing about? It is obvious that we can not fulfill the primary intention of feature descriptor to observe relations between fitness of a solution and its features. We have to come up with a universal feature descriptor applicable on every continuous problem that can map different solutions into different cells in feature space and thus provide us with diversity of the population. One possible solution looks as follows. Let's assume that the feature space has m dimensions and an individual has a chromosome of length n. Then the value of a solution in the first dimension of feature space is the sum of the first $n//m$ genes, where "//" is an integer division. Then the value of the second dimension of the feature space is equal to the sum of the next $n//m$ genes etc. Visual example o this feature descriptor is provided in the Figure 4.3. For additional note, see section B.1

4.3.3 Feature descriptor for travelling salesperson problem

What to use as a feature descriptor for TSP? One of the ideas that might come to mind is to measure the similarity of an individual with a solution based on the minimal spanning tree of the graph that describes particular TSP problem. But there is a problem with this approach. The population size can be much larger than the length of the path, and thus we can not create feature space of required size. How to deal with this obstacle? Well, a possible solution is to generate a whole bunch of paths and then choose the two least similar ones. Then the feature space would be two-dimensional. And this is the way I chose. For additional note, see section B.2

4.4 Cellular model

In conducted experiments, a one-dimensional lattice graph (ring) was used. As a neighborhood function, von Neumann neighborhood with range 1 was

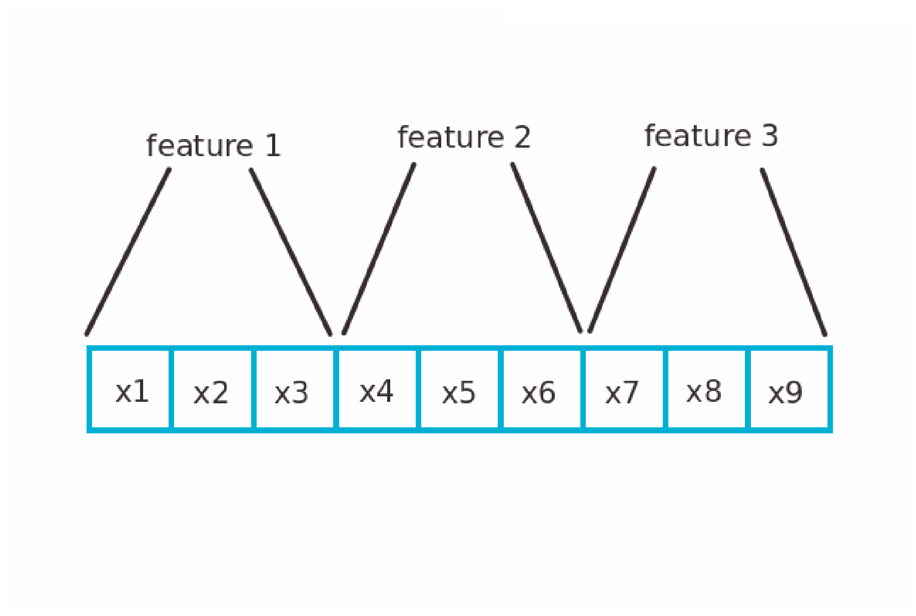


Figure 4.3: An example of feature descriptor used in continuous problems. In this particular example, the chromosome of the individual has length 9 and the feature space has 3 dimensions.

used. This function was introduced in the Sec. 2.4.2. Individuals were processed asynchronously in a random order every time.

Chapter 5

Experiments

In this chapter are present results of conducted experiments. All algorithms were introduced in chapter 2, their configuration in chapter 4 and testing problems in chapter 3. In this chapter, the following abbreviations are used:

- EA: simple evolutionary algorithm
- ALPS: Age Layered Population Structure
- CELL: cellular model
- MAP: MAP-Elites

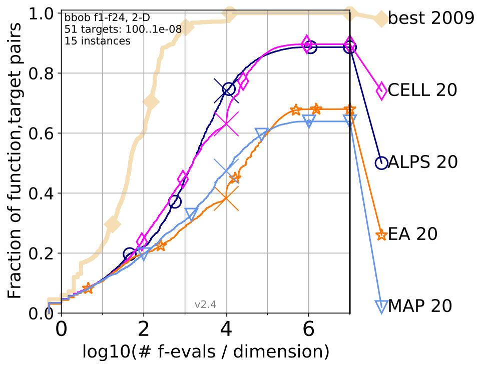
5.1 Continuous problems

Comparison of algorithms on continuous problems is depicted in graphs, where the higher value algorithm reached the better. Besides our algorithms, there is also an algorithm called "best 2009". That is the best algorithms tested on COCO to 2009.

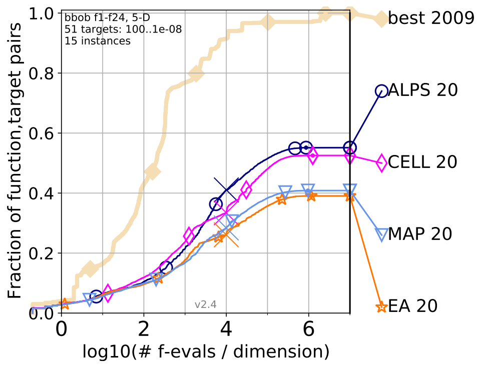
For population size 20 as we can see in Figure 5.1 ALPS and cellular model clearly outperform MAP-Elites and simple EA on 2 and 5-dimensional problems. On problems with 10 dimensions the difference in performance is almost negligible, and for 40-dimensions it is minimal. An interesting observation is that although simple EA is significantly worse than cellular model and ALPS for low dimensions, it is actually a bit better for high dimensions. Possible explanations could be that 40-dimension problems are really hard and algorithms have not enough evaluations to demonstrate advantages of structured populations. Another observation is that MAP-Elites is almost as poor in performance as simple EA. It is probably caused by the chosen feature descriptor.

The results are very similar for population size 50 as we can see in Figure 5.2. Although simple EA has significantly better performance and MAP-Elites did a bit worse. Another observation is that for 40 dimensions the ALPS stays behind other algorithms. This trend continues for population size 100 as depicted in Figure 5.3.

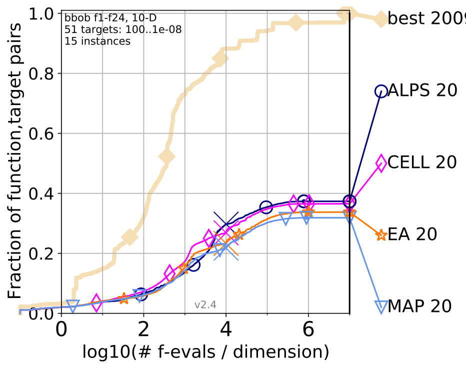
For better idea how population size influences performance of algorithms, graphs for individual algorithms throughout population sizes were plotted. Figure 5.4 for EA, Figure 5.5 for ALPS, Figure 5.6 for cellular model and Figure 5.7 for MAP-Elites.



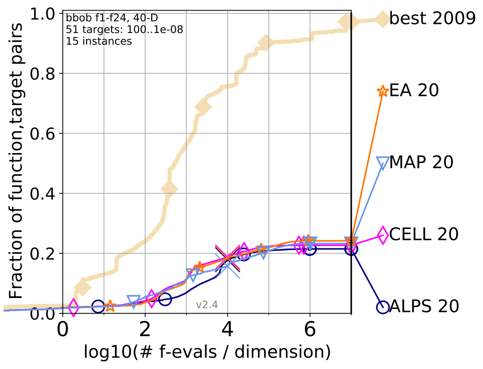
(a) For 2-dimensions



(b) For 5-dimensions



(c) For 10-dimensions



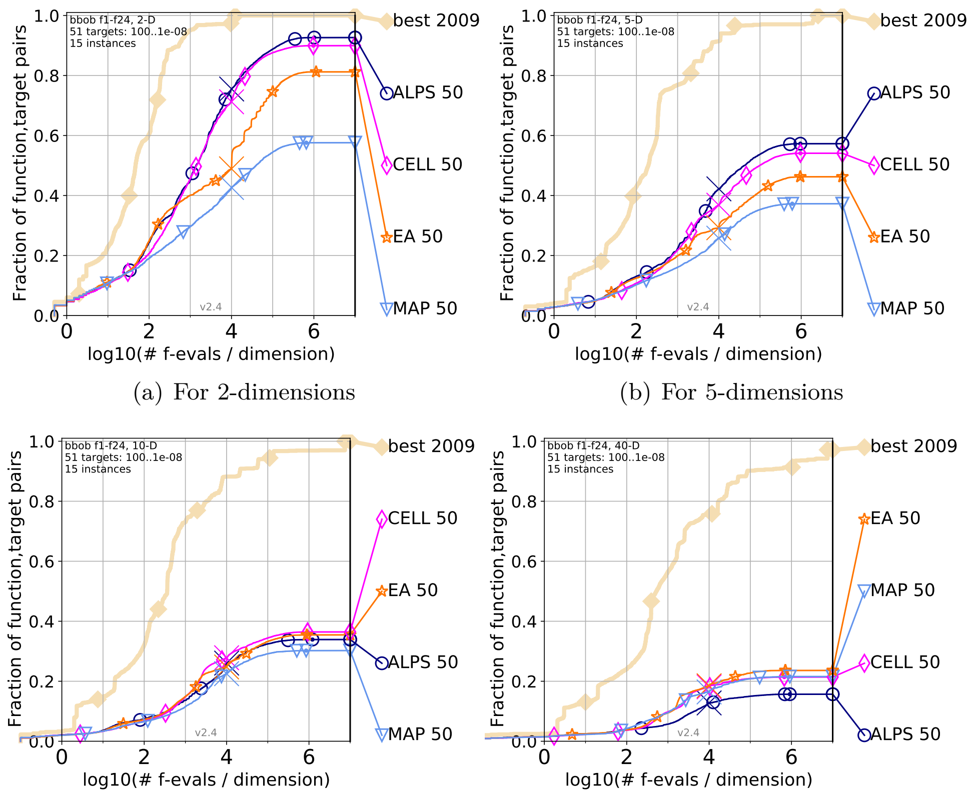
(d) For 40-dimensions

Figure 5.1: Comparison of all algorithms for population size 20 and throughout different dimensions

5.2 Traveling salesperson problem

As we can see in Figure 5.8, Figure 5.9 and Figure 5.10 the EA has bad performance on small instances of TSP problems, but surprisingly it was best on the biggest instance. Why is it so? If we look closely on the run of EA on smaller instances, we can notice that EA converges pretty quickly but then reach a local minimum and is unable to make any noticeable improvement. In gr48 are all algorithms at the end of run still in progress and none of them reached any local minima, so probably EA is not best, but our algorithms just was not provided with enough evaluations.

ALPS in general did pretty well on small instances throughout all population sizes, but was not quite good on gr48. The reason is probably lack of



(a) For 2-dimensions (b) For 5-dimensions

(c) For 10-dimensions (d) For 40-dimensions

Figure 5.2: Comparison of all algorithms for population size 50 and throughout different dimensions

evaluations.

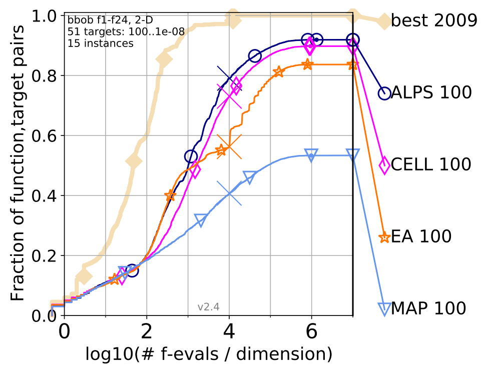
The performance of cellular model and MAP-Elites varies depending on instance of problem, and there is no simple observable trend in their performance.

It seems that population size has only small effect on the performance of algorithms. A possible reason might be that there are many candidate solutions and also many global optima [1] thus the distance between newly generated individual and global optima tends to be roughly the same every time.

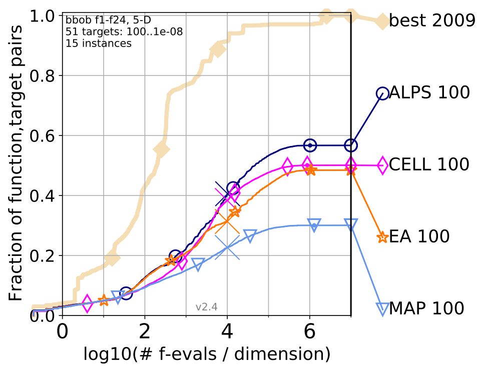
5.3 Overall results

Based on conducted experiments, the structured algorithms as ALPS and cellular model proved themselves to effectively deal with small population

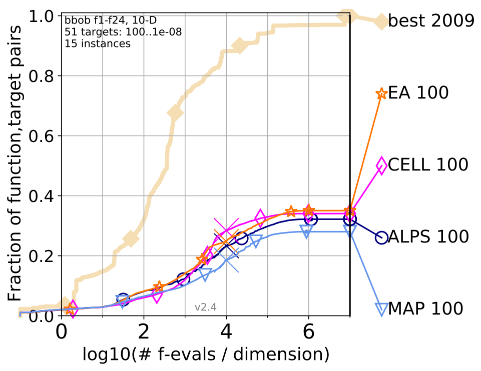
[1]The genotype of an individual is a permutation, thus there are $2 \cdot n$ genotypes representing each optimal phenotype



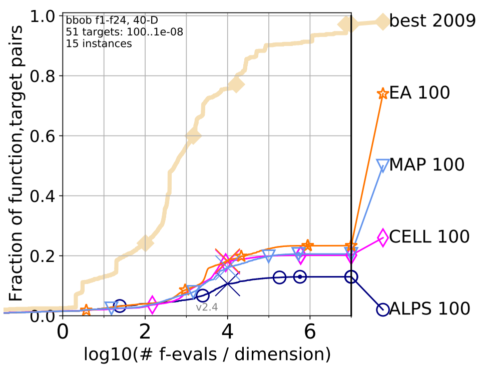
(a) For 2-dimensions



(b) For 5-dimensions



(c) For 10-dimensions

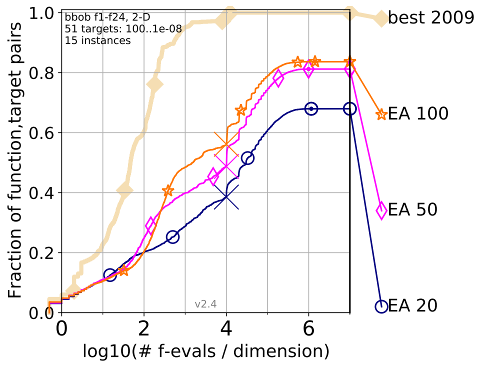


(d) For 40-dimensions

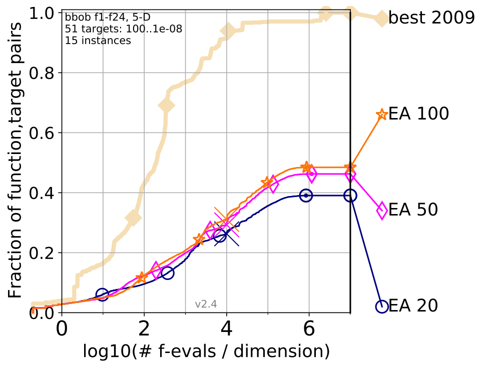
Figure 5.3: Comparison of all algorithms for population size 100 and throughout different dimensions

sizes on continuous problems. Simple evolutionary algorithm had worse performance than these two algorithms on low dimensional problems, but in higher dimensions the differences were almost negligible, actually the EA did a bit better. MAP-Elites had the worst performance of all algorithms, but this algorithm was designed for a different type of problems, plus the used feature descriptor was not meaningful.

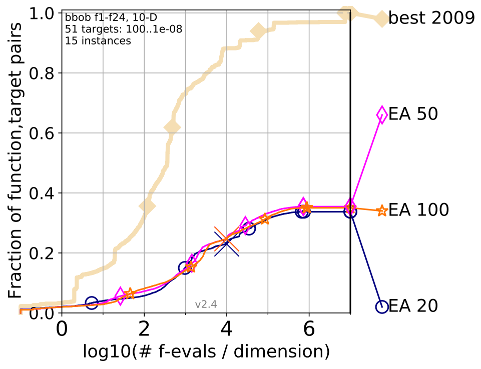
On TSP problems, structured algorithms were almost every time better than simple EA, thus proved that structured population can be really beneficial. However, EA was better on TSP instances with lots of dimension, but that could be because of insufficient number of evaluations. Also, it was observed that population size is not as important parameter as in continuous problems, at least for our representation and number of evaluations.



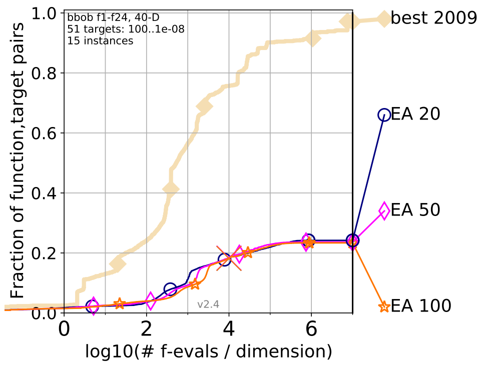
(a) For 2-dimensions



(b) For 5-dimensions

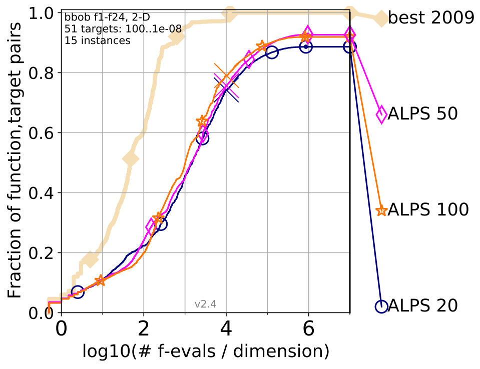


(c) For 10-dimensions

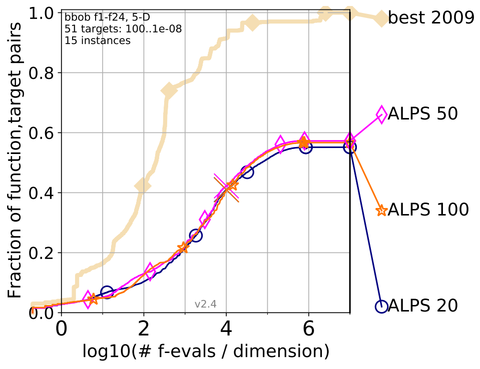


(d) For 40-dimensions

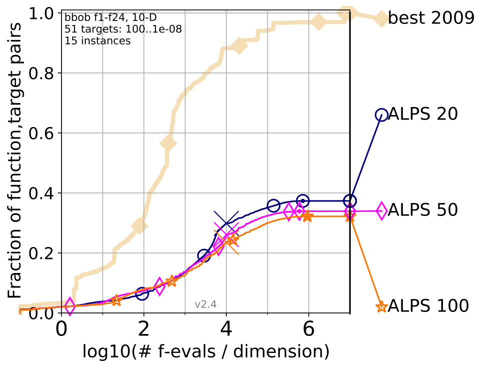
Figure 5.4: Comparison of simple evolutionary algorithms with different population sizes throughout different dimensions. The graph thus shows the ability of EA to deal with low diveristy.



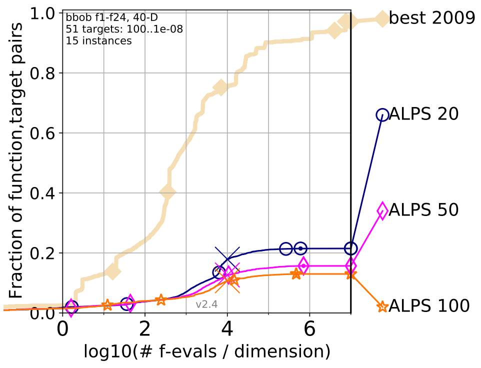
(a) For 2-dimensions



(b) For 5-dimensions

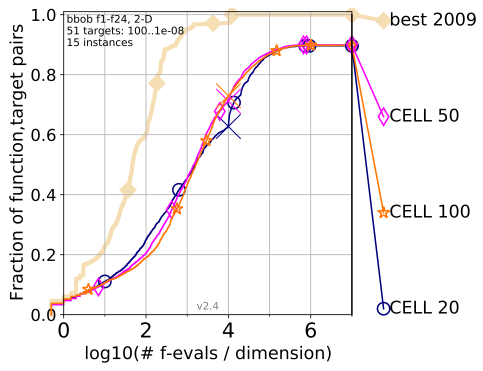


(c) For 10-dimensions

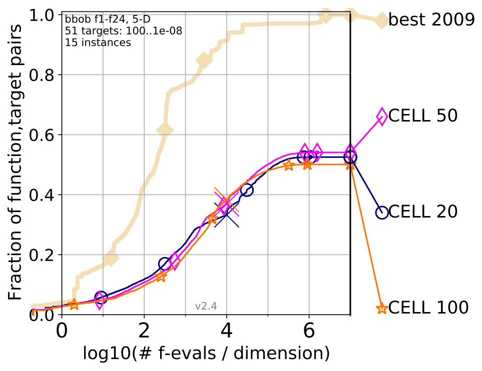


(d) For 40-dimensions

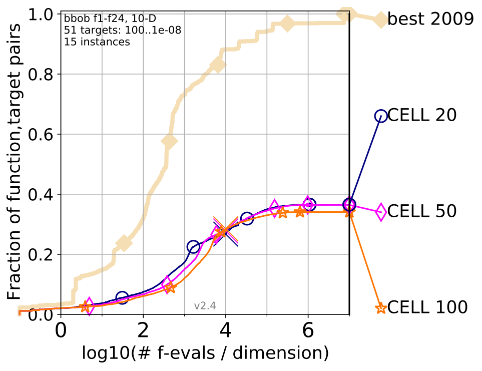
Figure 5.5: Comparison of ALPS with different population sizes throughout different dimensions. The graph thus shows the ability of ALPS to deal with low diversity.



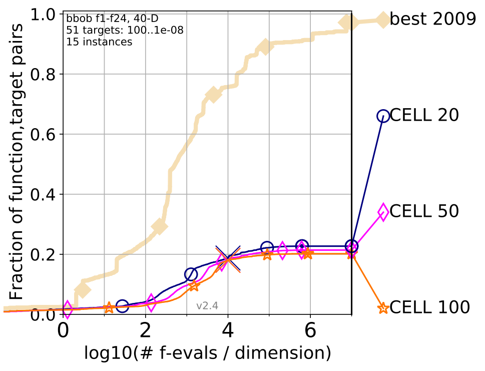
(a) For 2-dimensions



(b) For 5-dimensions

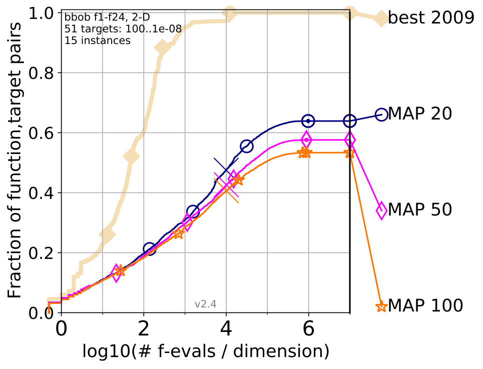


(c) For 10-dimensions

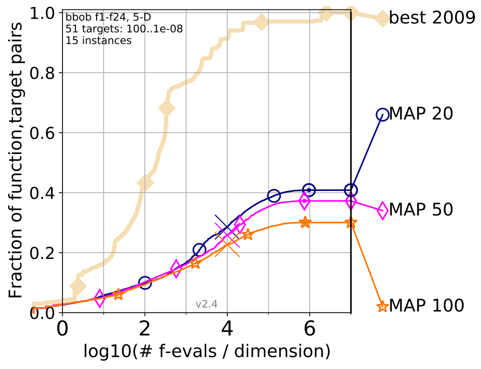


(d) For 40-dimensions

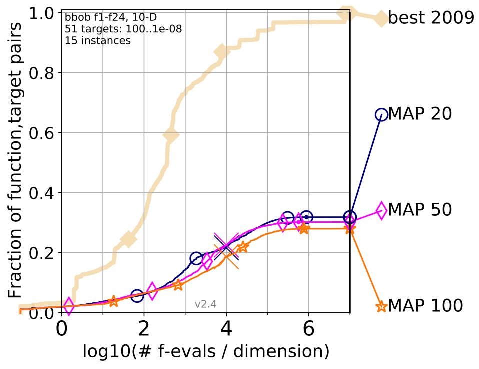
Figure 5.6: Comparison of cellular model with different population sizes throughout different dimensions. The graph thus shows the ability of the cellular model to deal with low diversity.



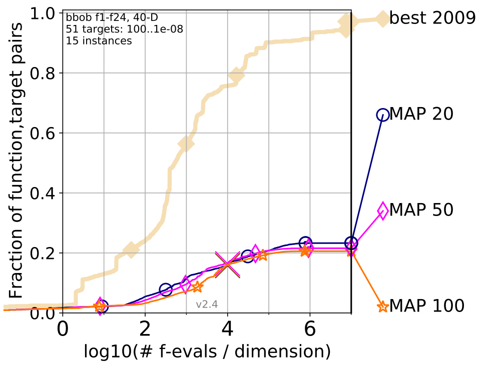
(a) For 2-dimensions



(b) For 5-dimensions

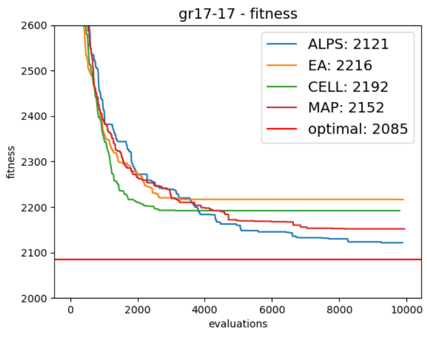


(c) For 10-dimensions

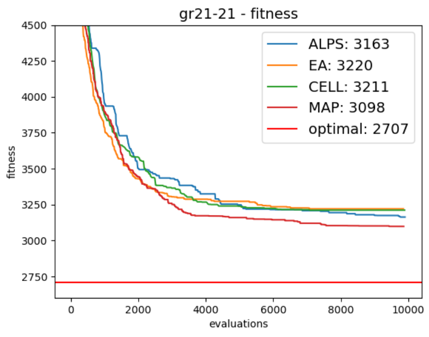


(d) For 40-dimensions

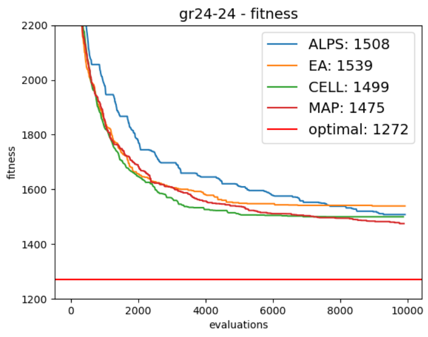
Figure 5.7: Comparison of MAP-Elites with different population sizes throughout different dimensions. The graph thus shows the ability of MAP-Elites to deal with low diversity.



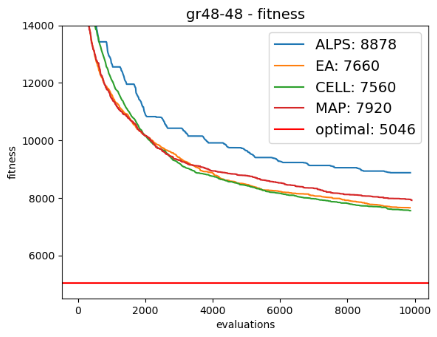
(a) gr17



(b) gr21

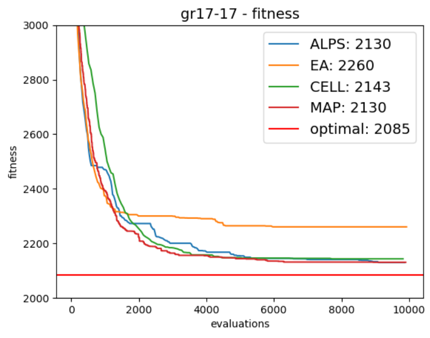


(c) gr24

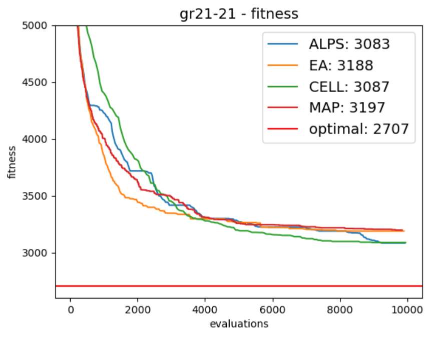


(d) gr48

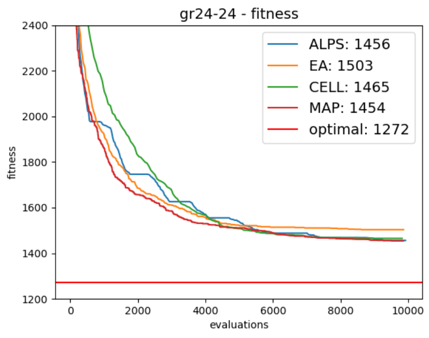
Figure 5.8: Comparison of algorithms on TSP problems for population size 20



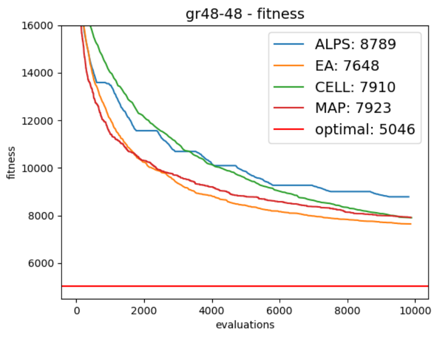
(a) gr17



(b) gr21

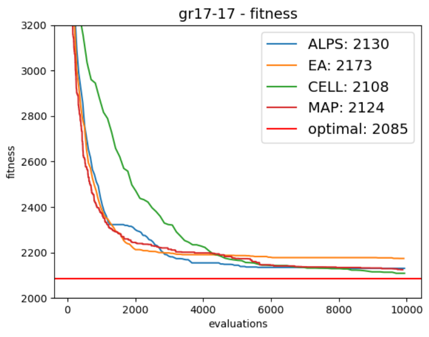


(c) gr24

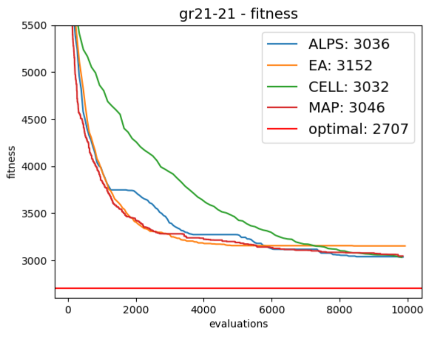


(d) gr48

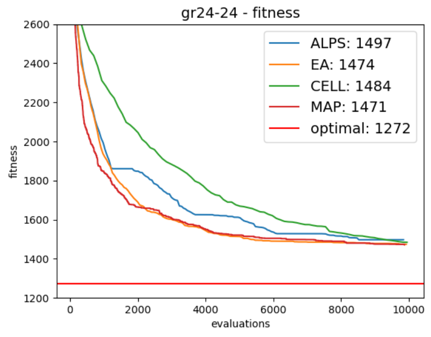
Figure 5.9: Comparison of algorithms on TSP problems for population size 50



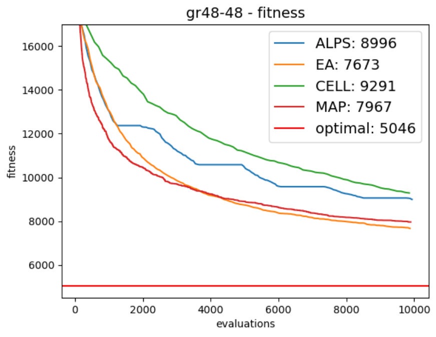
(a) gr17



(b) gr21



(c) gr24



(d) gr48

Figure 5.10: Comparison of algorithms on TSP problems for population size 100

Chapter 6

Conclusion

Evolutionary algorithms are powerful optimization methods, but have some limitations. A significant one is premature convergence when population of the algorithm converged to some local optimum and no more improvements could be made. A possible solution is to use structured population to maintain diversity in the population and thus improve performance of the algorithm. Several algorithms with structured population were tested and compared with simple evolutionary algorithm. These algorithms are ALPS: Age-Layered Population Structure, MAP-Elites: Multi-Dimensional Archive of Phenotypic Elites and cellular evolutionary algorithm. Algorithms were compared on various continuous problems as well as on traveling salesperson problems that are discrete. Based on conducted experiments, structured algorithms, except MAP-Elites, have overall significantly better performance than simple evolutionary algorithm for small population sizes on continuous problems. On TSP problems, all structured algorithms had better performance than a simple evolutionary algorithm, regardless of population size. The only exception is the instance with many dimensions, where apparently not enough evaluations were provided for conducted experiments, and thus results here are unclear.

Appendix A

Bibliography

[1] Optimization course. Cvut.cz. Available at: `https://cw.fel.cvut.cz/b192/_media/courses/b0b33opt/opt.pdf` (Accessed: August 10, 2021).

[2] Wikipedia contributors (2021b) Fitness landscape, Wikipedia, The Free Encyclopedia. Available at: `https://en.wikipedia.org/w/index.php?title=Fitness_landscape&oldid=1034646731` (Accessed: August 13, 2021).

[3] Hornby, G. S. (2006) "ALPS: The age-layered population structure for reducing the problem of premature convergence," in Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06. New York, New York, USA. Available at: `https://www.researchgate.net/publication/216300808_ALPS_The_age-layered_population_structure_for_reducing_the_problem_of_premature_convergence`

[4] Tomassini, M. (2005) *Spatially structured evolutionary algorithms: Artificial evolution in space and time.* 2005th ed. Berlin, Germany: Springer.

[5] Mouret, J.-B. and Clune, J. (2015) "Illuminating search spaces by mapping elites," arXiv [cs.AI]. Available at: `http://arxiv.org/abs/1504.04909`

[6] Quality Diversity optimisation algorithms: Toturials. Available at: `https://quality-diversity.github.io/tutorials`

[7] Gravina, D. Divergence and Quality Diversity: A collection of papers on divergence and quality diversity. Available at: `https://github.com/DanieleGravina/divergence-and-quality-diversity#quality-diversity-algorithms`

[8] Wikipedia contributors (2021) Bijection, injection and surjection, Wikipedia, The Free Encyclopedia. Available at: `https://en.wikipedia.org/w/index.php?title=Bijection,_injection_and_surjection&oldid=1023502940` (Accessed: August 12, 2021).

[9] Wikipedia contributors (2021) Travelling salesman problem, Wikipedia, The Free Encyclopedia. Available at: `https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1035459634` (Accessed: August 9, 2021).

[10] Abdoun, O., Abouchabaka, J. and Tajani, C. (2012) “Analyzing the performance of mutation operators to solve the Travelling Salesman Problem,” arXiv [cs.NE]. Available at: `http://arxiv.org/abs/1203.3099` (Accessed: August 12, 2021).

[11] Puljic, K. and Manger, R. Comparison of eight evolutionary crossover operators for the vehicle routing problem, Cvut.cz. Available at: `https://cw.fel.cvut.cz/b201/_media/courses/a0m33eoa/du/puljic2013crossoversforvrp.pdf` (Accessed: August 12, 2021).

[12] COCO: Numerical Black-Box Optimization Benchmarking Framework.Available at: `https://github.com/numbbo/coco` (Accessed: August 9, 2021).

[13] COCO: Real-parameter black-box optimization benchmarking 2009. Available at: `https://coco.gforge.inria.fr/lib/exe/fetch.php?media=download3.6:bbobdocfunctions.pdf` (Accessed: August 9, 2021).

[14] TSPLIB main page. Available at: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/` (Accessed: August 10, 2021).

[15] TSPLIB description of files. Available at: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf` (Accessed: August 10, 2021).

[16] TSPLIB assignments. Available at: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/` (Accessed: August 10, 2021)

[17] TSPLIB optimal values. Available at: `http://hhttp://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html` (Accessed: August 10, 2021).

Appendix B

Comments to feature descriptors

B.1 Continuous problems

Using our feature descriptor for continuous problems introduced in subsection 4.3.2 the number of dimensions of feature space and the number of cells in each dimension can not be arbitrary. If we want to ensure specific population size or alternatively the number of cells in the feature space that can be potentially occupied, we have to design proper feature space shape [1]. Therefore, approach using prime factorization was used. This approach is introduced in the pseudo-code.

Algorithm 6 Finding out proper shape of feature space

```
1: procedure FIND PROPER FEATURE SPACE SHAPE
2:     prime numbers list ← make prime factorization of the population size
3:     while length of prime numbers list > length of chromosome do
4:         product ← multiply two lowest numbers in the prime numbers list
5:         remove two lowest numbers in the prime numbers list
6:         add product to the prime numbers list
7:     return prime numbers list
```

So for instant, when we have population size equal to 100 and length of chromosome equal to 3 than our feature space shape would be [4, 5, 5].

B.2 Traveling salesperson problems

Here are some additional notes to the feature descriptor used for TSP problems that was introduced in subsection 4.3.3. The feature space cannot be created simply as two-dimensional space, where each dimension refers to the similarity with some path. The paths used in feature descriptor are not

[1]The population size of n-dimensional feature space is computed as $\prod_{i=1}^{n} x_i$ where x_i is the number of cells in the i-th dimension of the feature space

identical but have some shared edges. Therefore, there are cells in the feature space that can not be occupied in principle. There is a function that gives the number of cells that can be occupied in $n \cdot n$ feature space, where n is the dimension of the problem. Then m is the number of edges that paths used for feature descriptor have in common. This function I derived looks as follows: $f(m,n) = (n^2 + n + 2 \cdot m \cdot n - 3 \cdot m^2 - 3 \cdot m)/2$.

If the number of available cells is insufficient, we can simply add another dimension, but this case did not occur in this thesis. Finally, if the number of available cells is too big, we can adequately shrink the feature space.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Dulava** Jméno: **Tomáš** Osobní číslo: **483654**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Struktura populace v evolučních algoritmech

Název bakalářské práce anglicky:

Structured population in evolutionary algorithms

Pokyny pro vypracování:

Internal structure of a population in an evolutionary algorithm has been usually used as a means to preserve diversity in the population which is profittable e.g. when solving multimodal problems. The structure usually restricts with what other individuals from the population an individual competes, either for breeding or for survival. The goal of this project is to explore existing population structures, or to propose a new one, and evaluate and compare their effects on the run of evolutionary algorithms on unimodal and multimodal functions.
1. Familiarize yourself with the existing population structures proposed in literature.
2. Choose at least 3 of them (or propose your own) and implement them in an evolutionary algorithm.
3. Compare their effects on the algorithm performance on a set of benchmark problems/functions (BBOB, TSPLIB).

Seznam doporučené literatury:

[1] Gregory S. Hornby. ALPS: the age-layered population structure for reducing the problem of premature convergence. GECCO 2006
[2] Brian W. Goldman and William F. Punch. Parameter-less population pyramid. GECCO 2014
[3] Marco Tomassini. Spatially Structured Evolutionary Algorithms. Springer, 2005

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Pošík, Ph.D., katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.01.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Petr Pošík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání	Podpis studenta