Master's Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Multitasking in symbolic regression

Bc. Tomáš Dulava

Supervisor: Ing. Petr Pošík, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence
January 2024

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Dulava Tomáš**	Personal ID number:	**483654**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master's thesis details

Master's thesis title in English:

Multitasking in symbolic regression

Master's thesis title in Czech:

Multitasking v symbolické regresi

Guidelines:

In the thesis, multitasking is understood as the ability of an algorithm to solve several problems in a single run. The goal of the thesis is to explore the usage of multitasking in symbolic regression problems.
1. Survey the available literature on multitasking in black-box optimization in general, and in symbolic regression in particular.
2. Design or select several artificial and real-world datasets corresponding to symbolic regression tasks.
3. Choose a state-of-the-art (non-multitasking) symbolic regression algorithm as a benchmark.
4. Design and implement a multitasking algorithm for symbolic regression.
5. Evaluate the effects of multitasking on the algorithm runtime and achieved solution quality in the following scenarios.
* Compare a multitasking algorithm solving N problems at once with a non-multitasking algorithm solving N problems sequentially.
* Compare a multitasking algorithm solving problem P together with several artificially added problems at once with a non-multitasking algorithm solving only problem P.

Bibliography / sources:

1. Gupta, A., Ong, Y.-S., Feng, L. Multifactorial Evolution: Toward Evolutionary Multitasking. IEEE Trans. on Evol. Comp., Vol. 20, 2016.
2. Jaskowski, W., Krawiec, K., Wieloch, B. Genetic Programming for Cross-Task Knowledge Sharing. GECCO 2007.
3. Scott, E. O., De Jong, K. A. Multitask Evolution with Cartesian Genetic Programming. GECCO 2017

Name and workplace of master’s thesis supervisor:

Ing. Petr Pošík, Ph.D.   Department of Cybernetics FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **08.08.2023**   Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

Ing. Petr Pošík, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to express my deepest appreciation to Petr Pošík for his invaluable patience, knowledge, and guidance.

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, January 9, 2024

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

V Praze, 9. ledna 2024

Abstract

Evolutionary algorithms are a family of optimization techniques that are heavily inspired by organic evolution. They are popular mainly for their wide applicability. They are also suitable for black-box optimization where a description of the objective function is not known. Many variants of evolutionary algorithms were proposed during their history, but they were mainly focused on the optimization of one optimization problem at a time. Little effort was paid to multitasking which is an optimization of multiple optimization problems at the same run of evolutionary algorithm. Recently, a paper was published that introduced a new approach to evolutionary multitasking called multifactorial optimization. The motivation of this approach is better effectivity in optimization of several tasks concurrently than optimization of each problem independently. This work implements the multifactorial optimization for an important class of evolutionary algorithms which is genetic programming, that works with a tree representation of the candidate solutions. The goal of this thesis is the implementation of the genetic programming algorithm that optimizes in a multifactorial sense and subsequent analysis of its attributes and performance.

Keywords: multitasking, black-box optimization, evolutionary algorithms, genetic programming, multifactorial optimization

Supervisor: Ing. Petr Pošík, Ph.D.

Abstrakt

Evoluční algoritmy jsou skupinou optimalizačních technik, jež jsou inspirované organickou evolucí. Evoluční algoritmy mají širokou oblast použití a jsou také vhodné pro takzvanou black-box optimalizaci, kdy není znám předpis optimalizované funkce. Evoluční algoritmy mají dlouhou historii a byla přestavena řada variant jejich použití. Téměř všechny varianty se však zaměřovali na optimalizaci jedné účelové funkce najednou a málo pozornosti bylo věnováno optimalizaci vícero účelových funkci v jednom běhu evolučního algorithmu, tedy takzvanému multitaskingu. Nedávno byla přestavena metoda multifaktoriální optimalizace, která představuje způsob, jak se dají evoluční algoritmy efektivně aplikvat pro optimalizací vícero optimalizačních problému najednou. Hlavní motivací tohoto přístupu je očekávání lepší efektivity optimalizačního procesu, než při řešeni každého problému zvlášť. Tato práce obsahuje rekonstrukci navrhovaného řešení a následné analýzu jeho vlastností na jednu z významných odnoží evolučních algoritmů, kterou je genetické programování, které pracuje se stromovou reprezentací jedinců v populaci. Cílem této práce je anaýlza efektivity genetického programování implementující multifaktoriální optimalizaci.

Klíčová slova: multitasking, black-box optimalizace, evoluční algoritmy, genetické programování, multifaktoriální optimalizace

Překlad názvu: Multitasking v symbolické regresi

Contents

Figures

Tables

Chapter 1
Introduction

Evolutionary algorithms (EA) are a family of optimization algorithms inspired by organic evolution. There are plenty of EA techniques and approaches, but most of them are focused on single-objective (SO) optimization which is an optimization of one objective function at a time. If more objective functions needed to be optimized, then by the SO approach, the functions would be optimized sequentially, one after another. A little effort was paid to the possibility of optimizing multiple objective functions concurrently. This approach is also referred to as multitasking [1].

Potential benefit of multitasking is better efficacy and efficiency of the optimization process. If there is some similarity among optimal solutions of the objective functions, then if an optimum of one function is found, it could be used by another function and help it to find it its optimum. This phenomena might be especially beneficial in the cases, where a complex objective function, that is difficult to optimize, is optimized with another less complex task, with similar optimum. Another possible benefit might be a maintaining of genetic diversity in the population, if there is some exchange of genetic material between individuals of different objective functions.

On the other hand, the multitasking approach also might have a negative impact on the optimization process. There might be negative interference of the individual objective function. That might lead the multitasking optimization approach to spend more resources to reach the same level of optimality as in the single-objective approach.

In general, there are two main possible uses for a multitasking optimization:

- There is only one task that needs to be optimized, but other auxiliary tasks are added to the assignment to enhance the optimization process.
- There is a need to optimize several tasks and they are optimized concurrently.

[1]Note, that this multitasking described in the text, has nothing to do with parallel computing nor multi-threading. The multitasking is an optimization approach, no matter of the particular implementation of the algorithm.

One could ask about the efficacy of the multitasking approach, where multiple functions are optimized in the same run and there is allowed some information exchange between individual problems. The goal of this project is to assess the contributions of multitasking in both above mentioned scenarios compared to the single-objective approach. No other distinct algorithm is discussed in this project in detail.

This thesis is organized as follows. Chapter 2 introduces symbolic regression, which will be the goal of optimization throughout this project. Chapter 3 takes a deeper look into evolutionary algorithms and their functioning. A method of multitasking optimization called multifactorial optimization is introduced in Chapter 4. Algorithms of this project implement a branch of evolutionary algorithms called genetic programming which is described in Chapter 5. This chapter also discusses the implementation of multifactorial optimization by genetic programming. In Chapter 6 are introduced testing problems that are used to analyze the performance of designed algorithms. The settings of used algorithms are described in Chapter 7. This chapter also explains a process by which are found the most suitable settings for each algorithm. The performance of algorithms for testing problems is revealed in Chapter 8. The conclusion of the thesis is presented in Chapter 9.

Chapter 2

Symbolic regression

Let's have a system that consists of a set of variables. The values of these variables might not be independent, but they might be bonded by some kind of relationship. It might be desirable to estimate the relationship between those variables and thus obtain the model of the system.

Two types of variables are distinguished, independent and dependent. Independent are those variables that take value independently to other variables. Dependent variables are those variables that take value that is determined by values of the independent variables. For the sake of simplicity take into account only cases where there is a set of independent variables and exactly one dependent variable.

An example of this system might be a mathematical formula, for instance $y = 2 * x^2$. In this particular case, x is an independent variable that can take any value and y is a dependent variable, that is determined by the x. If the form of the mathematical formula is known, then there is no need to estimate anything. But it might happen that the exact mathematical formula is not known and only a set of value pairs of x and corresponding y might be obtained, such as [(1, 2), (-2, 8), (0, 0), (3, 18)]. How to get the original formula based on these values? How can one evaluate the correctness of the obtained formula? These questions might be answered by regression.

Regression [1, 2] is a set of statistical methods used for the estimation of relationships between independent variables (input of a function) and dependent variables (output of a function). By regression, a model of some system is obtained. The regression task consists of several basic ingredients [3]:

- Training set:
  Let's have a system that is the target of regression. Every observation of values of the variables of the system might be denoted as $(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y})$, where $\mathbf{x}$ represents values of independent variables and y represents value of the dependent variable. $\mathcal{X}$ is then space of values of independent variables and $\mathcal{Y}$ is space of values of dependent variable. Such an observation is called training example. The set of training examples is

then $\mathcal{T}_m = \{(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y}), i = 1, ..., m\}$

- Hypothesis class:
  For regression to take place, there must be defined a set of possible models. This set of models is also called hypothesis class, denoted as $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$. The function of the model is to derive value of the dependent variable based on the values of the independent variables, so the estimation of y by model $h \in \mathcal{H}$ is $y_h = h(\mathbf{x})$.

- Loss function:
  Loss function numerically evaluates a quality of a particular model h for a given training set $\mathcal{T}_m$. By $\mathcal{L}$ is typically denoted loss for a training set and by l is denoted loss for a training example. There is a plenty of possible loss functions, for instant root-mean-square error (RMSE). It is defined as:

$$RMSE(h, \mathcal{T}_m) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y_i - h(\mathbf{x}_i))^2}.$$

  This loss function will be used in all future uses of loss functions throughout this project.

The goal of the regression process is to choose a model with the lowest loss of all models from the hypothesis class. The idea of choosing the best model is captured by the following equation:

$$Regression(\mathcal{H}, \mathcal{T}_m) = \underset{h \in \mathcal{H}}{\arg\min}\, L(\mathcal{H}, \mathcal{T}_m)$$

The regression is a much broader topic than presented in the previous sections, but yet all necessary concepts needed in this project were introduced.

Symbolic regression is a special type of regression where the model of a function is presented as a symbolic expression. Every valid mathematical expression has its symbolic expression. This symbolic expression can be represented as a rooted directed acyclic tree (rooted DAG). For instance, take a look at image 2.1. In the image, a mathematical formula is presented, as well as its symbolic representation expressed by a tree. The tree has variables and constants in its leaves and mathematical operators in all other nodes. This holds for any symbolic expression.

The value of the expression can be obtained from the root of the tree by applying its operator to all its descendants. This is done recursively till to the leaves of the tree. If the tree contains only one node, then the value of the expression is equal to the value of this node. It also holds, that any subtree of the expression is also a rooted DAG, that represents a valid mathematical formula. Unfortunately, it does not hold that any two subtrees of the symbolic expression can be swapped and it is guaranteed to get a valid symbolic expression. But with caution, one can replace subtrees of the expressions by different subtrees and generate new expressions.

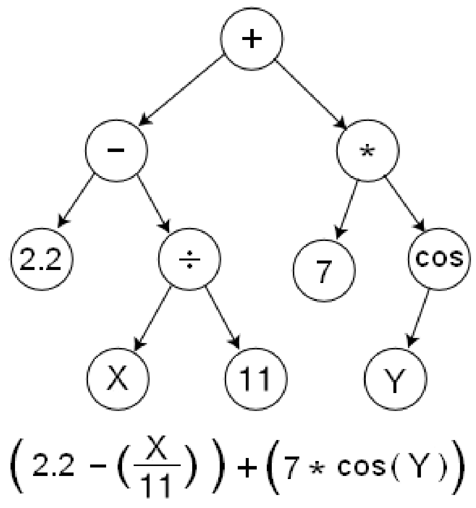

+
−
*
2.2
÷
7
cos
X
11
Y
$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$

Figure 2.1: A mathematical function and its symbolic representation [4]

Chapter 3

Evolutionary algorithms

Evolutionary algorithms (EA) are a family of optimization techniques inspired by organic evolution [7]. They are popular mainly due to their robustness and wide applicability. They can be applied to optimize a function even though a definition of the function is not provided, therefore EA algorithms are widely used in black-box optimization. EA were successfully applied in many fields of IT such as computer networks, image processing [5, 6], healthcare [8, 9] and others.

The basic workflow of an evolutionary algorithm looks as follows. At first, the population of candidate solutions is initialized. Then algorithm is iterated until terminal conditions are met. One iteration of evolutionary algorithm consists of following steps

- selection
- offspring generation
- evaluation
- replacement

In the following sections, the respective steps of the evolutionary algorithm are described in more detail.

3.1 Initialization

At first, random candidate solutions are generated for the target problem. These candidates can be generated according to some prior knowledge about the problem if available. Then all these individuals are evaluated according to the objective function of the optimization problem.

3.2 Selection

From the evaluated population a subset of individuals is selected. The selection can be performed under various criteria. The most straightforward approach is to select individuals according to their fitness values.

3.3 Offspring generation

The selected individuals are used to generate a new set of individuals. In EA terminology the original selected population of individuals is called parents and the population derived from the parents is called offspring. Offspring are generated using two evolutionary operators, mutation and crossover. Crossover is a process when two individuals are combined in order to create new individuals. The mutation takes only one individual and modifies it slightly to create a new individual.

3.4 Evaluation

When offspring are created, they are evaluated with respect to the objective function that is being optimized. This way, each individual obtains a fitness value.

3.5 Replacement

Having two sets of evaluated individuals, the previous population and offspring are used to create one new set of individuals for another iteration of the algorithm. In this process, a subset of individuals is kept and the rest is discarded. Often the original population is simply replaced by its offspring, but more advanced techniques were successfully introduced such as the elitist strategy, where the most fit individuals are chosen for the next iteration.

The whole process of EA is depicted in the pseudo-code Alg. 1. The EA runs as described until termination conditions are met. The termination criterion is typically number of generations, time of computation, or quality of the best individual in the population.

.

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 $population$
```

3.6 Extension of evolutionary algorithms

Many advanced techniques enhancing the efficacy of evolutionary algorithms were proposed and implemented such as structured populations [10, 11, 12, 13], local searches, adaptive parameters [14, 15] and others. The majority of these techniques though, are focused on the single-objective EA, where the algorithm has only one objective function to optimize. A little effort was paid to the multitasking approach, where several optimization tasks are optimized concurrently. Although some work was done in these regard[16, 17]. One of the state-of-the-art algorithms that addresses the multitasking approach is recently introduced multifactorial optimization. [18].

Chapter 4

Multifactorial optimization

The goal of the multifactorial optimization (MFO) [19] is to optimize multiple optimization problems concurrently and harness the potential latent genetic complementarities of the problems in order to more effectively optimize all the problems together than optimizing each problem independently. The purpose of MFO is not to find optimum trade-off among optimization problems, but to fully optimize each task.

Assume that K minimization optimization problems $T_1, T_2, ..., T_K$ are about to be solved simultaneously by the MFO algorithm. The population of candidate solutions consists of P individuals $p_1, p_2, ..., p_P$. With these general settings, let's take a look at several essential features of MFO, that are introduced in the following sections.

4.1 Unified representation scheme

MFO aims to be able to optimize distinct optimization tasks concurrently. In order to do so, MFO uses *unified representation scheme*. So, chromosome of each individual is represented as a vector $\mathbf{x} \in [0,1]^{D_{max}}$, $D_{max} = max\{D_j\}$, where D_j is the dimensionality of T_j. Each problem is mapped into interval $[0,1]^{D_j}$. To evaluate an individual for problem T_j, simply the first D_j genes of the chromosome are taken. This type of encoding operates as a layer of abstraction that allows the algorithm to take the same individual as a candidate solution for any optimization problem.

4.2 Factorial cost

For a given task T_j and an individual p_i the factorial cost is defined as $\Psi_j^i = f_j^i$, where f_j^i is simply the fitness value of p_i for T_j. [1]

[1]In the case of an objective function with constraints, the factorial cost is defined as $\Psi_j^i = \lambda \cdot \delta_j^i \cdot f_j^i$, where f_j^i is a fitness value of p_i for T_j, λ is a penalizing multiplier and δ_j^i is value of constraint violation.

4.3 Factorial rank

Let's have a list L_j, that represents the factorial cost for a particular task T_j and all individuals in the population, so $L_j = [\Psi_j^1, \Psi_j^2, ...\Psi_j^P]$. Also, let's have $\bar{L_j}$ that represents L_j sorted in ascending order. Factorial rank r_j^i is simply the index of p_i in the $\bar{L_j}$.

4.4 Scalar fitness

Let's have a list R^i, that represents factorial ranks of an individual p_i for each problem, so $R^i = [r_1^i, r_2^i, ..., r_K^i]$. The scalar fitness of p_i is defined as $\varphi_i = min\{R^i\}$ [2].

4.5 Skill factor

The skill factor of an individual p_i represents the task on which the individual is most effective. It is defined as $\tau^i = argmin\{R^i\}$

4.6 Multifactorial evolution

To determine the quality of an individual, it is needed to have some mechanism to compare individuals in the population. In the case of MFO, it cannot be done by comparing fitness values as in the single objective variant of optimization. A different approach must be adopted. For this purpose, scalar fitness is used. For two individuals p_a and p_b, the p_a is considered to dominate p_b in a multifactorial sense if and only if $\tau_a > \tau_b$. The factorial rank and therefore also scalar fitness and ordering of individuals are not absolute but they are determined by the quality of other individuals in the population. It holds that if an individual is the global optimum of some problem it will not be dominated by any other individual, thus the proposed comparison technique is indeed valid.
The basic workflow of the MFO is very similar to the one of EA, but MFO differs in several steps.

4.6.1 Initialization

After the individuals are generated, they are evaluated for each optimization task, in other words, each factorial cost is calculated. Then scalar fitness and skill factor of each individual are determined.

[2]In the original paper [18] introducing MFO, the Scalar fitness is defined as $\varphi_i = 1/min\{R^i\}$. The change was made so each optimization problem in this project is defined as a minimization

4.6.2 Offspring generation

Let's have two individuals p_a and p_b selected as parents. If these individuals have the same skill factor then they are considered as solutions to the same problem. By crossover, the parents produce offspring c_a and c_b. If the parents have the same skill factor, then the crossover is performed. If the parent differs in their skill factor, then the crossover is performed with random mating probability (rmp). If the crossover is not performed, then the offspring are produced by the mutation operator. Offspring inherit skill factors from their parents. If offspring are produced by parents with different skill factors, then the skill factor of an offspring is set to be as skill factor of the randomly selected parent.

4.6.3 Evaluation

When offspring are created, they are evaluated with respect to their skill factor only. This approach saves a lot of potentially wasted evaluations. As a consequence, once a skill factor of offspring is determined, it will never change afterward. This approach is not a necessity but is advantageous from a practical viewpoint.

Chapter 5

Genetic programming

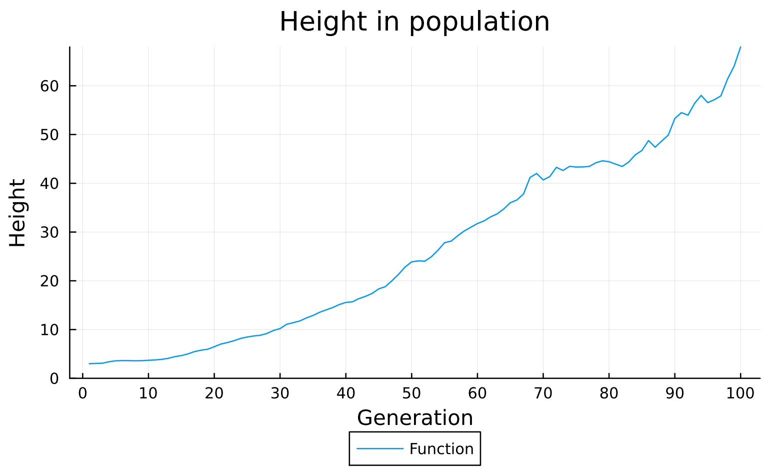
Genetic programming is a branch of evolutionary computation. The key difference between standard evolutionary algorithms and genetic programming is that evolutionary algorithms represent candidate solutions as vectors of numbers whereas genetic programming works with trees. Due to this fact, genetic programming is used on different optimization problems. Moreover, with tree representation there come new difficulties that have to be handled and are not present in evolutionary algorithms. The most prominent one is the bloat problem.

5.1 Bloat problem

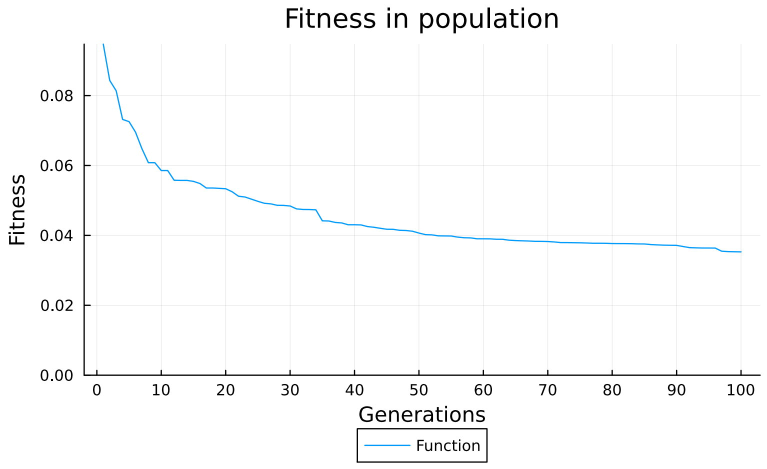
During the evolution of genetic programming (or any evolutionary algorithm using variable length of the genotype), the candidate solutions can change their size due to the mutation and crossover operators. It might happen, that the individuals tend to grow on their size without any meaningful benefit in terms of their fitness values. This phenomenon is known as the bloat problem. An illustration of the bloat problem is depicted in Fig.5.1

One might think that the explanation for the bloat problem is that the mutation operator tends to add a lot of new genetic material, thus the candidate solutions tend to get bigger over time regardless of their fitness values. To test this hypothesis, an evolutionary algorithm is tested on a constant function with a random selection. The results are depicted in Fig.5.2. One can conclude based on the graph, that with the very same settings, we got very different progress in terms of the height of the individuals. The average height tends to get bigger over time, but this effect is not as pronounced as in the case of the non-constant function, so the bloat problem is not just a bias of the algorithm in adding genetic material. The cause of the bloat problem has a different explanation.

The bloat problem was intensively studied and there are several explanations for such a phenomenon [21, 22, 23]. Detailed discussion of the reasons for bloat is out of the scope of this project.



(a) : Development of height of tree individuals of over time

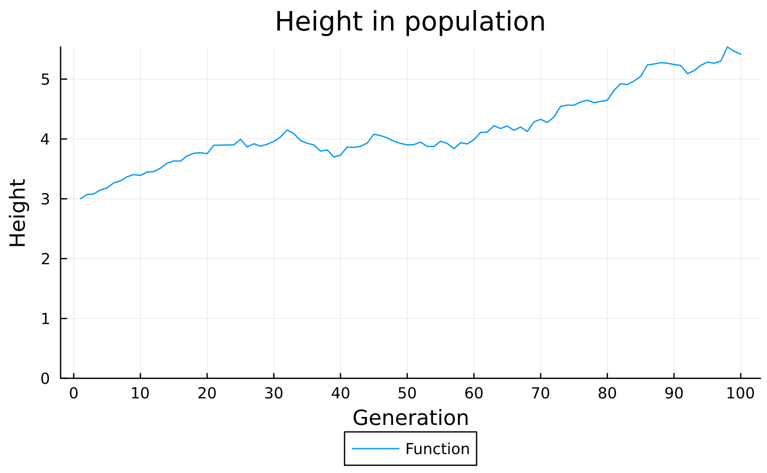


(b) : Development of fitness of tree individuals of over time

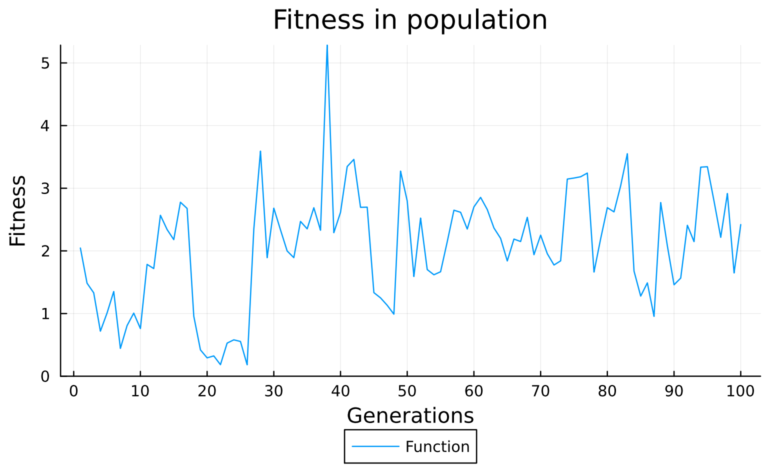
Figure 5.1: Illustration of the bloat problem

Two main difficulties come with the bloat problem. The first one is computational efficacy. Evaluation of a big tree is computationally more demanding than evaluation of a small tree, so the optimization process might create fewer candidate solutions with the provided computational resources. The second difficulty refers to the interpretability of the tree. It is easier to interpret the function of a small tree than a large tree with a lot of redundant genetic material. In addition, the bigger the tree is, the harder it is to make any modifications and adjustments to it.

There were introduced many techniques and ideas on how to deal with the bloat problem [24, 25, 26]. Among the most common ones belongs introducing a fixed maximal depth of the tree, defining tree size as a second criterion of



(a) : Development of height of tree individuals of over time



(b) : Development of fitness of tree individuals of over time

Figure 5.2: Illustration of the bloat problem for constant function

multiobjective optimization problem, updating the fitness value of a tree by a penalization, that depends on the complexity of the tree, and others. Several techniques of bloat control are tested and discussed in the experimental section of this work.

5.2 Multifactorial optimization in genetic programming

It was already discussed, how the multifactorial optimization works, what are the advantages and possible disadvantages of this approach, and how it is implemented in the case of evolutionary algorithm. A question arises, how to transfer and successfully use multifactorial optimization for genetic programming. It might be worth reminding that the key difference between evolutionary algorithm and genetic programming is, that in the genetic programming the tree representation is used instead of the vector one.

Successful attempts [19, 20] of using multifactorial optimization for genetic programming were made, but so far only with trees of fixed size. To the best knowledge of the authors of this project, no previous attempts of multifactorial optimization with trees of variable length were made.

One can conclude, that many concepts of multifactorial optimization such as factorial cost, factorial rank, scalar fitness, and skill factor can be used in genetic programming without any struggle. A bit of a challenge comes with the unified representation scheme. However, this project is focused primarily on symbolic regression of real-valued functions. Also an assumption of no prior knowledge about the function, in terms of present operators, is made. With this in mind, one can easily decide to use the very same set of operators in the optimization of each function. One thing that might vary though, is the dimensionality of the optimized function. It is assumed in this project, that the dimensionality of the objective function is known.

Another important thing is the process of evaluation of an individual. In this project, the evaluation works as follows. For an objective function, generate a corresponding number of samples. Given a candidate solution, compute its fitness for all samples. Assemble these fitnesses with root-mean-square error and obtain the final fitness of the individual.

Chapter 6
Testing problems

It is a crucial part of analyzing an algorithm, to come up with a robust, relevant, and suitable testing set to explore the algorithm's performance.

But first, take a look at the terminology used throughout this project. Any objective function is called a *task*. An optimization problem is called an *assignment* and it consists of one or more tasks. So each assignment of SO has exactly one task and each assignment of MFO has at least two tasks.

The testing problems used for the analysis of proposed algorithms are heavily inspired by the benchmark set presented in the paper "Genetic Programming Needs Better Benchmarks"[27]. In this project, there are 4 assignments used in total.

6.1 Koza

This assignment consists of the following tasks:

- Koza-1: $x^6 - 2x^4 + x^2$
- Koza-2: $x^5 - 2x^3 + x$
- Koza-3: $x^4 + x^3 + x^2 + x$

The domain of all tasks is $X = [-1, 1]$. All tasks are polynomial, but the orders as well as signs and coefficients slightly differ. Nonetheless, all tasks have non-empty intersection, thus there are possible benefits in sharing genetic information across tasks.

All Koza tasks are depicted in Fig 6.1.

6.2 Nguyen

This assignment consists of tasks

- Nguyen-1: $x^3 + x^2 + x$

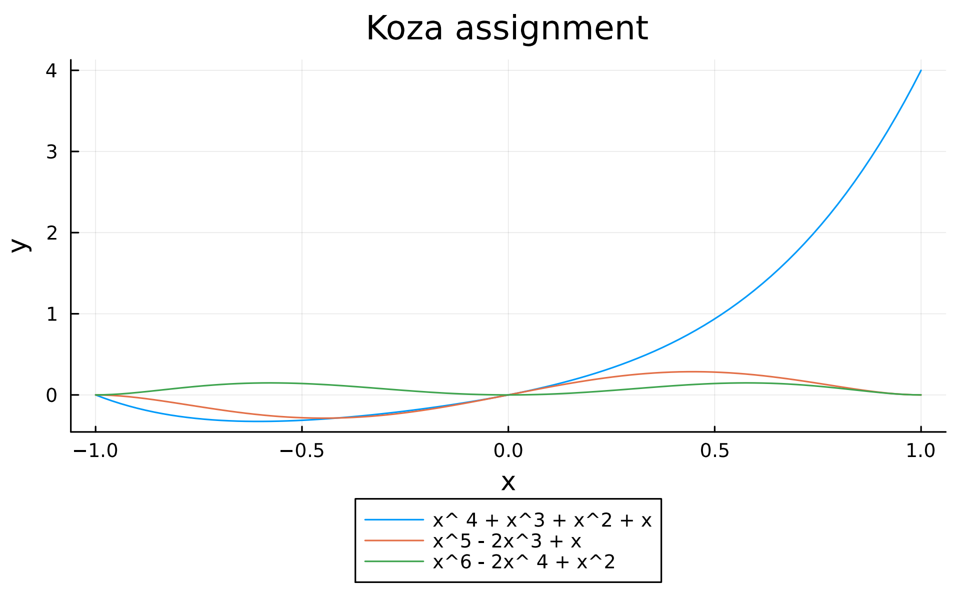


Figure 6.1: All tasks present edin Koza assignment

- Nguyen-2: $x^5 + x^4 + x^3 + x^2 + x$
- Nguyen-3: $x^6 + x^5 + x^4 + x^3 + x^2 + x$

The domain of all tasks is $X = [-1, 1]$. As in the Koza assignment, all tasks in Nguyen are polynomials, but their similarity is more pronounced. All members of each task are positive and have coefficients equal to 1. As a matter of fact, each Nguyen task contains the previous one and just adds one or more members to it.

All Nguyen tasks are depicted in Fig.6.2. From the graphical point of view, as well as from the symbolic one, all Nguyen tasks are very similar. The expectation is that Nguyen assignment will have the most promising results in terms of the beneficial effect of the multifactorial approach.

6.3 Keijzer

This assignment consists of tasks expressed by the very same mathematical formula. The difference between individual tasks is in their domains. These tasks are:

- Keijzer-1: $0.3xsin(2\pi x)$ for $x \in [-1, 1]$
- Keijzer-2: $0.3xsin(2\pi x)$ for $x \in [-2, 2]$
- Keijzer-3: $0.3xsin(2\pi x)$ for $x \in [-3, 3]$

All Keijzer tasks are depicted in Fig. 6.3.

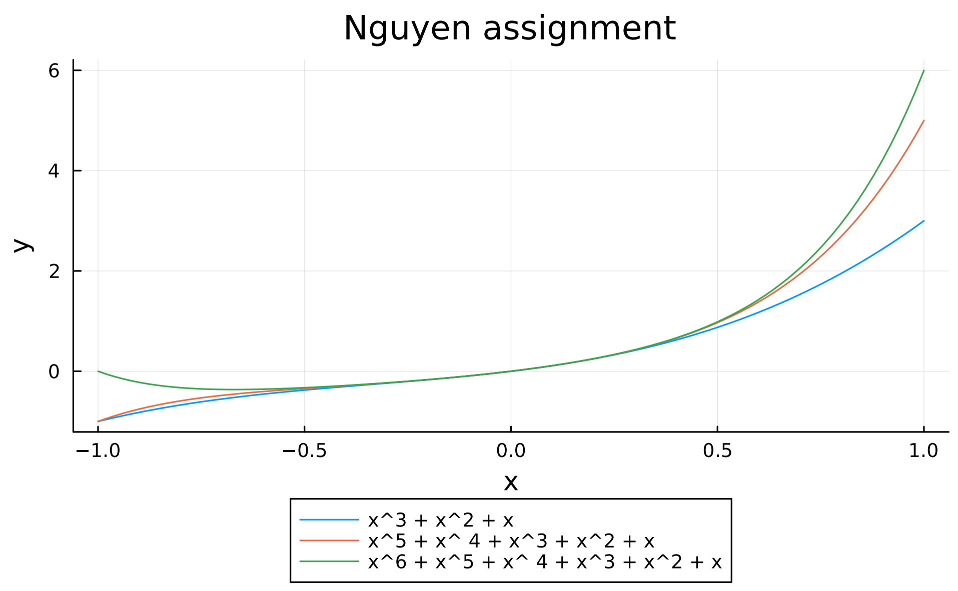


Figure 6.2: All tasks presented in Nguyen assignment

6.4 Keijzer2

All previous assignments were one-dimensional, but Keijzer2 is two-dimensional. The tasks look as follows:

- Keijzer2 - 1: $xy + sin((x-1)(y-1))$
- Keijzer2 - 2: $x^4 - x^3 + y^2/2 - y$
- Keijzer2 - 3: $6sin(x)cos(y)$

The domain of all tasks is $X = [-3, 3]^2$. Keijzer2 tasks are depicted in Fig. 6.4. All Keijzer 2 tasks are symbolically more distinct than tasks in the previous assignments. Thus, the expectation is that multifactorial optimization will struggle with this assignment.

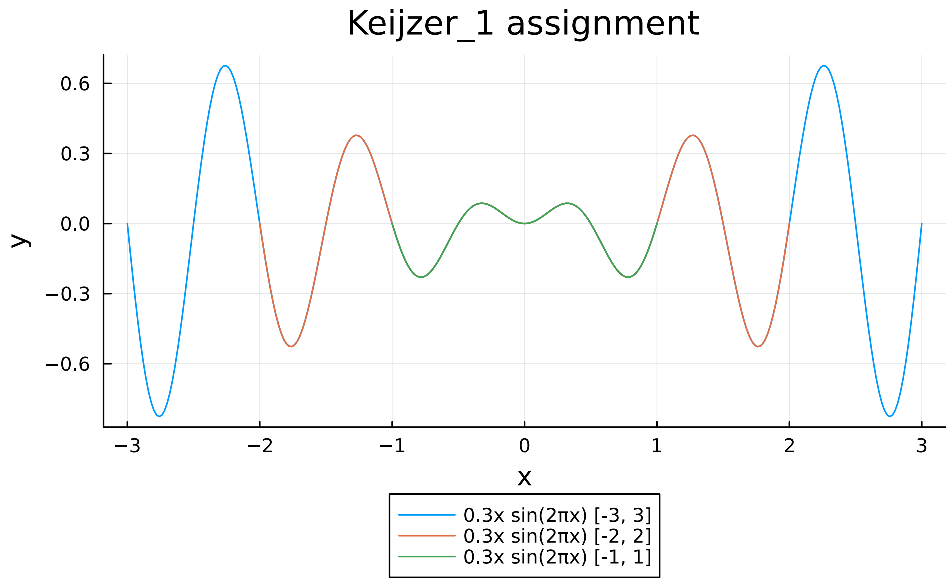
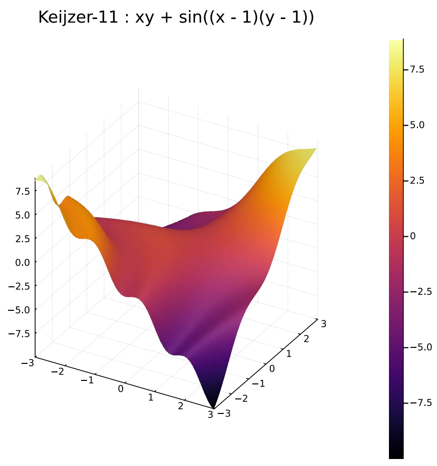
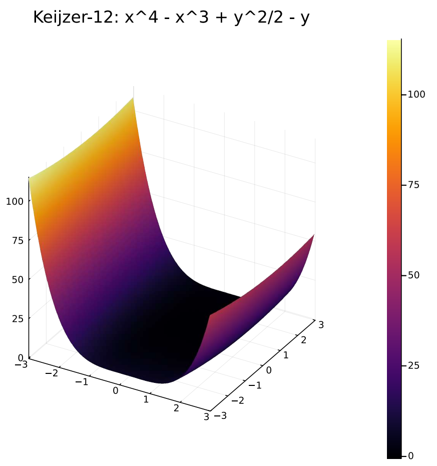


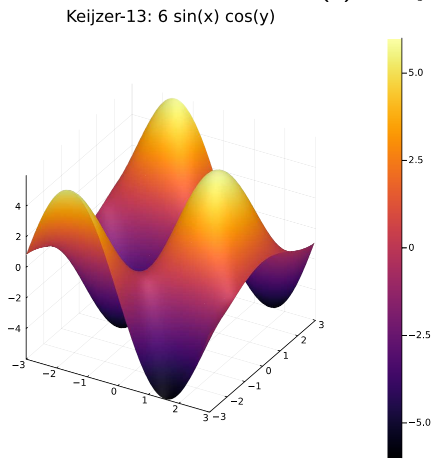
Figure 6.3: All tasks presented in Keijzer assignment



(a) : Keijzer2 - 1 task



(b) : Keijzer2 - 2 task



(c) : Keijzer2 - 3 task

Figure 6.4: All tasks presented in Keijzer2 assignment

Chapter 7

Parameters

Algorithms of evolutionary computation have usually several parameters that determine, how the algorithms work and how effective they are. Some parameters have a greater influence on the efficacy of the algorithms than others, but all parameters should have to be set to some reasonable values.

7.1 Description of parameters

This project also contains several parameters, that modify the behavior and efficacy of the algorithm.

7.1.1 Population size

This parameter determines the number of individuals present in the population. In this project, population size is constant. But there exist many EA algorithms with dynamic population size, for instance ALPS [11] or MAP elites[12].

Even though this project has a constant population size, the population is not panmictic in the case of MFO. Unlike the single-objective approach the MFO has structured population due to the restricted crossover probability, which depends on the skill factor of particular individuals.

In general, the population size can take any positive integer value. In the case of multifactorial optimization (MFO), the population size should be at least one individual for every objective function the MFO is intended to optimize.

7.1.2 Selection strategy

A random selection strategy is used. The probability of mating is uniform throughout the whole population. Note that even when two individuals are chosen to mate with each other, the crossover itself might not be performed. It depends on the skill factor of the individuals and random mating probability.

7.1.3 Replacement strategy

An elitist strategy is used. The original and offspring populations are merged and the half of the population with the minimal factorial cost is chosen for the next iteration of the algorithm.

7.1.4 Stopping condition

There has to be defined a condition that determines when to terminate the evolutionary algorithm. Very often it is the number of generations that the algorithm is allowed to proceed or the number of evaluated individuals. However, since this project uses a variable length of genotype of candidate solutions, this end condition might not be very suitable. Since the computational resources needed for an evaluation of a candidate solution are strictly connected to the amount of genetic material the individual has, the number of evaluated tree nodes might be more suitable in reflecting how computationally demanding an evaluation of a candidate solution is.

So for a run of the algorithm, a budget of evaluations of nodes must be set. After the budget is depleted, the algorithm is terminated and then the individuals with the best fitness for each objective function are returned.

7.1.5 Evaluation budget

This parameter determines how many evaluations of a defined type (generations, individuals, nodes, etc.) are proceeded until the algorithm ends.

7.1.6 Number of runs

Since the evolutionary computation algorithms (ECA) are stochastic, the algorithms might have a different performance every time they are launched. With this behavior, it might be difficult and uncertain to analyze the performance of a particular algorithm or make a comparison of different algorithms. A common procedure to deal with this issue is to launch a particular algorithm independently several times and then make an average of their performance over all runes. The number of runs parameter then refers to the number of independent runs the results of the algorithm are averaged over.

The Number of runs can take any positive integer value. The higher value the more statistically significant the results are.

7.1.7 Random mating probability

This parameter was already discussed in the theoretical section about multifactorial optimization in Chapter 4. It determines with what probability two individuals with different skill factors shall undergo crossover with each other. If the crossover is not performed, than individual mutation kicks in.

Random mating probability can take any value from $(0, 1)$

7.1.8 Random crossover probability

In the case of single-objective (SO) optimization, there are no individuals of different skill factor, thus random mating probability parameter has no effect. However, it is still desired to control the frequency of crossover and mutation. Therefore, random crossover probability is introduced. In the single-objective optimization two individuals undergo crossover with this probability, otherwise, mutation is performed.

Random mating probability can take any value from $(0, 1)$

7.1.9 Evaluation type

As already discussed in the theoretical section about genetic programming, the fitness value of a tree individual is obtained as the value of the root of the tree. An alternative evaluation of an individual is introduced in this project.

To evaluate the root of a tree, all other nodes of the tree must be evaluated first. So fitness value of an individual can be also obtained as the fitness value of any node of the tree. This can be done without any additional evaluations.

The alternative evaluation is simply obtaining the fitness value as the value of the root of the best subtree in the individual, the tree itself included.

The evaluation type can be:

- Root - Fitness value is obtained from the root
- All nodes - Fitness value is obtained from the best subtree

7.1.10 Local improvement

As already mentioned in the section about the bloat problem, the complexity of the tree might grow over time with a crippling effect on the performance of the algorithm. To avoid this phenomenon, some countermeasures must be made.

In this project, it is done by local improvement method, where an individual tree is replaced by some of its subtree. The question is when to execute this local improvement and what subtree to choose.

In this project, there are designed two methods that decided when to undergo this replacement:

- Always - In each iteration, each tree in the population is replaced by its best subtree.
- On height - In each iteration, the height of each tree is measured. If the height of the tree is greater than the *Height threshold* value, then this tree is replaced by its best subtree of maximal height *Height limit.*

Note, that the Always method is a special case of the On height method with threshold 0 and infinite height limit. But for clarity, these two strategies will not be unified in future sections.

7.1.11 Height threshold

An auxiliary parameter used in the On height local improvement method. It determines what minimal height the tree shall have to undergo local improvement.

The Height threshold can take any integer value.

7.1.12 Height limit

Another auxiliary parameter used in On height as well as in the Always local improvement method. It determines what maximal height a subtree can have to replace the original candidate solution.

The Height limit can take any positive integer value.

7.2 Parameter tuning

As laid out in the previous section, genetic programming has a lot of parameters to set. One might ask what are the optimal values for these parameters for the algorithm to be as effective as possible. This is indeed a tough question and that is so for several reasons. Firstly, the parameters are very often not independent but interfere with each other. Secondly, the optimal settings of parameters might vary from task to task. Finally, there are many parameters that can take on various values, so finding the right settings might be very difficult due to the curse of dimensionality [28]. Despite these difficulties, an experiment was conducted to get at least a rough idea of how different configuration settings affect the efficacy of the algorithm and what configuration setting is the most preferable one if no prior knowledge, about the task to optimize, is known.

Configurations			
Evaluation type	Local improvement	Height threshold	Height limit
All nodes	Always	x	10
All nodes	Always	x	20
All nodes	On height	10	10
All nodes	On height	10	20
All nodes	On height	20	10
All nodes	On height	20	20
Root	Always	x	10
Root	Always	x	20
Root	On height	10	10
Root	On height	10	20
Root	On height	20	10
Root	On height	20	20

Table 7.1: All configuration settings for the bloat control parameter tuning

To reduce the unpleasant effect of the curse of dimensionality the set of parameters are divided into two parts that are tuned independently. The parameters were divided so, that there are expected low relations between parameters of different parts, even though the effect of interference is not guaranteed to be not present. So it might happen, that sub-optimal settings will be found.

In the first group, parameters related to evaluation and bloat control are presented. Namely, these parameters are *Evaluation type*, *Local improvement*, *Height threshold*, *Height limit*. For both *Evaluation types* and for both *Local improvement* strategies, values 10 and 20 were tested for *Height threshold*, and the same values were also tested for *Height limit*. In total, it makes 12 different configurations to test. The tested configurations are depicted in the Tab. 7.1

In the second group of parameters, settings related to population control are presented. These parameters are *Population size* and *Random mating probability* for the multifactorial algorithm and *Random crossover probability* for the single-objective algorithm respectively. The tested values of *Population size* are 50, 100, 150, 200. The *Random mating probability* or *Random crossover probability* was tested for values 0.3, 0.6, and 0.9. In total, it makes 12 different configurations. These configurations are depicted in Tab. 7.2

Moreover, in order to make a fair comparison of multifactorial optimization (MFO) and single-objective optimization (SO), both algorithms were tuned individually. So the whole tuning process looks as follows. First, an algorithm is tuned for the first group of parameters. For this purpose the parameters from the second group were set to some reasonable moderate values, more

Configurations	
Population size	RMP / RCP
50	0.3
50	0.6
50	0.9
100	0.3
100	0.6
100	0.9
150	0.3
150	0.6
150	0.9
200	0.3
200	0.6
200	0.9

Table 7.2: All configuration settings for the population control parameter tuning

precisely the *population size* was set to 100 and *random mating probability* was set to 0.5 for MFO and the *random crossover probability* was set to 0.5 for SO respectively. After the tuning for the first group of parameters, the tuning of the second group of parameters takes place. For this purpose, the parameters from the first group are set to the values found in the first part of the tuning.

Another important question is on which assignment defined in Chapter 6 tests the configuration settings. If different configurations are tested on just one assignment, it could happen that the optimal configuration found, will be optimal just for the one assignment, and it will have very poor efficacy for different assignments. The goal of this parameter tuning is to find a configuration, that works well in general, therefore the configurations will be tested on several different tasks and the results will be then compiled together.

But how reasonably aggregate results of different configurations across different assignments? The first idea, that might come into mind, is to just sum the best fitnesses from all assignments and choose the configuration with the lowest sum value. The significant drawback of this approach is that different assignments might have fitness values of different magnitudes. Therefore it might happen, that just one assignment will be taken into account and the others will have just negligible effect. To avoid this drawback a relative comparison of configurations is proposed.

The relative comparison of configurations works as follows. At first, all assignments for each configuration setting are conducted and the results are stored in the corresponding logbooks. At this point, all raw data are available. Secondly, for each assignment, one after another, the absolute data in the logbooks are transformed into relative values. This is done by averaging fitness values for a particular assignment across all configurations and thus obtaining the average fitness value for a given assignment. All fitness values of the assignment are then divided by this averaged value. By this process, a relative performance of each configuration for a given assignment is obtained. This is done for each assignment. At this point all relative values are available. Thirdly, make a compilation of results across all configurations. This is done by computing an average of the performance of a given configuration across all tasks in all assignments.

This approach has some drawbacks as well. The main one is that, from the relative values it is not possible to analyze the performance of individual configuration settings. For instance, if the relative fitness of a configuration setting grows, one cannot tell what is happening with the absolute fitness. It could grow faster than the fitnesses of other populations, it could be constant, but the fitnesses of other populations tend to decline, or it could decline, but the fitnesses of other populations tend to decline much faster. But since the goal of the parameter tuning is to find the best configuration setting from a given set, this drawback is not significant and the proposed approach is suitable.

It is true that trying to optimize all parameters at once and to test more different configurations might bring better results and find better configuration of settings. But to do so, much more computational power would be needed. The introduced approach was chosen as a compromise between efficacy and time resources.

7.2.1 Tuning of multifactorial optimization

Let's take a look at the result of applying the proposed parameter tuning method for the multifactorial optimization. The development of efficacy of different configurations is depicted in graphs.

In the graph Fig. 7.1, there is a line for each configuration. To make the graph more intelligible, the lines are distinguished based on the configuration they represent. *Evaluation type* is expressed by line pattern. *All nodes* is solid and *Root* is dotted. *Local improvement* is expressed by width of the line. *Always* is slim and *On height* is thick. Different configurations of particular Evaluation type are then distinguished by different colors.

Based on the graph, one could conclude that configuration with the Always strategy tends to be more effective than those with the On height strategy. Also, each configuration with All nodes evaluation is more effective than its Root counterpart. From these observations, one might conclude that configuration with the All nodes evaluation type and the Always strategy might be the most suitable one for future experiments. For this configuration the Height limit 10 has better performance than 20, thus it will be chosen. In conclusion, in all future experiments, multifactorial optimization will have configuration with these settings: Evaluation type is All nodes, Local improvement is Always, Height limit is 20.

In Fig. 7.2 are shown results of parameter tuning of parameters related to population control i.e. population size and random mating probability. As the graph shows the algorithm is more effective with a higher population size as 150 or 200. Also for population sizes 50 and 100, it looks like the algorithm has better efficacy for lower random mating probability. No other obvious tendency can be observed from the data, thus the configuration with the best performance is chosen. In conclusion, in all future experiments, multifactorial optimization will have a configuration with these settings: Population size is 150, and Random mating probability is 0.6.

7.2.2 Tuning of single-objective (SO) optimization

For the single-objective optimization was conducted a very similar experiment to the one for multifactorial optimization. The key difference is that in the single-objective approach, only one task can be optimized in the assignment. So, instead of optimizing 4 assignments each with 3 tasks, 12 individual tasks were optimized. The results of each task were averaged over 6 independent runs.

The graph design distinguishing different configurations is the same as in the multifactorial case.

The bloat control tuning for SO is depicted in Fig. 7.3. In the single-objective approach, there can not be found the same efficacy tendencies in the bloat control tuning as in the multifactorial approach. The All nodes Evaluation type occupies the best as well as the worst positions in the performance. The Always strategy also does not show any convincing efficacy. Based on the graph, in all future experiments single-objective optimization will have a configuration with these settings: Evaluation type is All nodes, Local improvement is On height, Height threshold is 10, and Height limit is also 10.

The SO population control tuning is depicted in Fig. 7.4. From the graph, one can observe the very strong tendency of SO to have poor performance for population size 50. The performance for higher population sizes seems almost identical, but there just are relatively close to each other in comparison with the very poor performance of population size 50. In conclusion, in all future experiments, single-objective optimization will have a configuration with these settings: Population size is 150, and Random mating probability is 0.3.

7.3 Final parameters

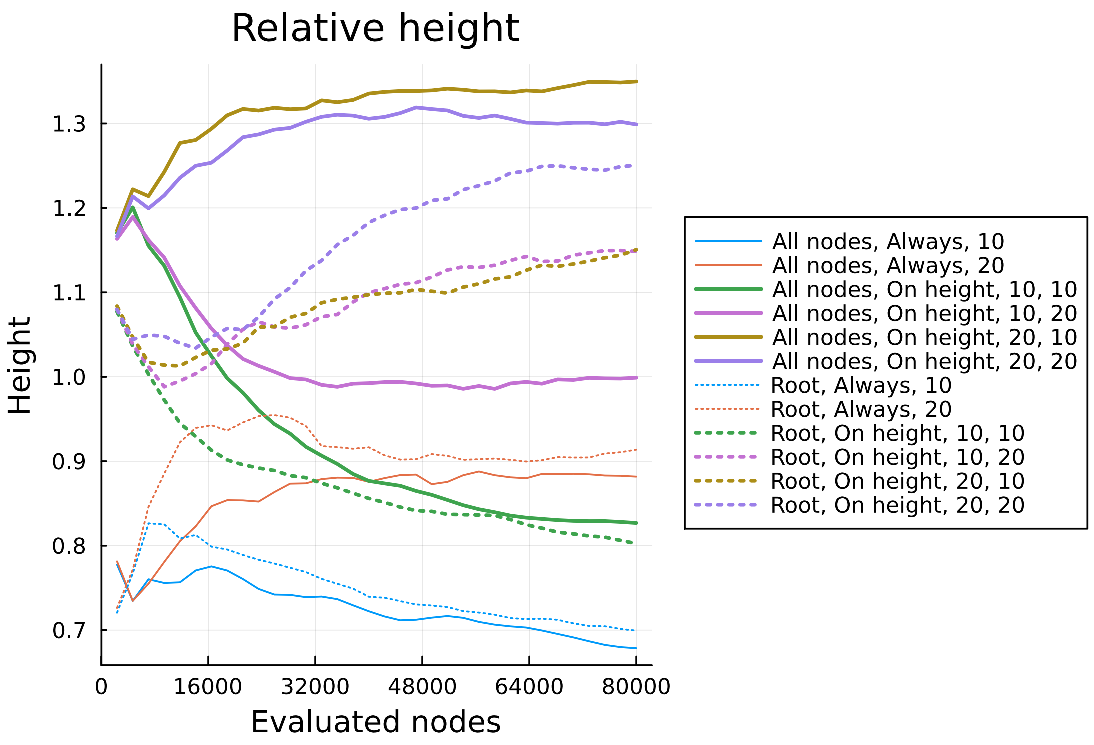
After the tuning process, the following parameter values were decided to be set for conducting of all experiments.

For multifactorial optimization:

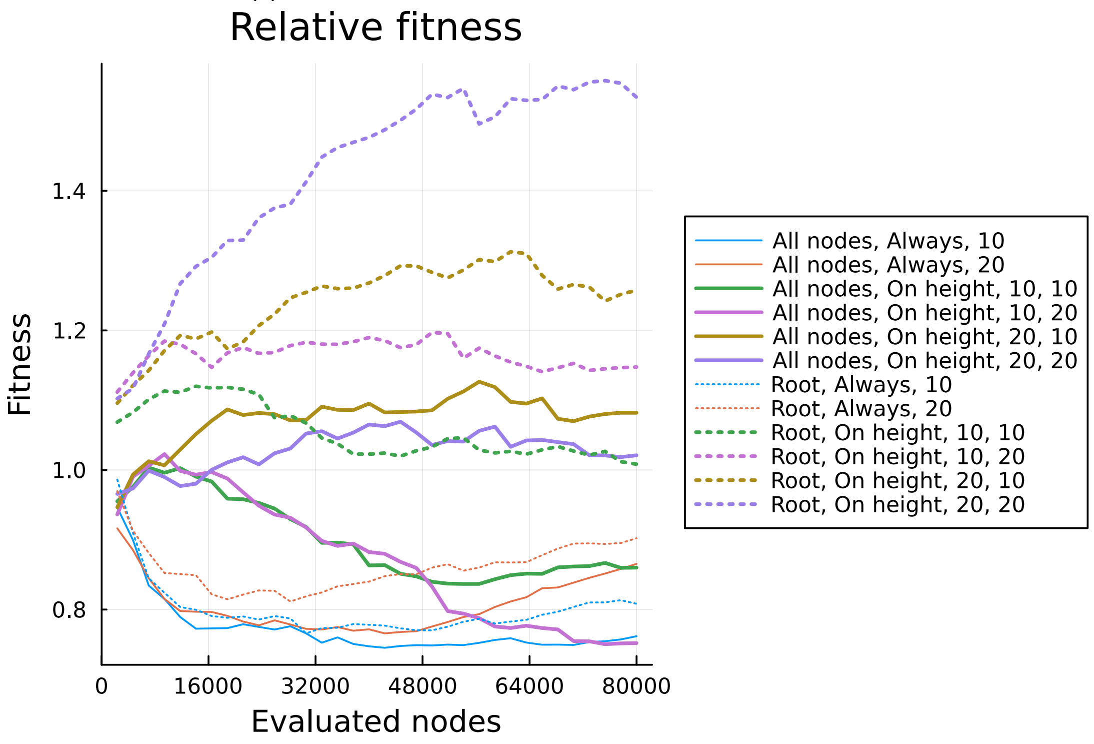
- End condition: Number of nodes
- Evaluation budget: 80000
- Number of runs: 100
- Population size: 150
- Random mating probability: 0.6
- Evaluation: All nodes
- Local improvement: Always
- Height limit: 10

For single-objective optimization

- End condition: Number of nodes
- Evaluation budget: 80000
- Number of runs: 100
- Population size: 150
- Random crossover probability: 0.3
- Evaluation: All nodes
- Local improvement: On height
- Height threshold: 10
- Height limit: 10

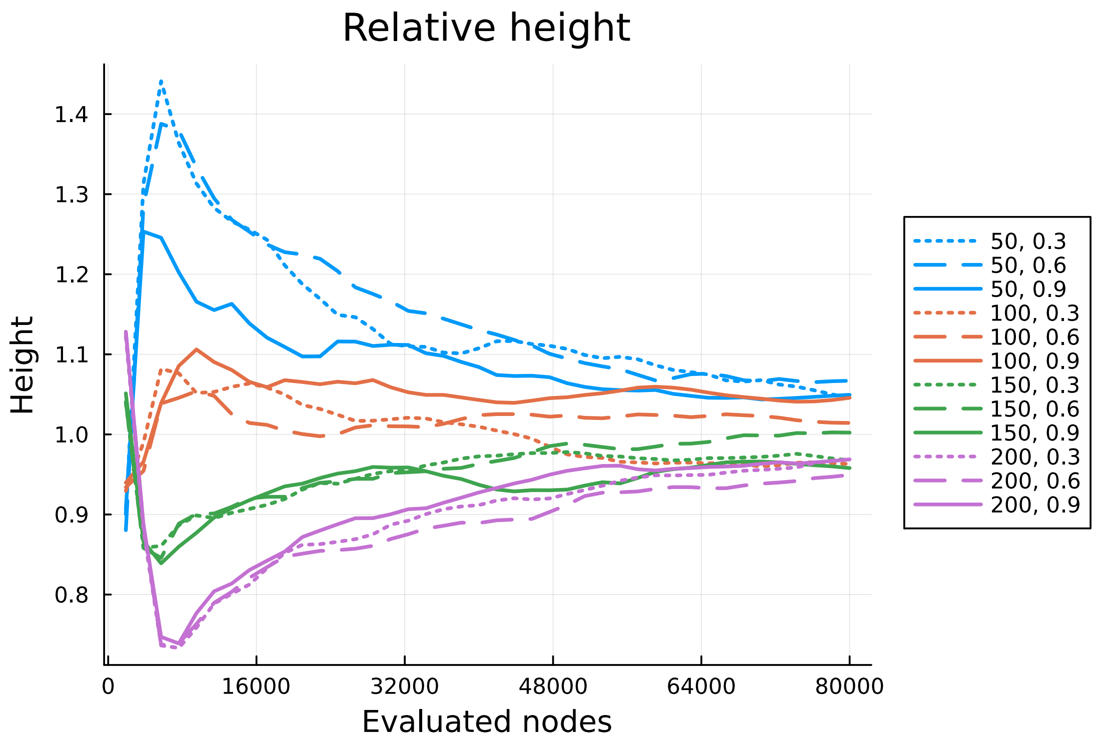


(a) : Development of height of tree individuals over time

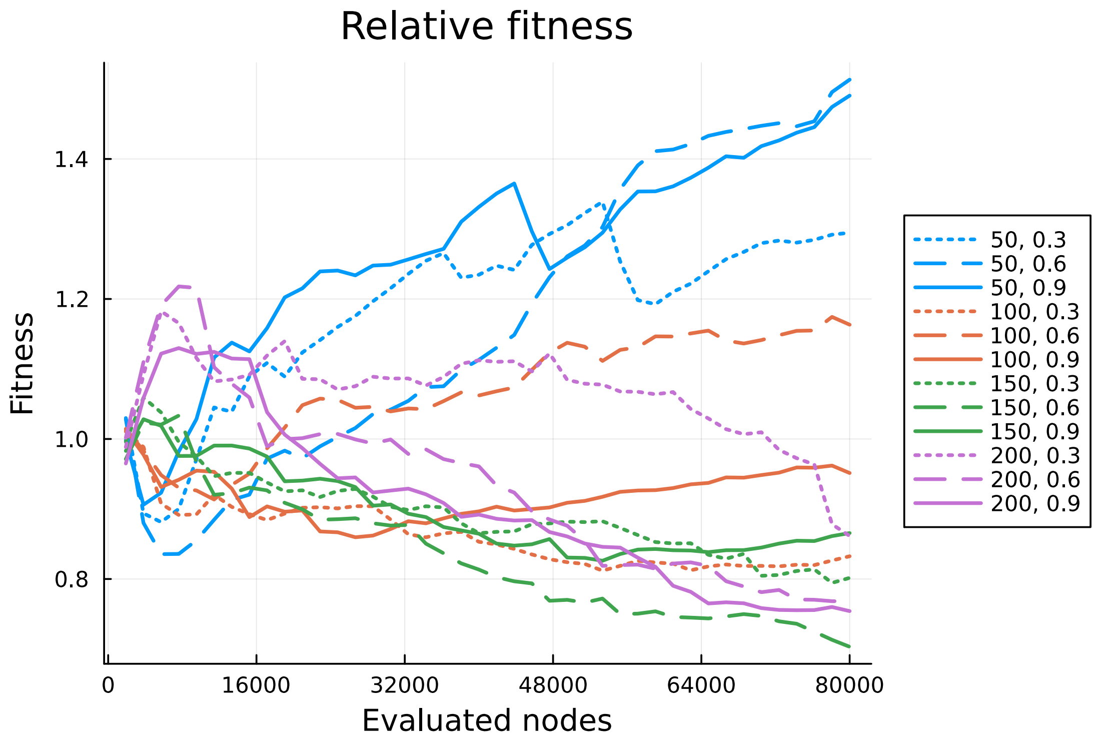


(b) : Development of fitness of tree individuals over time

Figure 7.1: Bloat control parameter tuning for MFO. Fitness shall be minimized. In the legend of each graph, the individual parameters are separated by a comma. The parameters are: Evaluation, Local improvement, Height threshold, Height limit.

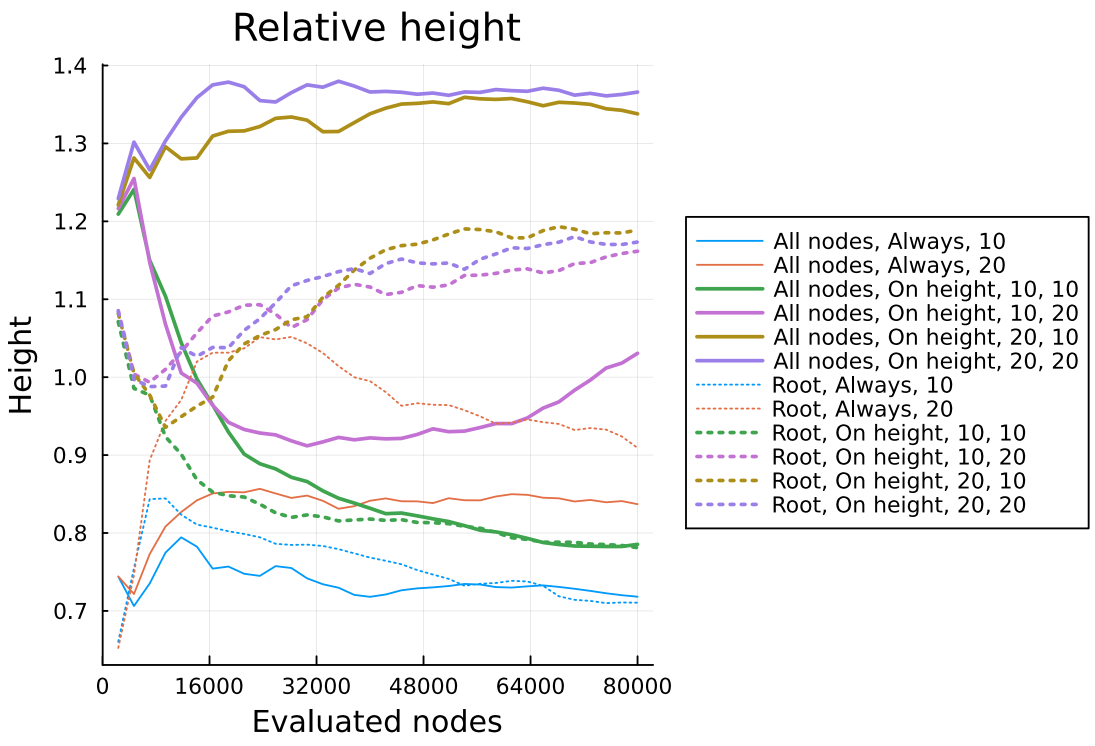


(a) : Development of height of tree individuals over time

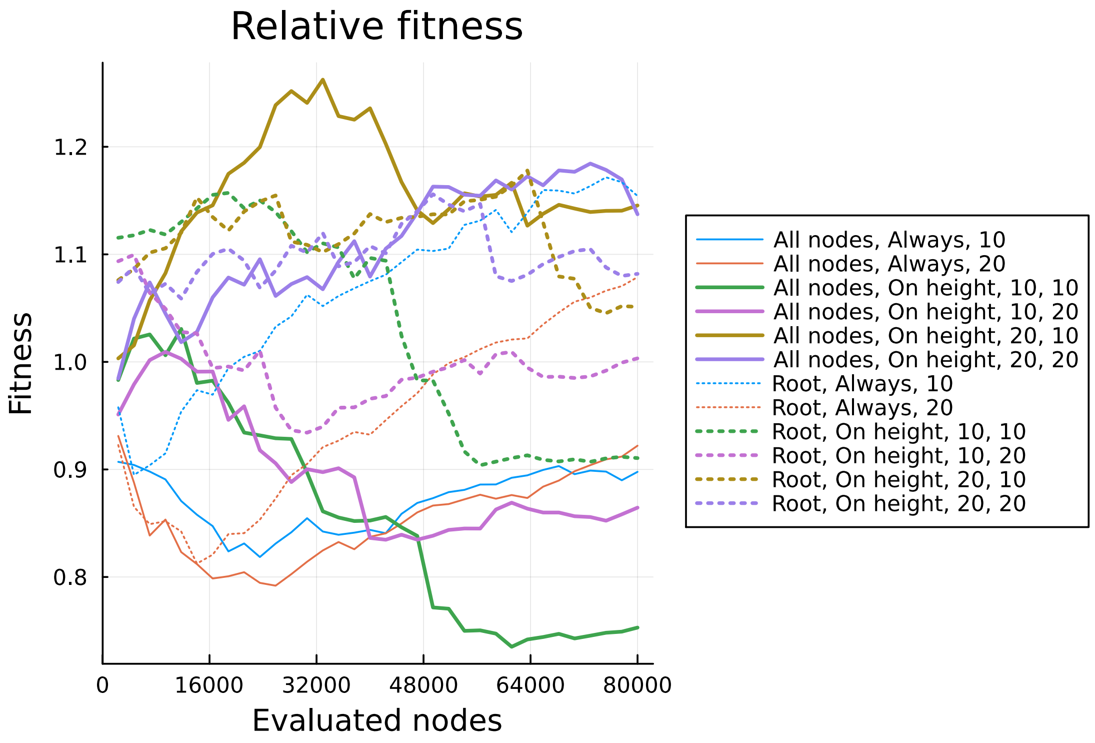


(b) : Development of fitness of tree individuals over time

Figure 7.2: Population control parameter tuning for MFO. Fitness shall be minimized. In the legend of each graph, the individual parameters are separated by a comma. The parameters are: Population size and random mating probability.

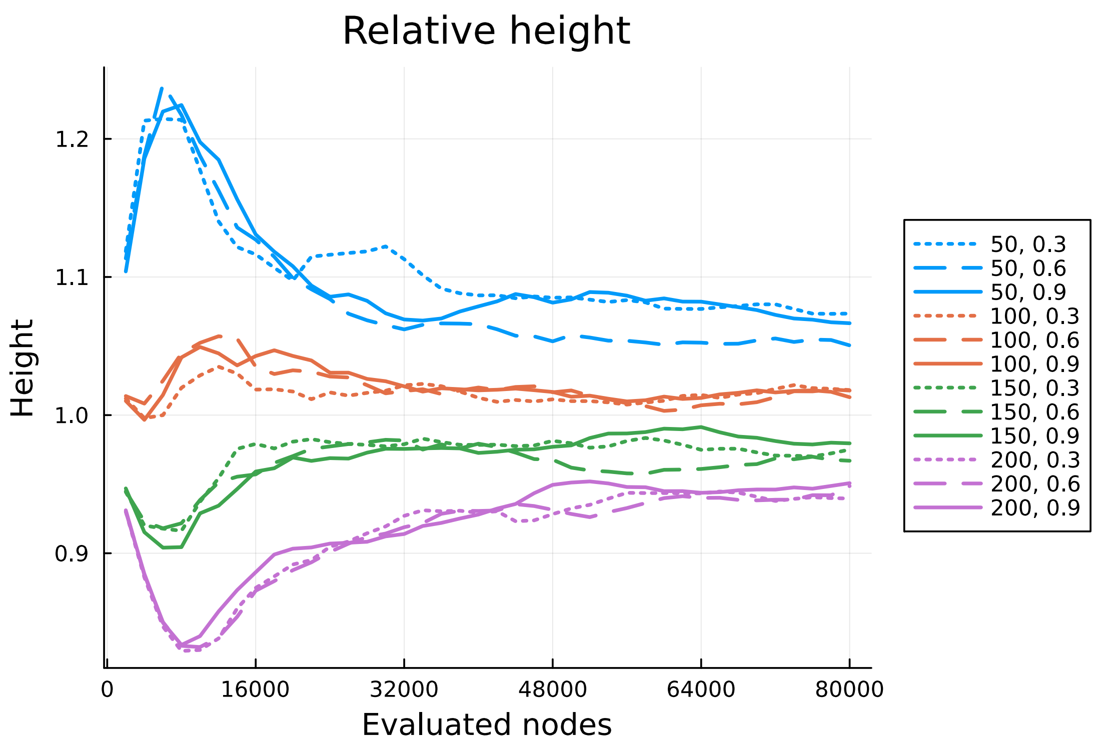


(a) : Development of height of tree individuals over time

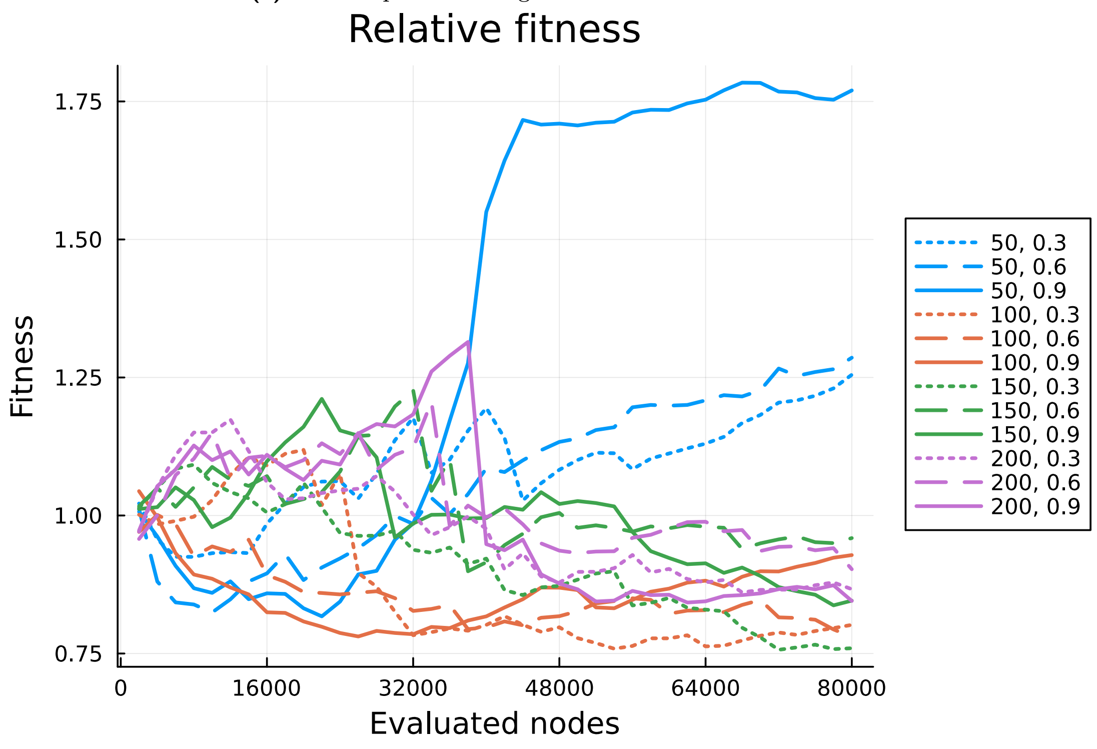


(b) : Development of fitness of tree individuals over time

Figure 7.3: Bloat control parameter tuning for SO. Fitness shall be minimized. In the legend of each graph, the individual parameters are separated by a comma. The parameters are: Evaluation, Local improvement, Height threshold, Height limit.



(a) : Development of height of tree individuals over time



(b) : Development of fitness of tree individuals over time

Figure 7.4: Population control parameter tuning for SO. Fitness shall be minimized. In the legend of each graph, the individual parameters are separated by a comma. The parameters are: Population size and random crossover probability.

Chapter 8

Experiments

This chapter presents the results of conducted experiments and the following analysis of the performance of the individual optimization approaches (MFO, SO) as well as their comparison. These experiments are focused on detailed analysis of the mentioned approaches, therefore no other alternative algorithm is included.

The experiments were conducted on testing problems introduced in Chapter 6 An the optimization algorithms had configuration settings showed in Section 7.3.

The goal of the conducted experiments is to find out whether in any of the stated cases of multitasking usage, mentioned in Chapter 1, the multifactorial optimization brings any improvement in terms of performance of optimization of particular tasks. Each case is analyzed in an individual section.

In the following sections, only a fitness graph is presented, since analyzing the development of the height of population has no relevancy, with respect to the performance comparison.

8.1 Optimizing one task with auxiliary tasks

This section analyses a case where only one task is needed to be optimized. Three distinct algorithms were launched in order to optimize the target task.

1. Single-objective approach that optimizes the target task only, this approach will be also referred to as *SO*.

2. Multifactorial approach with two different auxiliary tasks, this approach will be also referred to as *MFO*.

3. Multifactorial approach that has the target tasks in its assignment three times, this approach will be also referred to as *Multiple*. The reason for the Multiple approach is to find out whether the potential benefits of multifactorial optimization come from the intercommunication of

different tasks or if it is because of the structure of the population that the multifactorial optimization introduces.

All algorithms have a budget of 80000 evaluations and the results are averaged over 100 independent runs, as already discussed in the section 7.3

An important thing to keep in mind is that if each algorithm has the same evaluation budget, then the MFO approach with 3 different tasks in the assignment, will spend approximately only 1/3 of the evaluations for the target task. The rest of the budget is spent on auxiliary tasks that are not the goal of the optimization.

8.1.1 Koza

Based on Fig. 8.1, one can conclude that all the approaches have similar performance for the second and the third tasks, even though the multifactorial approach is slightly worse for these tasks. Quite interesting is the performance for the first task, the hardest one. Here, the multifactorial approach has a significant benefit over other approaches. A possible explanation is that for the hardest task, the multifactorial optimization could take advantage of inter-genetic communication and enhance the performance of the hardest task by partial solutions from the easier tasks.

8.1.2 Nguyen

For the Nguyen tasks the SO approach is the worst of all approaches, as depicted in Fig. 8.2. Even with the advantage of more evaluations per task, the SO approach is not dominant in either task. The multiple and multifactorial approaches have similar performance, even though the multifactorial approach is slightly better.

8.1.3 Keijzer

The Keizjer is clearly dominated by the SO approach. It has the best performance for all tasks. The worst is the multifactorial approach and a slightly better is the Multiple approach. This assignment has tasks of the same mathematical formulation, therefore it might be startling for someone, that the multifactorial approach has so poor performance.

One might assume, that the latent genetic complementarities should be intensive in this assignment and help the multifactorial approach to reach good performance. One possible explanation for the paradoxical observation might be, that the tasks are not solved completely correctly, but rather are approximated by another function. Even though the mathematical formulation is the same for all tasks, because of the different range of the tasks, their final form is quite distinct and therefore the approximating solution for each task might be also quite distinct, unfavorably to the multifactorial approach.

8.1.4 Keijzer2

For the Keijzer2 assignment, the performance of the Multiple and SO approaches is quite comparable. For the first task, the SO approach is better. For the second task, both approaches are of the same performance, and for the third task, the Multiple approach is better. The undoubtedly worst performance has the multifactorial approach, but since all the tasks in this assignment are mathematically distinct as well as spatially distinct, the observed data are in line with our preliminary suggestions.

8.1.5 Summary

Experiments of optimization of one task with auxiliary tasks were conducted and following analysis of their performance was discussed. Based on the results, the multifactorial approach tends to be dominant to single-objective approach in these cases:

- All tasks are symbolically and spaciously similar to each other.
- The task that shall be optimized is lunched together with auxiliary tasks, that are easier and share non-trivial amount of genetic material with the main task.

On the other hand, the multifactorial approach seem to not bring merit to optimization process in cases where

- The tasks are spaciously distinct and are too complex to be optimized precisely.
- All the task are symbolically and spaciously different.

8.2 Optimizing several tasks concurrently

This section analyses a case, where all tasks are the goal of the optimization. This section reveals a comparison of SO and MFO approaches for all assignments. Each task has the same evaluation budget, therefore the SO approach will have only a third of evaluations for a task in comparison with MFO since SO optimizes only 1 task at a time instead of 3 as in the MFO. In other words, for every 3 optimization processes of SO for different tasks of the assignment, 1 optimization process of MFO is made for all three tasks of the assignment and with a 3 times higher evaluation budget.

Each graph in this section contains 8 distinct lines. There are 3 lines for the MFO approach for the individual tasks. Then there is 1 line representing the compiled results of MFO performance for the whole assignment. In the compiled version, there are present fitnesses of all tasks in an interlacing manner and divided by the number of tasks. Then, there are 3 lines for the SO approach for individual tasks and also 1 line for the complied version of SO.

Due to the fact, that 8 lines per graph might be overwhelming, the lines are graphically clearly differentiated. The multifactorial approach is represented by solid lines, whereas the SO approach is presented by dashed lines. Due to the fact, that the SO approach has a smaller evaluation budget, its lines end early in the graph. For better graphical comparison, the SO lines are extended by dotted lines at the height of their best fitness for a given task. Each task it's own represented by a distinct color. The compiled versions of both approaches are then differentiated by greater thickness and unique color.

Here, as well as in the previous section, only graphs of fitnesses are presented, since the height is not relevant in performance comparison of individual approaches.

One can notice, that the line representing the compiled result of SO approach does not start at the beginning of the graph as other lines, but a bit later. It is because the compiled results need at least one iteration of the SO algorithm for all tasks, therefore it needs 3 times more evaluations to determine the fitness value of its first record.

Another unclear observation about the line representing the compiled version of SO is that its slope in some cases changes frequently. It is so because the compiled version was created by interlacing of results for individual tasks. Each task might have a different slope for any point in the graph. By interlacing these individual results, the slope of the compiled version is inherently also interlaced and thus might change.

8.2.1 Koza

As one can conclude based on the graph Fig. 8.5, the Multifactorial approach is systematically better for the Koza assignment than the SO approach. The MFO is better for all tasks and that is directly reflected in the compiled version of both approaches.

8.2.2 Nguyen

In the case of Nguyen the advantage of MFO is intensively pronounced. As depicted in Fig. 8.6 the MFO reached better results even during the run of SO. It means that the MFO got better results for a particular task even when splitting its evaluation budget between other tasks. This observation is in agreement with the results obtained for Nguyen task in the previous section. For the first task, both approaches managed to reach a good quality solution, even though MFO converged a bit faster. For the second and third tasks, the MFO revealed a great advantage towards a single-objective approach.

8.2.3 Keijzer

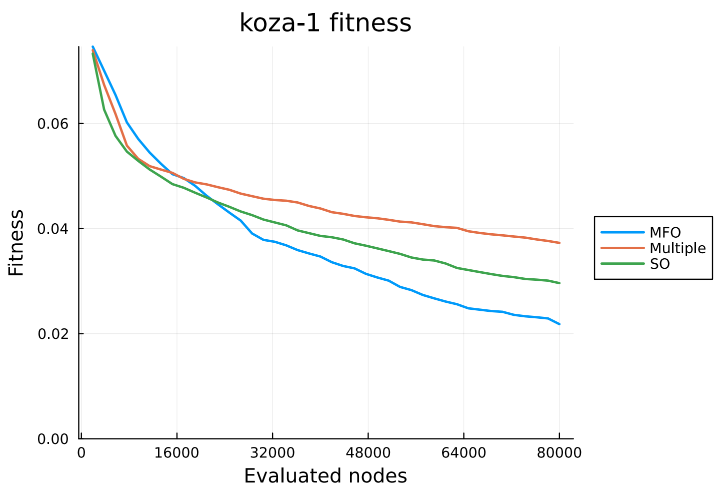
As one can conclude based on the graph Fig. 8.7, the Multifactorial approach is systematically better for the Keijzer assignment than the SO approach. For the second and third tasks, the Multifactorial approach has better performance, and even for the first task, the multifactorial approach is slightly better.

8.2.4 Keijzer2

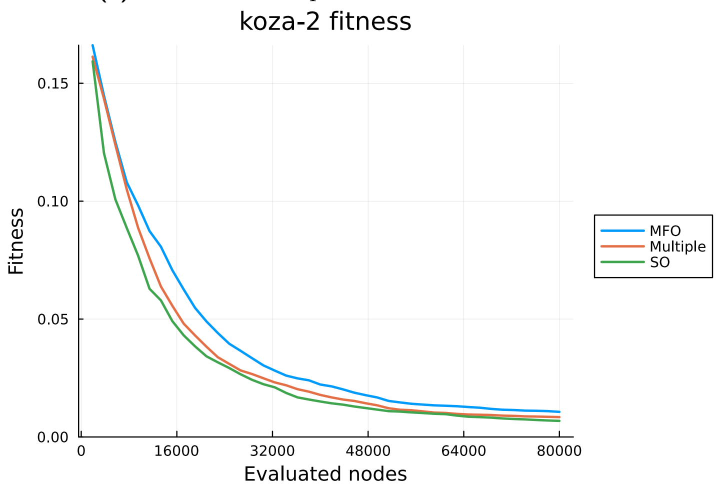
For the Keijzer2 assignment, the results for the first task are comparable, but the Fig. 8.8 shows a better results for the second and third task. It this case the MFO shows no a great advantage compare to the SO approach.

8.2.5 Summary

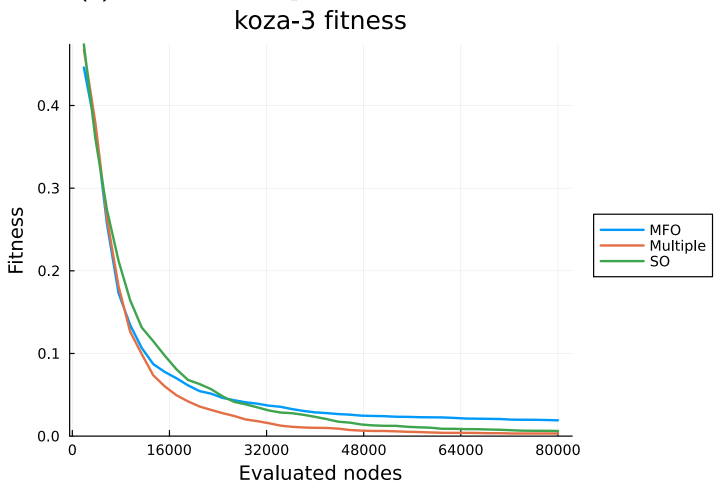
Experiments of optimization of several tasks concurrently were conducted and the following analysis of their performance was discussed. Based on the results, the multifactorial approach tends to have better performance in all tested cases.



(a) : Fitness development of the firs task of Koza

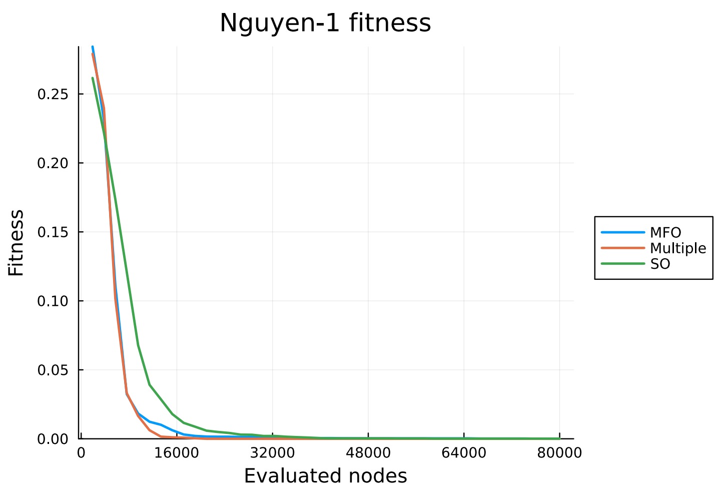


(b) : Fitness development of the second task of Koza

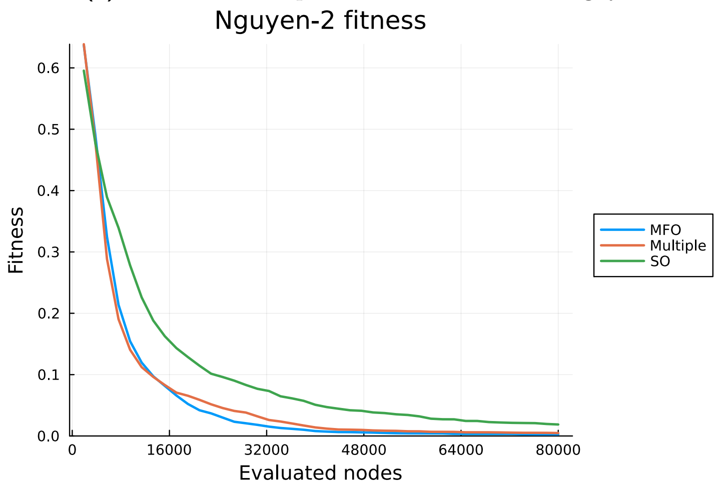


(c) : Fitness development of the third task of Koza

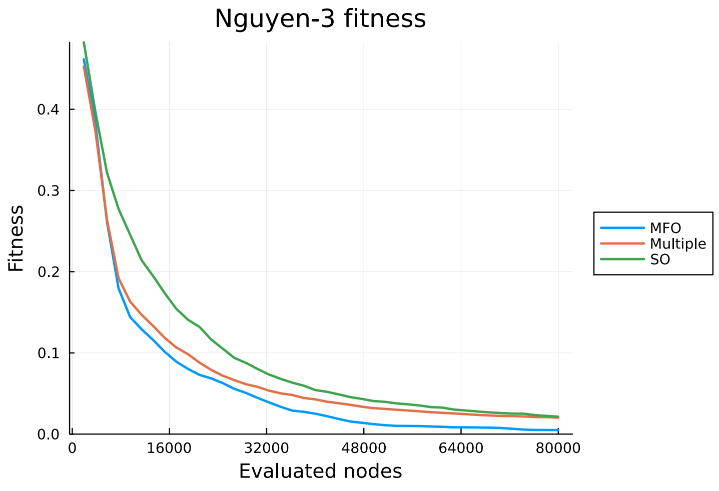
Figure 8.1: Comparison of MFO, Multiple and SO approach for the Koza assignment



(a) : Fitness development of the firs task of Nguyen

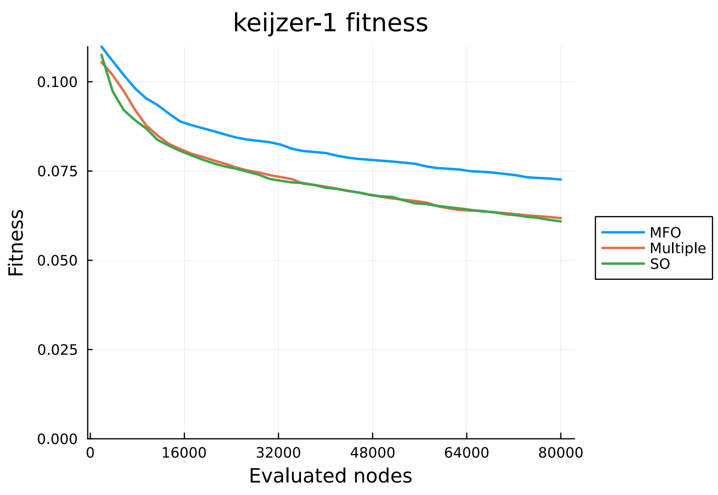


(b) : Fitness development of the second task of Nguyen

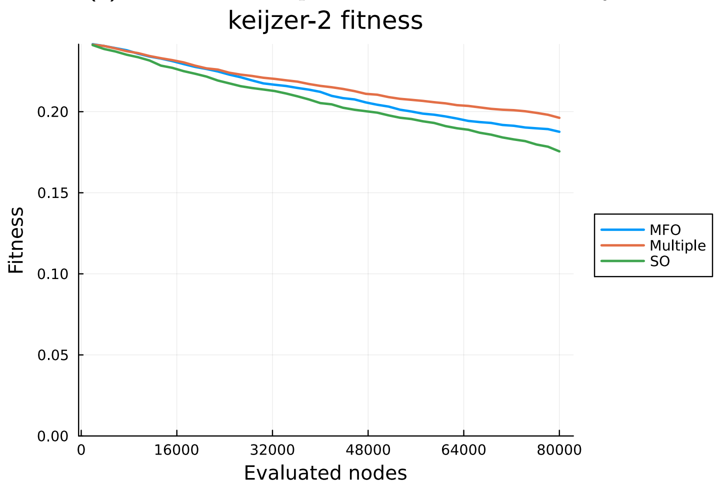


(c) : Fitness development of the third task of Nguyen

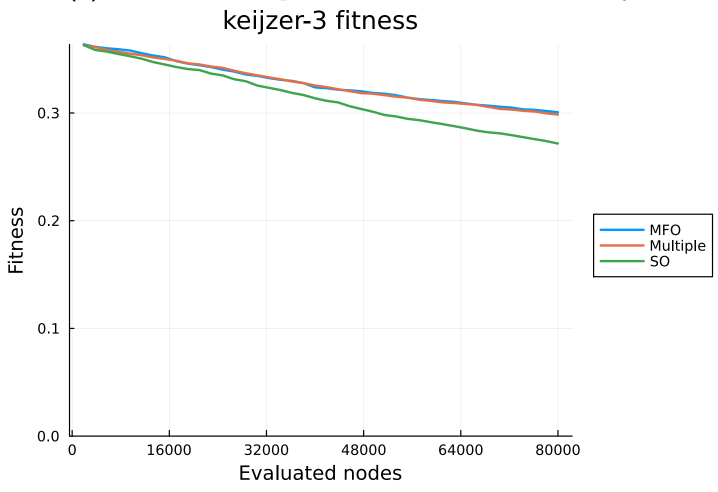
Figure 8.2: Comparison of MFO, Multiple and SO approach for the Nguyen assignment



(a) : Fitness development of the firs task of Keijzer

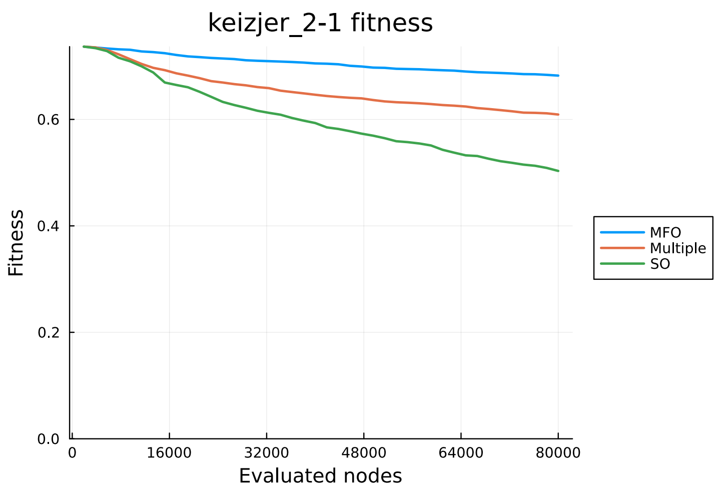


(b) : Fitness development of the second task of Keijzer

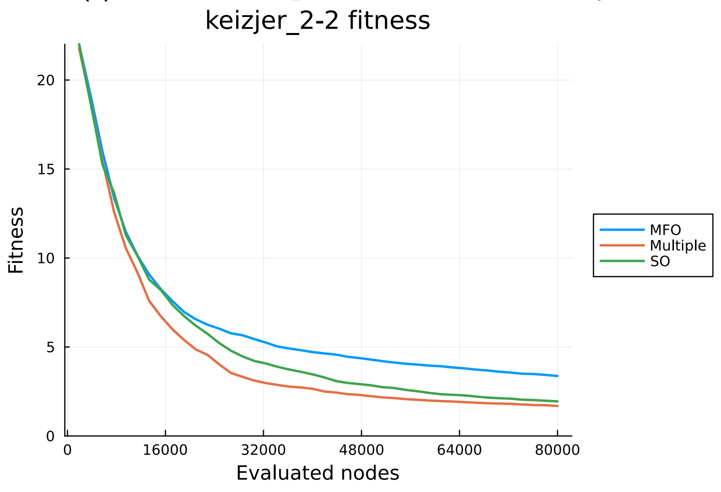


(c) : Fitness development of the third task of Keijzer

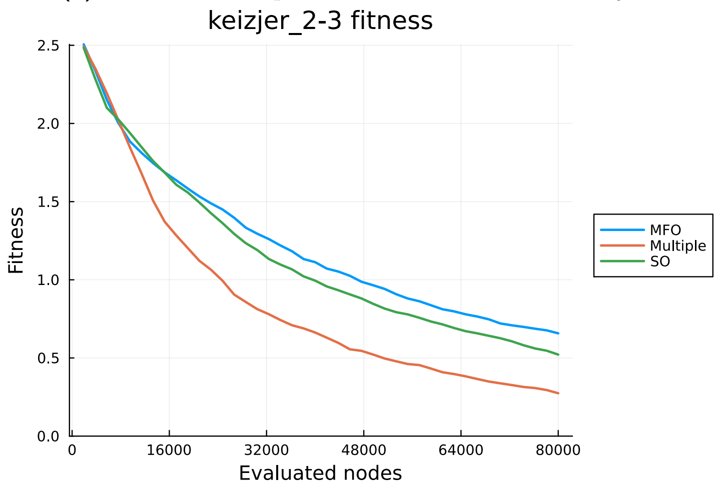
Figure 8.3: Comparison of MFO, Multiple and SO approach for the Keijzer assignment



(a) : Fitness development of the firs task of Keijzer2



(b) : Fitness development of the second task of Keijzer2



(c) : Fitness development of the third task of Keijzer2

Figure 8.4: Comparison of MFO, Multiple and SO approach for the Keijzer2 assignment

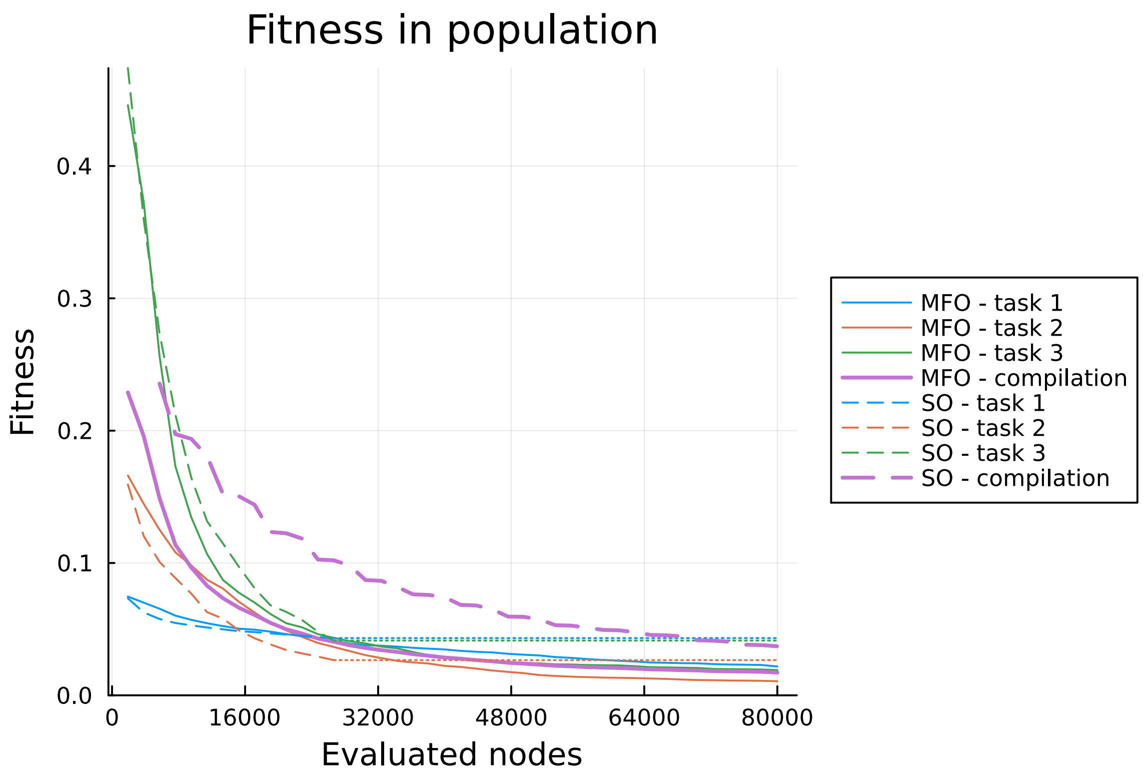


Figure 8.5: All tasks present in Koza assignment

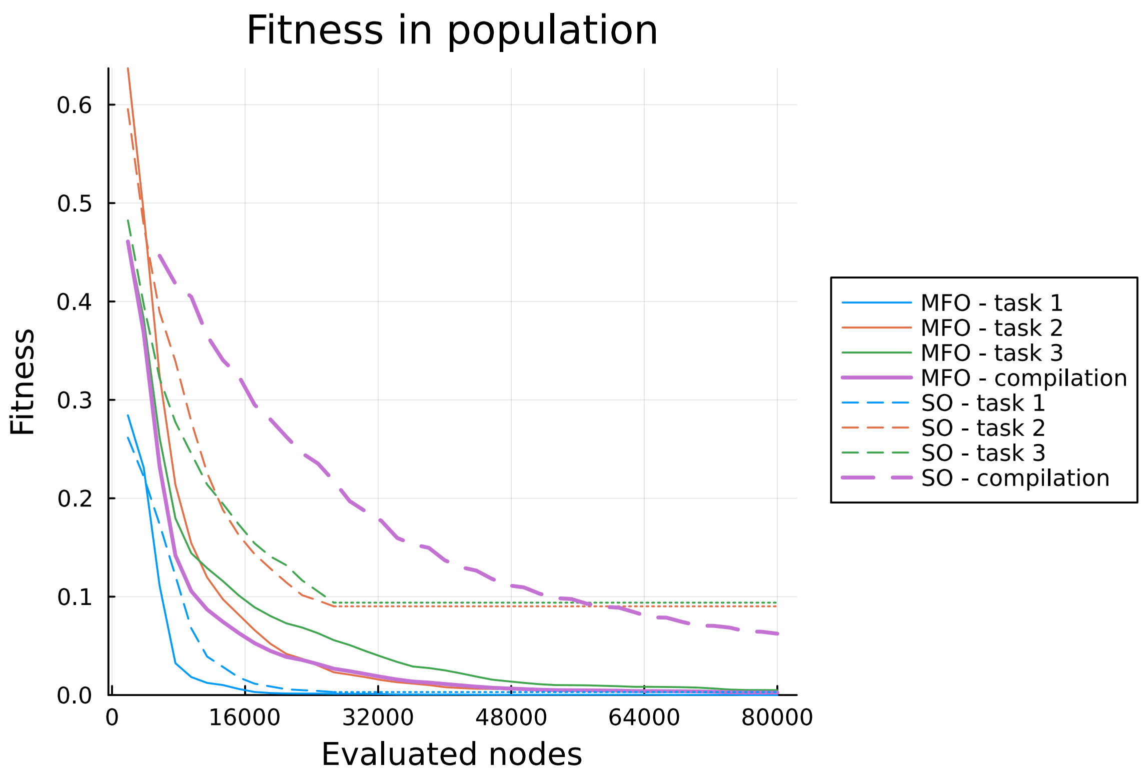


Figure 8.6: All tasks present in Nguyen assignment

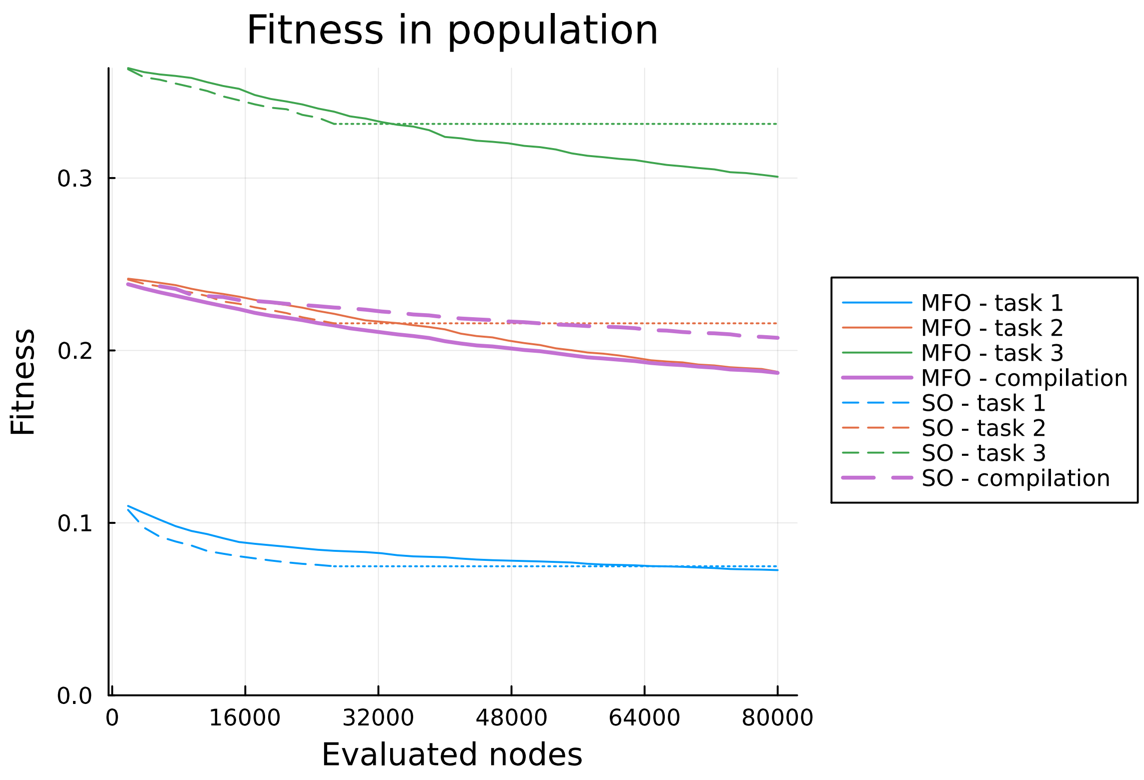


Figure 8.7: All tasks present in Keijzer assignment

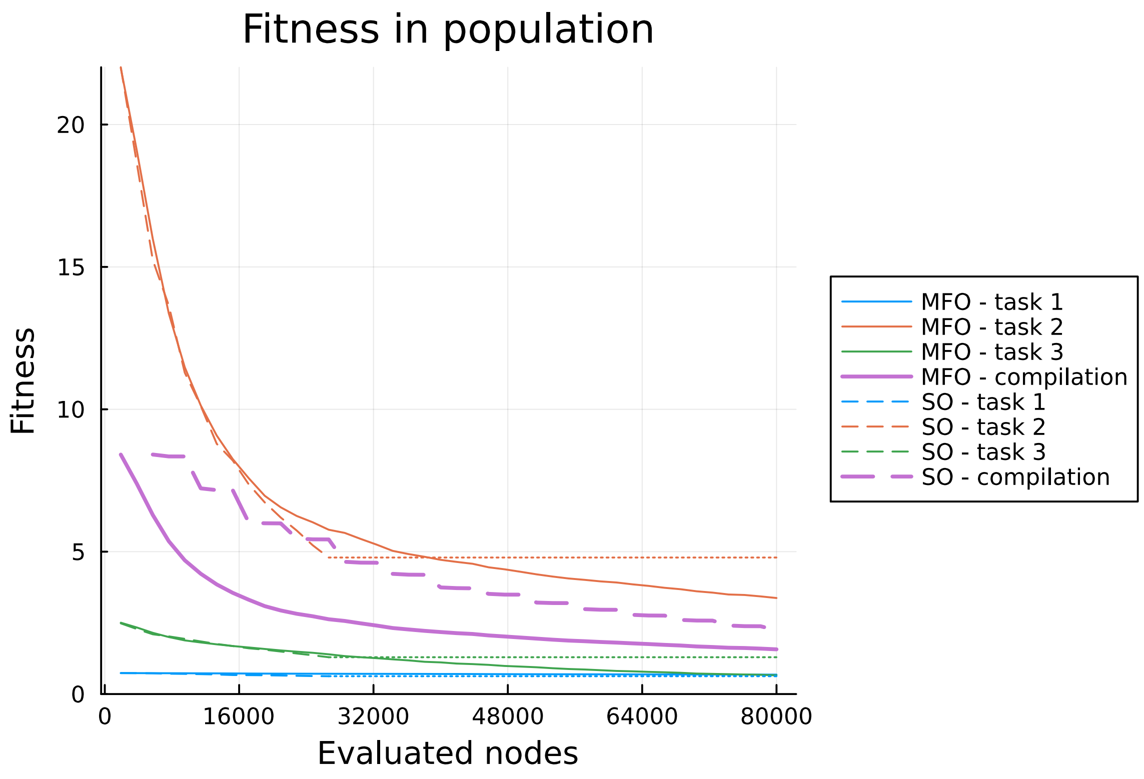


Figure 8.8: All tasks present in Keijzer2 assignment

Chapter 9
Conclusion

Evolutionary computation is a family of popular optimization techniques that are successful in many domains of optimization. Even though the development of the evolutionary algorithms was mainly focused on single objective cases, lately a new approach of multifactorial optimization has been introduced. It tries to achieve better efficiency by optimizing multiple optimization tasks concurrently and combining the genetic material of candidate solutions of different tasks. In this project, a genetic programming implementing multifactorial optimization was introduced. Several experiments were conducted in order to analyze the behavior of the multifactorial approach for genetic programming and compare its performance with the single-objective approach. These experiments were conducted two multiple testing problems and multiple scenarios. The results of experiments showed, that multifactorial optimization can be indeed beneficial and more effective than a single objective approach in a case where all multiple tasks are goals of the optimization and thus are optimized concurrently in a multifactorial sense. Despite the promising results, it is important to keep in mind, that a very limited set of settings and optimization problems were tested. Further experiments need to be conducted to get a better analysis of multifactorial optimization performance.

9.1 Future work

To get a more thorough analysis of the performance of the multifactorial approach, more testing problems need to be included. Especially real-world problems might give valuable insight into the possibilities of utilization of the multifactorial approach and multitasking in general.

Appendix A
Bibliography

[1] Wikipedia contributors. (2023a, December 9). Regression analysis. Wikipedia. https://en.wikipedia.org/wiki/Regression_analysis

[2] Lee, Chien-Chiang, Huang, Jikun, Charness, Gary B., Kotabe, Masaaki (2017) Mikehttps://www.sciencedirect.com/topics/economics-econometrics-and-finance/regression-analysis

[3] (2022) https://cw.fel.cvut.cz/b221/_media/courses/be4m33ssu/er_ws2022.pdf

[4] File:Genetic Program tree.png - Wikimedia Commons. (n.d.). https://commons.wikimedia.org/w/index.php?title=File:Genetic _Program_Tree.png&oldid=655982164

[5] Paulinas, M.; Ušinskas A. (2015) A survey of genetic algorithms applications for image enhancement and segmentation.

[6] Hassanat, A.B.; Prasath (2017) V.S.; Mseidein, K.I.; Al-Awadi, M.; Hammouri, A.M. A HybridWavelet-Shearlet Approach to Robust Digital ImageWatermarking.

[7] T. Back, U. Hammel, and H. P. Schwefel (1997) "Evolutionary computation: Comments on the history and current state," IEEE Trans. Evol. Comput., vol. 1, no. 1, pp. 3–17.

[8] Ashish, K.; Dasari, A. (2018) Chattopadhyay, S.; Hui, N.B. Genetic-neuro-fuzzy system for grading depression.

[9] Omisore, M.O.; Samuel, O.W.; Atajeromavwo, E.J. (2017) A Genetic-Neuro-Fuzzy inferential model for diagnosis of tuberculosis.

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

[11] 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.

[12] Mouret, J.-B. and Clune, J. (2015) “Illuminating search spaces by mapping elites,” arXiv [cs.AI].

[13] Dulava, T. (2021) Structured population in evolutionary algorithms [Bachelor’s Thesis, Czech technical university in Prague]. Dspace. http://hdl.handle.net/10467/96720

[14] Beyer, H. (2001). The Theory of evolution Strategies. In Natural computing series. https://doi.org/10.1007/978-3-662-04378-3

[15] Agapie, A. (2001). Theoretical Analysis of Mutation-Adaptive Evolutionary Algorithms. Evolutionary Computation, 9(2), 127–146. https://doi.org/10.1162/106365601750190370

[16] Jaśkowski, W., Krawiec, K., & Wieloch, B. (2007). Genetic programming for cross-task knowledge sharing. GECCO’07. https://doi.org/10.1145/1276958.1277281

[17] Scott, E. O., & De Jong, K. A. (2017). Multitask evolution with cartesian genetic programming. Proceedings of the Genetic and Evolutionary Computation Conference Companion. https://doi.org/10.1145/3067695.3075615

[18] Gupta, A., Ong, Y., Feng, L. (2016). Multifactorial evolution: toward evolutionary multitasking. IEEE Transactions on Evolutionary Computation, 20(3), 343–357. https://doi.org/10.1109/tevc.2015.2458037

[19] Zhong, J., Feng, L., Cai, W., Ong, Y. (2020). Multifactorial genetic programming for symbolic regression problems. IEEE Transactions on Systems, Man, and Cybernetics, 50(11), 4492–4505. https://doi.org/10.1109/tsmc.2018.2853719

[20] Wang, S., Jin, Y., & Cai, M. (2023). Enhancing the robustness of networks against multiple damage models using a multifactorial evolutionary algorithm. IEEE Transactions on Systems, Man, and Cybernetics, 53(7), 4176–4188. https://doi.org/10.1109/tsmc.2023.3241621

[21] Banzhaf, W., Langdon, W. B. (2002). Some Considerations on the Reason for Bloat. Genetic Programming and Evolvable Machines, 3(1), 81–91. https://doi.org/10.1023/a:1014548204452

[22] Poli, R. (2003). A simple but Theoretically-Motivated method to control bloat in genetic programming. In Lecture Notes in Computer Science (pp. 204–217). https://doi.org/10.1007/3-540-36599-0_19

[23] Skinner, C. D., Riddle, P., Triggs, C. M. (2005). Mathematics Prevents Bloat. IEEE. https://doi.org/10.1109/cec.2005.1554710

[24] Anuradha Purohit, Narendra S. Choudhari, ArunaTiwari (2018); Code Bloat Problem in Genetic Programming; Int J Sci Res Publ 3(4) (ISSN: 2250-3153). http://www.ijsrp.org/research-paper-0413.php?rp=P16986

[25] Nguyen, Q. U., & Chu, T. H. (2020). Semantic approximation for reducing code bloat in Genetic Programming. Swarm and Evolutionary Computation, 58, 100729. https://doi.org/10.1016/j.swevo.2020.100729

[26] Trujillo, L., Muñoz, L., Galván, E.,& Silva, S. (2016). neat Genetic Programming: Controlling bloat naturally. Information Sciences, 333, 21–43. https://doi.org/10.1016/j.ins.2015.11.010

[27] McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaśkowski, W., Krawiec, K., Harper, R., De Jong, K., O'Reilly, U. (2012). Genetic programming needs better benchmarks. GECCO. https://doi.org/10.1145/2330163.2330273

[28] Wikipedia contributors. (2023, December 20). Curse of dimensionality. Wikipedia. https://en.wikipedia.org/wiki/Curse_of_dimensionality