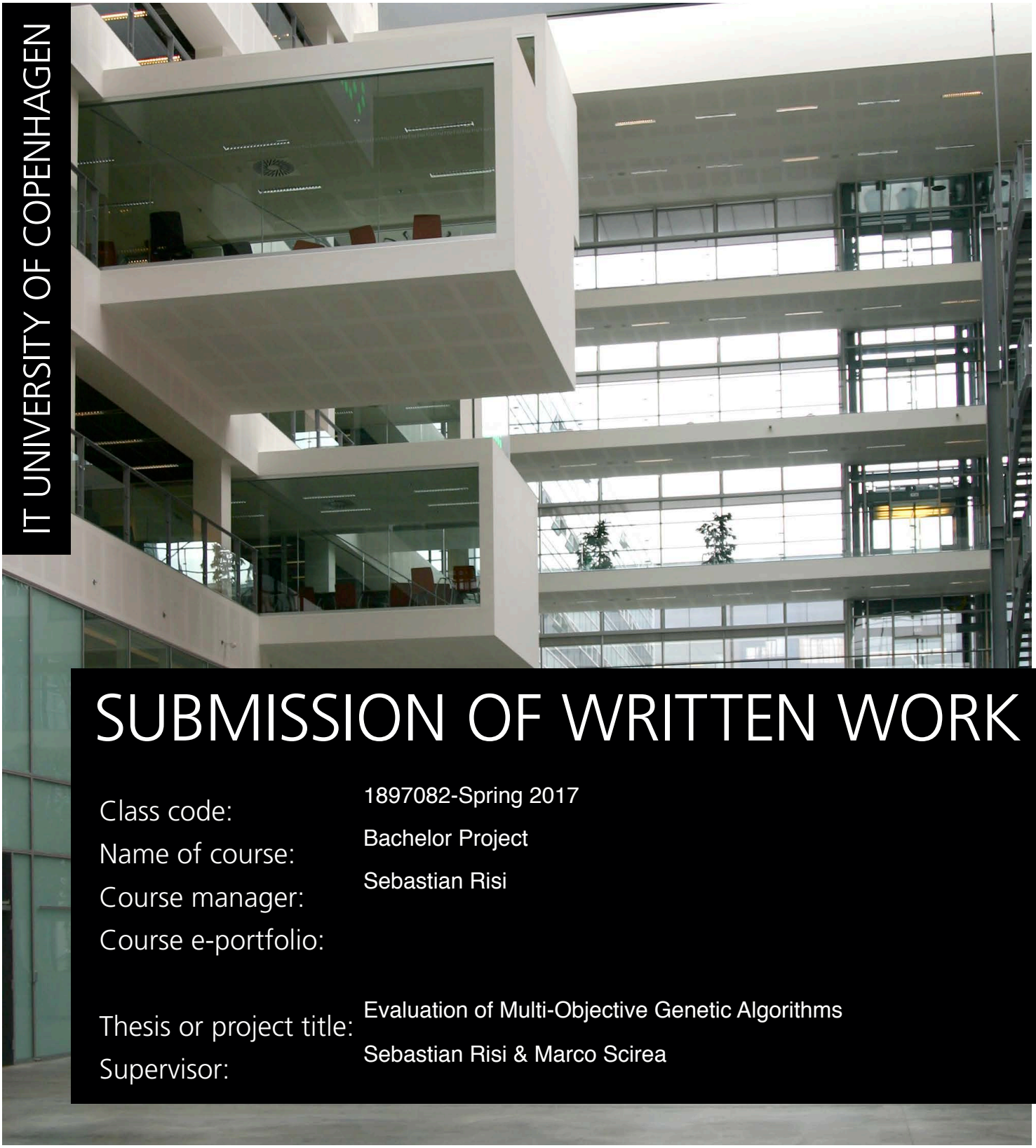


IT UNIVERSITY OF COPENHAGEN

SUBMISSION OF WRITTEN WORK

Class code:	1897082-Spring 2017
Name of course:	Bachelor Project
Course manager:	Sebastian Risi
Course e-portfolio:	
Thesis or project title:	Evaluation of Multi-Objective Genetic Algorithms
Supervisor:	Sebastian Risi & Marco Scirea

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Sebastian Benjamin Wrede	26/10-1995	sbwr@itu.dk
2.	Jon Voigt Tøttrup	03/06-1994	jvoi@itu.dk
3.	Kasper Hjort Berthelsen	22/03-1994	khjo@itu.dk
4.	Sebastian Baunsgaard	28/07-1992	sebab@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Evaluation of Multi-Objective Genetic Algorithms

A comparison of NSGA-II, NSFI-2Pop, Ray's algorithm & NSEA

BACHELOR PROJECT BY:

Jon Voigt Tøttrup

Kasper Hjort Berthelsen

Sebastian Baunsgaard

&

Sebastian Benjamin Wrede

IT UNIVERSITY OF COPENHAGEN

Abstract

In this project, three constrained multi-objective optimization genetic algorithms are compared with NSFI-2POP - an algorithm developed by Scirea, Togelius, Eklund, *et al.* The evaluation is based on seven constrained problems described in Deb, Pratap, and Meyarivan [2] and they are evaluated on three measures: average solution distance to the Pareto-optimal front, distribution of solutions and convergence rate. Three different experiments are run, each 10 times, to compare the algorithms with different parameters and the algorithms are also compared across problems. The results show that NSFI-2POP performs as good as the best of the three, while higher having variance in average distance to the Pareto-optimal front across different problems.

Contents

1 Introduction

The purpose of this project is to compare the performance of three constrained multi-objective optimization genetic algorithms with an algorithm developed by Scirea, Togelius, Eklund, *et al.* The project consists of:

- A comparison of the algorithms.
- Implementing the algorithms in the framework JMetal [3] [4] for Java.
- Apply the constrained test problems described in Deb, Pratap, and Meyarivan [2] to the algorithms.
- Measure and compare the performance by analysing the collected data.

Marco Scirea's algorithm is described in the paper:

- M. Scirea, J. Togelius, P. Eklund, *et al.*, "MetaCompose: A compositional evolutionary music composer", in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, Springer International Publishing, 2016, pp. 202–217. DOI: 10.1007/978-3-319-31008-4_14. [Online]. Available: https://doi.org/10.1007%2F978-3-319-31008-4_14

The four algorithms compared are described in the following papers:

- K. Deb, S. Agrawal, A. Pratap, *et al.*, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II", 2000
- N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms", pp. 221–248, 1994
- F. Jiménez and J. L. Verdegay, "Constrained multiobjective optimization by evolutionary algorithms", 1998, pp. 266–271
- T. Ray, K. Tai, and K. C. Seow, "Multiobjective design optimization by an evolutionary algorithm", pp. 399–424, 2000

The theory required to understand the field of study can be found in section 2. The algorithms are examined in section 3. The constrained test problems are described in section 2.10. Data collection is briefly touched upon in section 4. The results and analyses are in section 5. In section 6 the considerations implicit in the experiments and measures are examined, and potential further work is explored. Finally, the conclusion can be seen in section 7.

2 Theory

In this section, the theories and concepts used during the project will be presented. The descriptions mainly regard the theoretical aspects rather than specific implementations, but certain theories need interpretation to be applicable to this project and these interpretations are described as well. The algorithms are not presented in this section, but can be found in section 3. However, a concise description of the framework the algorithms were implemented in can be found at the end of this section.

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a group of algorithms that solve problems by generating a population of possible solutions and letting 'natural selection' cause a rise in fitness of the population. A solution can also be called an individual in evolutionary algorithms [9, p. 16].
In order to define fitness, all EAs require a function for determining the quality of a solution, referred to as a fitness or objective function [9, p. 15].
To simulate the process of evolution, EAs initially generate a random population of solutions. Then, by applying variation operators, see section 2.3, a new generation is created from the initial population.

In order to prompt an increase in quality, EAs also utilize a number of operators, see section 2.3.

2.2 Genotype & Phenotype

The genotype of a solution (or individual) is the encoding that determines what that solution 'is'. It is often compared to DNA in organisms. The genotype is usually represented as a vector of 'genes', which can be any type of data (such as integers or ASCII characters)[10, p. 2]. The phenotype is the result of applying the genotype to the objective functions, which returns a set of objective values [10, pp. 1-2].

2.3 Variation Operators

Variation operators modify individuals in the algorithms. This is almost always done in a stochastic manner, i.e. done based on random numbers or random modifications of one or more arguments. Variation operators can be classified in different groups based on what they use and/or modify [11] [12, p. 21] [9, p. 15-16]. The type of an operator is determined by the number of individuals needed for the operator. If the operator is on a single individual it is a mutation operator, see section 2.3.1. If the operator is on two individuals it is a crossover operator, see section 2.3.2. Operators involving more individuals are general operators, see section 2.3.3, this includes selection operators.

2.3.1 Mutation Operators

Mutation operators are performed on a single individual. These operators are used to maintain genetic diversity in the population by introducing new genomes. Through a mutation operation, a number of genes in the genotype of an individual are changed.

In evolutionary algorithms, a mutation operator usually has a percentage chance to change a genome, so that for each gene there is a chance that a mutation will occur. This is also known as a uniform mutation [7, p. 4].

2.3.2 Crossover Operators

A crossover operator is used to produce offspring based on two existing solutions. The idea is to create a new solution that has the characteristics of the parent solutions. Different crossovers produce different amounts of offspring[12, p. 21]. The following crossover operators are described in Michalewicz [12].

Single-point Crossover: A point is chosen in the genome. Before that only genes from one parent is copied to the offspring. After that, only from the other parent.

Two-point Crossover: The same as single point, just with two splitting points.

Random crossover: Also, known as *uniform crossover*. This crossover takes genes from either parent at random, at same locations in the genotype.

2.3.3 General Operators

The most common general operator is the selection operator. This is used to select individuals for crossover, mutation and/or other operators. The selection of individuals is usually very specific for the different algorithms.

2.3.4 Elitism

Elitism in genetic algorithms is the process of transferring some solutions from the population to a new generation without modification [13, p. 2]. The number of solutions transferred and how they are selected varies drastically from algorithm to algorithm (see differences in Jiménez and Verdegay [7, p. 4] and Deb, Agrawal, Pratap, *et al.* [5, p. 5]). But, in essence, the idea of elitism is the same in each algorithm; keep good solutions until better are found.

2.4 Problems

A **problem** in evolutionary algorithms is any challenge that can be solved. For instance, it could be the travelling salesman problem, cutting many shapes out of a metal plate, constructing a string containing as many '0's as possible or others. For each problem, there is several solutions.
An example solution to the problem of constructing a string of numbers containing '0's is "0123418250". It is easy to conclude that a more ideal solution would be "00000000" or maybe an infinite row of '0's.

An **optimization problem** is a problem that tries to minimize/maximize a value.
The example earlier is a maximization problem, a minimization problem would be to construct a string of numbers containing as few '0's as possible.

A **constrained problem** is a problem containing one or more constraint(s). All solutions for the given problem either do or do not violate the constraint(s). It is possible to violate one constraint without violating another.
Continuing the example from before, a constraint might be the length of the string, for example, saying the length must be shorter than 10 characters. Another constraint might be that a '0' may never come in pairs. Now we can define solutions as feasible and non-feasible. For instance, "000" is a non-feasible solution because it violates one of the constraints. The solution "010101010101" is also non-feasible because it violates the length constraint. A feasible solution would be "0130210".

A **multi-objective optimization problem** is an optimization problem with multiple objectives. This means that, in contrast to single-objective optimization problems, there may not exist a single solution, but rather a set of solutions [6].

Constructing a multi-objective problem from the example could be, in addition, have the string to be as short as possible. In this instance, an empty solution would be a good solution for one of the objectives and a long solution with a lot of '0's would be good for the other objective.

A **constrained multi-objective optimization problem** can be defined mathematically[7]:

$$f_i(x), i = 1, ..., p \tag{1}$$

$$g_j(x) \leq 0, j = 1, ..., m \tag{2}$$

Where the objective functions $f_i(x)$ are minimized, subjected to the constraints $g_j(x)$. p is the number of objectives, m is the number of constraints and x is an n-dimensional vector containing the genotype[7]. This means that it is a problem with any number of constraints, any number of objectives and any solution vector length.

The problems used to test the algorithms are multi-objective constrained minimization problems. Further details can be seen in section 2.10.

2.5 Constrained Multi-Objective Evolutionary Algorithms

A constrained multi-objective evolutionary algorithm is an evolutionary algorithm(see section 2.1) that tries to find a set of solutions to a constrained multi-objective optimization problem(see section 2.4). These are the types of algorithms examined in this project, which are described in section 3.

2.6 Pareto-Optimal Solutions

As mentioned in section 2.4, a multi-objective problem can have more than one solution. The set of solutions have one common characteristic and that is that they are superior when all objectives are considered, but may be inferior when looking at a single objective in isolation. The set of solutions are called *Pareto-optimal* or *non-dominated* [6, p. 1].

Said more precisely, given a minimization problem, a solution vector x_1 *dominates* another solution vector x_2 if $x_1 \prec x_2$, meaning that x_1 is partially less than x_2. A vector x_1 is partially less than x_2, if at least one value in x_1 is strictly less than the corresponding value in x_2, while all remaining values are equal[6, p. 3].

2.7 Constrained Multi-Objective Optimization Algorithms without Evolution

Constrained multi-objective optimization problems can be solved without using evolutionary algorithms. The issue when designing an algorithm for solving a problem with multiple objectives is that it is not always possible to achieve optimal solutions on all objectives simultaneously. Most algorithms that are not based on evolution solves this issue by reducing the problem to a scalar optimization problem. This means that instead of finding a set of non-dominated solutions, a set of scalars are defined and a solution to the problem is found by using the specified scalars on the objective functions. This reduces the

multi-objective problem to a single-objective problem. Common methods are: objective weighting, distance functions and min-max formulation. As a decision-maker, it is often preferable to find a set of good solutions and choose among those solutions instead of finding a single solution. This is why evolutionary algorithms are a viable option for multi-objective optimization[6].

2.8 Plotting Solutions and Objectives

The Pareto-optimal solutions for a constrained multi-objective problem with two objectives can be plotted in graphs, where the first axis represents objective f_1 and the second axis represents objective f_2. An example can be seen in fig. 1, where the Pareto-optimal solutions are the lowest values allowed within the constraints, which means that the Pareto-optimal solutions would be on the solid red line. The Pareto-optimal solutions can be grouped like in fig. 2, where the constraint line creates feasible and non-feasible areas so that five groups of solutions appear. The graphs are described in more detail in section 2.10.1.

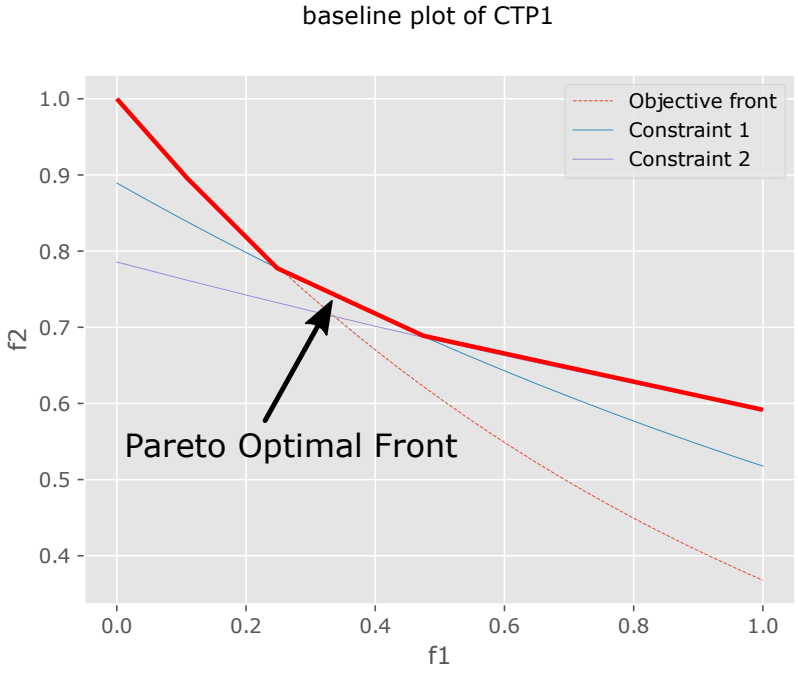


Figure 1: The Global Pareto-optimal front

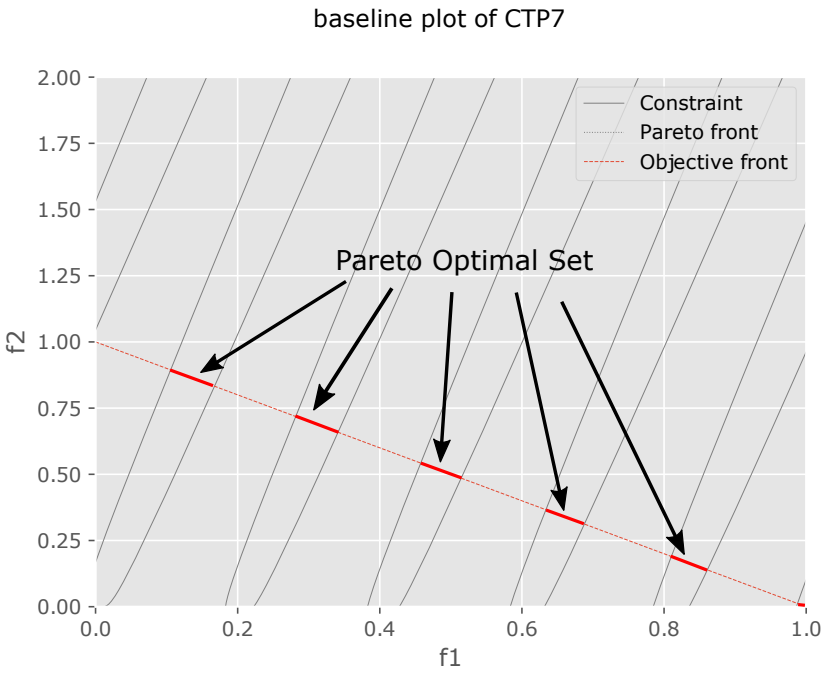


Figure 2: Pareto-optimal sets

2.9 Measures

When multi-objective genetic algorithms are evaluated, different measures for determining the performance of the algorithms can be utilized. Most research papers use diverse measures [2], [5]–[7], [14]. Because of this diversity, comparing algorithms across papers can be a challenge. A standardized procedure for evaluation would be simpler, but such a standard does not exist. This opens a range of possibilities for selection of measures in our project. The existing research projects within the area presents measures that could possibly prove applicable to our project. The most interesting measures will be described in the following sections.

2.9.1 Simple Objective Value Comparison

The simplest measure, which is used in most research papers[5]–[8]. A first non-dominated Pareto-front from a generation can be examined by looking at the phenotypes of the solutions, which in our case, is the objective values. The objective values can be examined either by looking at the plain values in a table, like in Jiménez and Verdegay [7], which can be quite confusing if the number of solutions are large. Instead, the objective values can be shown in a scatter plot, like in Deb, Agrawal, Pratap, *et al.* [5], Srinivas and Deb [6], and Ray, Tai, and Seow [8]. This should only be done if the number of objectives are three or less, but ideally only with two objectives, because the plot will be easier to grasp on a

piece of paper with only two dimensions.[1] A scatter plot of the objective values is a good solution if the data set is small and a quick overview of a generation is desired. However, the information contained in a scatter plot is not enough to describe the performance of an entire algorithm execution, because a plot of a single generation says nothing about the change of objective values throughout the execution. Additionally, the algorithms would be inaccurately evaluated if plotting was the only measure employed, because the distance and distribution of the solutions would be estimated only by the naked eye. The conclusion is that other measures are needed.

2.9.2 Distance from Global Pareto-Optimal Front

One of the most interesting observations of the results is the distance from the global Pareto-optimal front to the first non-dominated front of a generation. Indeed, this is mostly what is observed when looking at the plots described above, but a more precise measure could be developed for this purpose. A global Pareto-optimal front is the front that we strive to find with the genetic algorithms. It is the perfect solution(s) to the problems given within the constraints [5, p. 855]. The distance from the known global Pareto-optimal front can be used to describe the convergence of the solutions towards the perfect solution(s). In Deb, Agrawal, Pratap, *et al.* [5], the distance from global Pareto-optimal front to first non-dominated front is called the *convergence property* of an algorithm and is calculated by approximating the Pareto-optimal front as a combination of 500 piecewise linear segments and then calculate the average perpendicular distances from these segments. The convergence property in Deb, Agrawal, Pratap, *et al.* [5] has formed the basis of the Pareto distance measure used in this project (see section 5).

2.9.3 Distribution of Non-Dominated Pareto Front

When finding the optimum of a multi-objective constrained problem, we are not only interested in a single solution, but rather a set of solutions that together cover the entire Pareto-optimal front [5], [6]. In Srinivas and Deb [6], NSGA is evaluated using a measure based on a chi-square-like calculation [6, p. 13]. The aim for the measure is to determine whether the solutions are equally distributed among all sub-regions of the algorithm. If the distribution of solutions is perfect, each sub-region will have the same number of solutions. The performance measure in Srinivas and Deb [6, p. 17] can be seen in equation 3.

$$\iota = \sqrt{\sum_{i=1}^{q+1} \left(\frac{n_i - \overline{n}_i}{\sigma_i} \right)^2} \tag{3}$$

Where q is the number of desired optimal points, n_i is the actual number of individuals serving i-th sub-region of the non-dominated front, $\bar{n}_i$ is expected number of individuals serving the i-th sub-region of the non-dominated front, and ${\sigma_i}^2$ is the variance of individuals serving i-th sub-region of the non-dominated front.

[1] The number of dimensions can be greater, but the plot will quickly become difficult to comprehend when more dimensions are added. In our project, only two objectives are used.

The idea of a distribution measure is relevant for all multi-objective optimization algorithms, but it is not all algorithms that have their solutions divided into a fixed number of sub-regions, hence the performance measure for NSGA cannot be used when comparing different algorithms.

In Deb, Agrawal, Pratap, *et al.* [5] about NSGA-II, another distribution measure has been utilized. NSGA-II does not have a fixed set of sub-regions, hence it cannot use a chi-square-like measure to determine the distribution of solutions over the sub-regions. Instead, an equation based on Euclidean distance among the solutions in the first non-dominated front has been applied. The equation used in the paper can be seen in eq. (4).

$$\Delta = \sum_{i=1}^{|F_1|} \frac{\mid d_i - \overline{d} \mid}{\mid F_1 \mid} \tag{4}$$

Here, d_i is the Euclidean distance between two solutions in the first non-dominated front F_1. $\mid F_1 \mid$ is the number of individuals and $\bar{d}$ is the average Euclidean distance between solutions in this front. That the measure in eq. (4) does not make use of any specific features of the algorithm makes it ideal for general comparison of multi-objective optimization algorithms.

2.9.4 Convergence Time

The distance from the global Pareto-optimal front and the distribution of the first non-dominated front could be called *static* measures, because they are good measures when describing the situation of the solutions in a given generation. The two measures are, however, insufficient when describing the convergence towards an acceptable solution during execution of an algorithm. To describe the development of the first non-dominated front, inspiration could be found in Srinivas and Deb [6] where the performance measure of each generation is plotted with generation number. In this way, an overview of the development can be seen by looking at the plot, but since this relies on interpretation, it becomes more inaccurate (see section 5.1.3).

The performance measure of NSGA in Srinivas and Deb [6] is algorithm-specific, which means that it cannot be used as a general measure for comparison between algorithms. The reason behind this is the same as described in section 2.9.3. This measure is a distribution measure, which means that it only conveys the distribution of the solutions. In Deb, Agrawal, Pratap, *et al.* [5] and in this project, two separate measures describe different aspects of the algorithm. When plotting the convergence of the algorithms, only a single performance measure is used, hence the relative importance of the two measures should be decided. In Srinivas and Deb [6], the distribution measure is deemed most important, but the distance from the global Pareto-optimal front is also important when looking at the development of the solutions even though that measure has been left out in Srinivas and Deb [6]. In this project, both measures are used.

2.10 Constrained Test Problems

To test the different algorithms systematically the constrained test problems(CTP) from Deb, Pratap, and Meyarivan [2] have been used. The test problems pro-

vide different degrees of difficulty for multi-objective algorithms depending on different variables. These problems can be divided into two groups: CTP 1 and CTP 2-7. All seven CTPs have similar characteristics, but test different aspects of the algorithms.

2.10.1 Common Characteristics

All Constrained problems uses a function called $g(x)$, which Deb, Pratap, and Meyarivan [2, p. 289] describes:

> (...) the function $g(x)$ can be a complex multi-variable function.

In the tests done in this project g(x) is defined as in eq. (5).

$$g(X) = \frac{x_1 + x_2 + ... + x_n}{n - 1} + 1 \tag{5}$$

Where X is the genotype and can be of variable length and x_1 to x_n are all the elements of X except the x_0 element, which is the first element of the genotype. The function $g(x)$ is the average of all the genotype-values except the first. This implementation of $g(x)$ has been chosen because it makes use of every element of X, except x_0. Furthermore, it is possible to use the same function when additional elements are added to X, which means that it is a solution that can be generalized to different implementations of the solution genotype.

2.10.2 CTP 1

The Standard form of CTP 1 consists of an objective and two constraints[2]. The Design of this problem allows for an infinite amount of constraints, but in this instance the default problem is used. The amount of constraints is defined as J.

$$f_1(X) = x_1 \tag{6}$$

$$f_2(X) = g(X) \cdot e\left(\frac{-f_1(X)}{g(X)}\right) \tag{7}$$

$$c_j(X) \equiv f_2(X) - a_j \cdot e(-b_j \cdot f_j(X)) \geq 0, j = 1, 2, ..., J \tag{8}$$

There are multiple steps involved in calculating the a_j and b_j. These can be found in the paper [2]. For the $J = 2$ the values are as shown in table 1.

j	a_j	b_j
1	0.8894	0.541169
2	0.7857	0.283167

Table 1: CTP1 a_j and b_j values

2.10.3 CTP 2 through 7

The objective front of CTP 2-7 is identical, therefore objective f_1 and f_2 are the same for all of them. The constraint is what sets CTP 2-7 apart from each other. The objectives can be seen in eq. (9) and eq. (10) and the general formula for the constraint can be seen in eq. (11).

$$f_1(X) = x_1 \tag{9}$$
$$f_2(X) = g(X) - f_1(X) \tag{10}$$
$$c(X) \equiv \cos(\theta)(f_2(X) - e) - \sin(\theta) \cdot f_1(X) \geq a|\sin(b\pi(\sin(\theta)(f_2(X) - e) + \cos(\theta)f_2(X))^c)|^d \tag{11}$$

The difference between CTP 2-7 is the values of a, b, c, d, e and θ. The different values are shown in table 2.

CTP	**a**	**b**	**c**	**d**	**e**	θ
2	0.2	10	1	6	1	-0.2π
3	0.1	10	1	0.5	1	-0.2π
4	0.75	10	1	0.5	1	-0.2π
5	0.1	10	2	0.5	1	-0.2π
6	40	0.5	1	2	-2	0.1π
7	40	5	1	6	0	-0.05π

Table 2: The Different Values for CTP 2-7

2.10.4 Plotting Lines

Some lines are constructed using implicit functions, others using normal functions. The functions are deduced from the formulas described in Deb, Pratap, and Meyarivan [2].

The functions used for plotting CTP1 are in eq. (12).

$$\begin{aligned} \text{Objective Front:} &\quad y = e(-x) \\ \text{Constraint 1:} &\quad y = 0.8894 \cdot e(-0.541169x) \\ \text{Constraint 2:} &\quad y = 0.7857 \cdot e(-0.283167x) \end{aligned} \tag{12}$$

CTP 2-5 is plotted using the functions in eq. (13). The objective front function in eq. (13) is deduced from eq. (5), eq. (9) and eq. (10) using reduction and switching the $g(x)$ function with 1 that is the smallest result it can return.

$$\begin{aligned} \text{Objective Front:} &\quad y = -x + 1 \\ \text{Constraint:} &\quad 0 = a \cdot |\sin(b\pi(\sin(\theta) \cdot (y - e) + \cos(\theta) \cdot x)^c)|^d \\ &\quad - (\cos(\theta) \cdot (y - e) - (\sin(\theta) \cdot x)) \\ \text{CTP 2-5 Pareto Front:} &\quad 0 = x \cdot \sin(\theta) - y \cdot \cos\theta + e \cdot \cos\theta \end{aligned} \tag{13}$$

The Pareto-optimal front of CTP 6 is based on a complex oscillating curve and therefore it is difficult to make a function that fits perfectly. Instead of this function, the objective front of CTP 7 has been used. In CTP 7, the objective front is the same as the Pareto-optimal front.

2.10.5 CTP Challenges

The CTPs provide different challenges to the algorithms. Below, the main challenges of each algorithm are described.

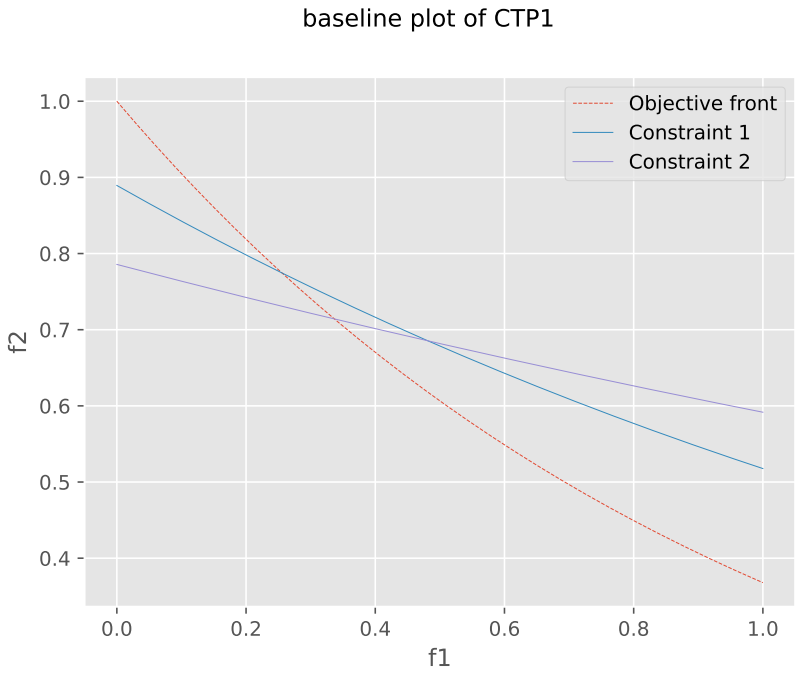


Figure 3: Empty CTP 1 plot with lines

CTP1 The difficulty in this problem is to find solutions on a non-linear constraint boundary. The number of constraints could be increased, but the two constraints chosen in this project are enough to construct a piecewise global Pareto-optimal front, meaning that solutions not only have to deal with the objective front, but also differing constraints which have increasing influence to the right. The problem is plotted in fig. 3.

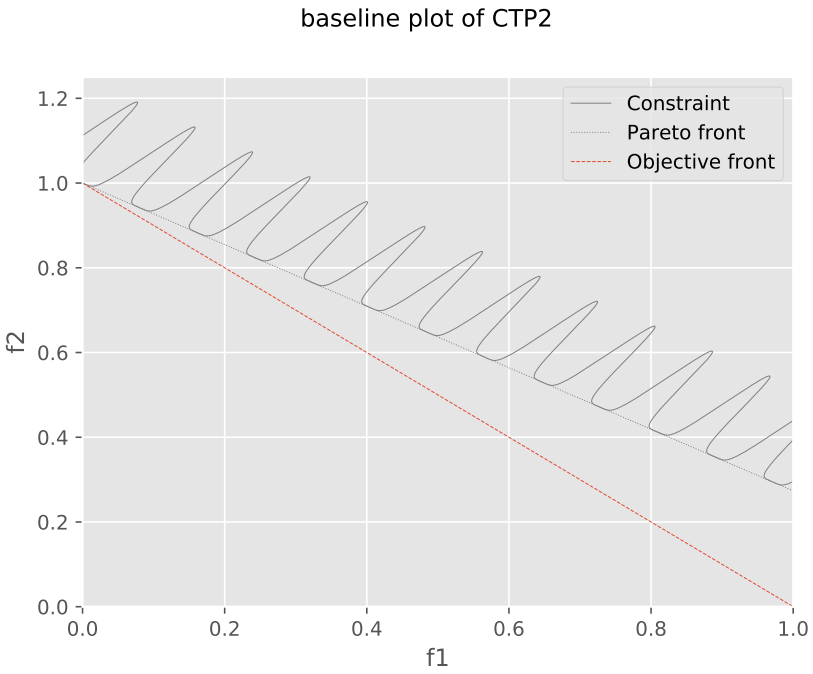


Figure 4: Empty CTP 2 plot with lines

CTP2 The difficulty of this problem has some things in common with the rest of the problems. The constraint is based on sine and cosine, which means that the constraint boundary has a wave-like shape. The global Pareto-optimal front consists of several disconnected regions, because the wave-like constraint boundary touches the Pareto-optimal front several times. The challenge for the algorithms is then to find solutions that are placed in every disconnected region of the global Pareto-optimal front. The problem can be seen in fig. 4. The number of disconnected regions can be increased by increasing the parameter b.

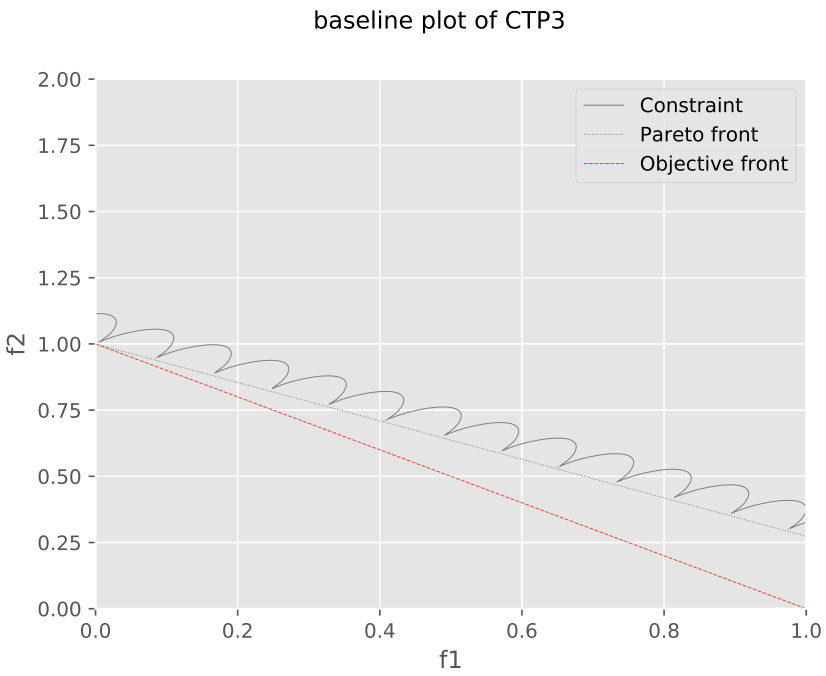


Figure 5: Empty CTP 3 plot with lines

CTP3 This problem is much like CTP2, but the disconnected regions are smaller, which means that there is only space for a single solution in each disconnected region. This is done by decreasing the value of d and a. When there is only space for a single solution in each disconnected region, the difficulty is greater because as the solutions approach the global Pareto-optimal front, the feasible search space diminishes. CTP3 can be seen in fig. 5.

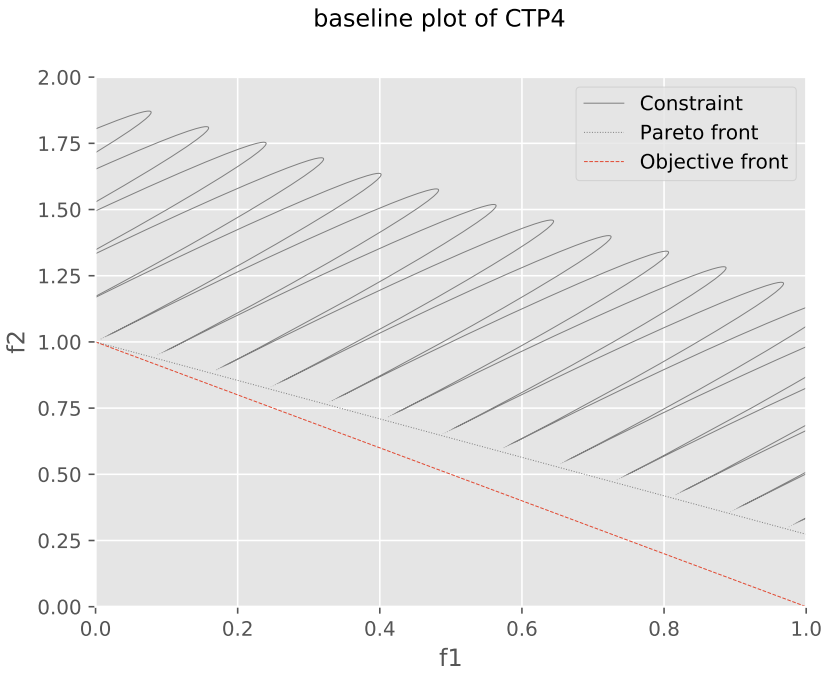


Figure 6: Empty CTP 4 plot with lines

CTP4 By increasing the value of parameter a compared to the value in CTP3, the transition between the continuous feasible region to the discontinuous region is done far from the global Pareto-optimal front. This shows as a series of narrow "tunnels", which can be seen in fig. 6. The difficulty in finding the global Pareto-optimal region in this problem is due to the distance the solutions must travel through the separate tunnels. When genes are shared among individuals in different tunnels, the offspring could become infeasible. This could potentially limit the evolutionary process so severely that the algorithm never finds a perfect solution.

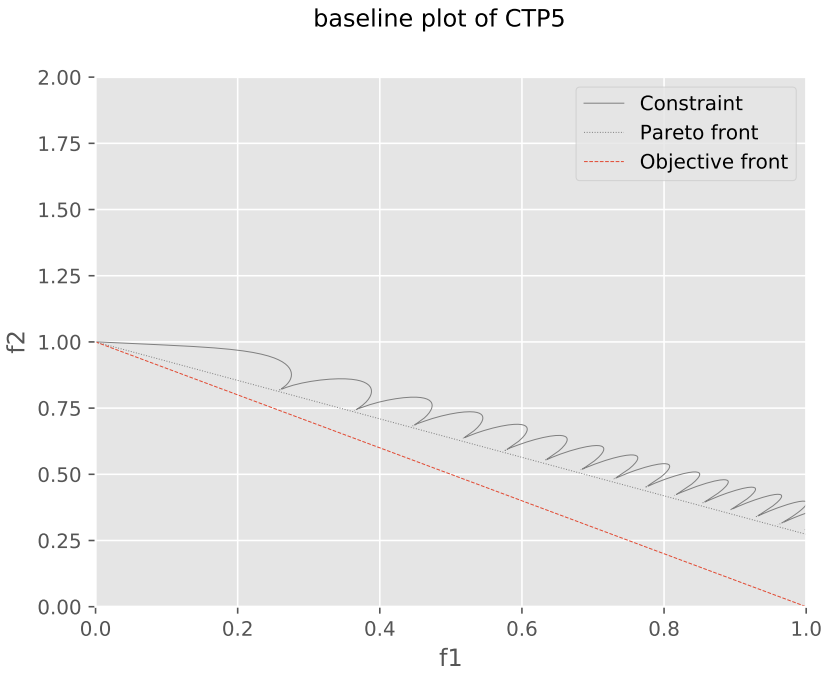


Figure 7: Empty CTP 5 plot with lines

CTP5 The difficulty in this problem is that the feasible region "tunnels" are closer in one side of the plot compared to the other. This is done by changing the value of the parameter c so that $c \neq 1$. This test the flexibility of the algorithms, since the algorithms must find solutions that are far apart, but also solutions that are closely packed[2, p. 298]. The algorithms must maintain diversity even though the majority of the solutions are near each other. The plot of CTP5 can be seen in fig. 7.

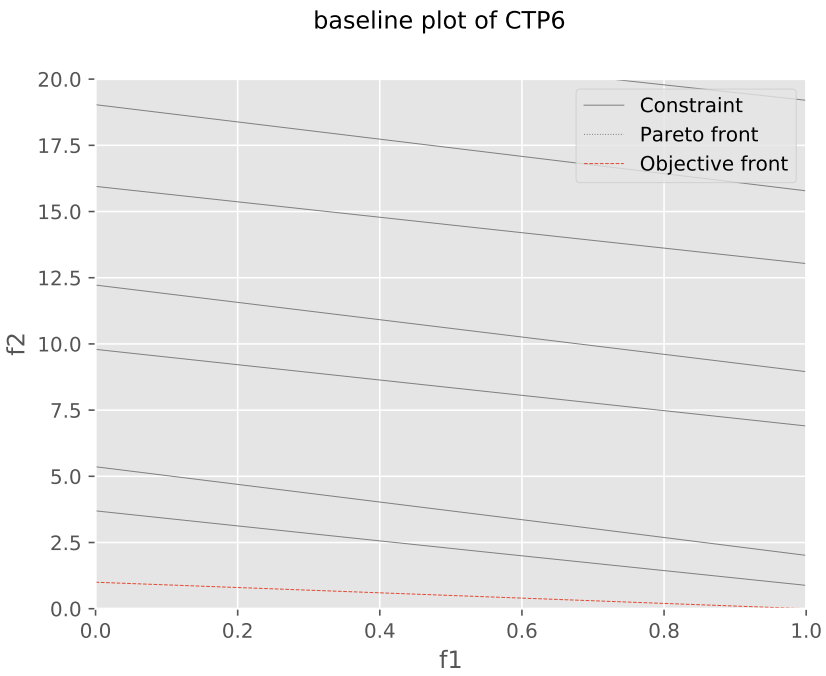


Figure 8: Empty CTP 6 plot with lines

CTP6 This problem is a bit different from the previous problems although it is based on the same formulas as CTP2-5. The difficulty for the algorithms is to cross the infeasible regions to converge towards the global Pareto-optimal front. Furthermore, the global Pareto-optimal front is placed on a constraint boundary, which contrasts with CTP2-5. The problem is plotted in fig. 8, where some of the space between the lines are infeasible regions.

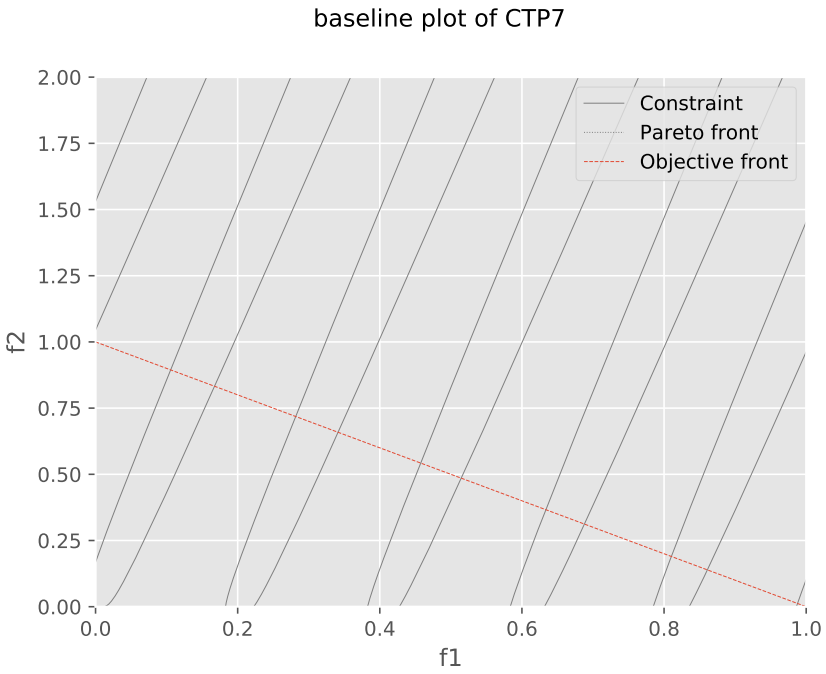


Figure 9: Empty CTP 7 plot with lines

CTP7 The difficulty of this problem is to maintain diversity in the population from the beginning and avoid the infeasible regions along the way to the global Pareto-optimal front. The problem can be seen in fig. 9.

2.11 JMetal: A Framework for Multi-Objective Optimization with Metaheuristics

JMetal [3] [4] is the chosen framework in this project. It is a Java Framework for developing and running multi-objective genetic algorithms, and comes preconfigured with several problems and algorithms, including NSGA-II, as described in section 3.1. It allows for easy development by providing a fixed but open design, that abstracts away from lower level details. JMetal provides four main interfaces[15]:

Solution interface: Represent solutions/individuals of the problems.

Operator interface: Represents the interface for the different operators, for instance: crossover operators, mutation operators and selection operators (see section 2.3). The operator interface can be extended with the specialized types of operators and reused in different algorithms. Several operators has been developed during this project by implementing JMetal interfaces, but the operators will not be described in this section. See section 3 for details about the operators.

Problem interface: The problem interface defines the problems presented to the algorithms (see section 2.4), and is what has been used to implement the CTP's.

Algorithm interface: Algorithm interface is the interface implemented by all algorithms. It contains a method to run the algorithm and a method that gets the result, which means that it returns a list of all the non-dominated solutions in the population. An example of an implementation of the run method can be seen in listing 1.

```
66  @Override public void run() {
67          List<S> offspringPopulation;
68          List<S> matingPopulation;
69
70          population = createInitialPopulation();
71          population = evaluatePopulation(population);
72          initProgress();
73          while (!isStoppingConditionReached()) {
74                  matingPopulation = selection(population);
75                  offspringPopulation = reproduction(matingPopulation);
76                  offspringPopulation = evaluatePopulation(offspringPopulation);
77                  population = replacement(population, offspringPopulation);
78                  updateProgress();
79          }
80  }
```

Listing 1: A snippet of the run function in the JMetal source: AbstractEvolutionaryAlgorithm.java [16]

2.12 Summary

This section introduces the concept of *evolutionary algorithms*, algorithms that apply natural selection to find solutions. Also introduced are:

Variation Operators that are applied by the algorithm to change solutions.

Objectives that are used to evaluate the fitness of the solutions.

Constraints that must be fulfilled for a solution to be feasible.

Pareto-optimal Solutions that are the optimal compromise between the different objectives and constraints.

Constrained Test Problems (CTP's) which are used to test the algorithms' performance.

JMetal that is the framework used in this project.

3 The Algorithms

In this section, the algorithms to be evaluated and compared are examined and accounted for. Any reservations in regard to the implementation of the algorithms is also touched upon.

3.1 NSGA-II: Non-dominated Sorting Genetic Algorithm for Multi-Objective Optimization

NSGA-II, as described in Deb, Agrawal, Pratap, *et al.* [5], is widely used and has been implemented in many programming languages, such as R[17], Python[18] and Java[19]. NSGA-II is an enhancement of the original NSGA algorithm described in Srinivas and Deb [6]. NSGA was one of the first algorithms of its kind, but it had several weaknesses, which Deb, Agrawal, Pratap, *et al.* tries to address. The three points of criticism in the paper are:

High Computational Complexity: NSGA has $O(MN^3)$

No elitism: NSGA has no elitism, which means that there is a potential risk of evolution moving the wrong direction.

Uses sharing parameter: NSGA requires the use of a sharing parameter. Sharing as a concept is used to ensure a diverse set of non-dominated solutions. In NSGA, this is implemented by having a solution's presence influence the fitness values of the solutions around it. The sharing parameter defines the maximum distance a solution can have to another solution to become member of the same niche. A greater number of solutions in a niche leads to lower fitness values of solutions in the niche[6, Section 4.2].

3.1.1 Reducing Complexity

Before NSGA performs selection of individuals, it sorts them into several fronts. The first front being the non-dominated individuals, the second being the non-dominated individuals if we assume the first front does not exist and so on. To do this, NSGA compares every individual with every other individual for every front, thus giving a complexity of $O(N^3)$ where N is population size. Since there are multiple objectives, the comparison must be done for every objective M giving a complexity of $O(MN^3)$[6, Section 4]. NSGA-II reduces the complexity of the original NSGA algorithm by implementing another strategy for ranking the individuals. NSGA-II tracks all non-dominated solutions in a set S as they are discovered during the ranking. The ranking is performed by comparing solutions to the solutions contained in S. If a solution a dominates a solution b that is currently in S, a takes b's place. Every new solution introduced to S is then compared with the rest of S so that any solutions in S dominated by the new solution are removed. This has a complexity of $O(N^2)$, giving a final complexity of $O(MN^2)$ making this approach an order of magnitude faster[5, Section 3.1].

3.1.2 Adding Elitism

Selection, crossover and mutation are applied to create the new population. For the next generation, this is the parent population. Elitism, in a broad sense,

is therefore included in this algorithm by having the parents compete with the children for the second round of evolving.[5, Section 3.4]

3.1.3 Removing the sharing parameter

In NSGA-II, crowding distance is used to ensure a diverse set of solutions by punishing solutions that are tightly packed. This serves the same function as the σ_{share} did in NSGA, therefore replacing it.[5, Section 3.4]

3.1.4 Implementation

There are differences in how NSGA-II is described in Deb, Agrawal, Pratap, *et al.* [5] and how it is implemented in JMetal[20]. In Deb, Agrawal, Pratap, *et al.* [5], the algorithm joins the parent and children population, sorts them and chooses fronts until the population size exceeds N. Then it sorts according to crowding distance[5, Section 3.2] and selects the top N solutions. Sorting according to crowding distance in the paper means first sorting based on non-domination rank and second based on local crowding distance. This means that all fronts except the last one are selected based on non-domination rank. In JMetal, the algorithm also joins the parent and children population, sorts the combined population according to non-domination rank and chooses fronts until the population size would exceed N if another front was added. If p fronts have already been added to the population, the front F_{p+1} would be sorted according to local crowding distance and individuals would be added to the population from F_{p+1} one at a time until the population has N individuals. The JMetal implementation does the same as described in the paper Deb, Agrawal, Pratap, *et al.* [5], but instead of sorting the entire collection of fronts it only sorts the last front, because the other fronts will be selected anyway.

3.2 NSEA: A Nondominated Sorting Evolutionary Algorithm

NSEA is a constrained multi-objective optimization algorithm. It was proposed in Jiménez and Verdegay [7] and has been applied in Deb, Pratap, and Meyarivan [2] (with poor results[2, section 5]) to the same set of problems. It divides the population into fronts containing feasible solutions, where the first front dominates the second and all below, the second dominates the third and all below and so forth. The very last front is used to contain all the infeasible solutions, see fig. 10.

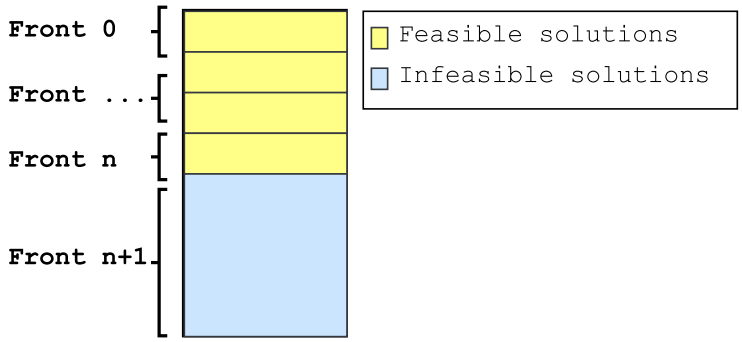


Figure 10: Pareto fronts in NSEA

The NSEA algorithm relies on a list of parameters that determine how the algorithm performs, see table 3. These parameters need to be defined before using the algorithm. The standard values populating table 3 have been chosen on the basis of the original paper, as they were said to have worked well in experiments[7, Section 2.6].

Variable	**Description**	**Standard Value**
$popsize$	The maximum solution population size	–
T	The maximum generation number	–
q	The probability distribution parameter	0.9
P_{2c}	The probability of 2-point-crossover	0.2
P_{ac}	The probability of arithmetical crossover	0.2
P_{um}	The probability of uniform crossover	0.2
P_{nm}	The probability of non-uniform crossover	0.2
c	The degree of non-uniformity	2
CF	The crowding factor: the number of individuals to be removed to avoid crowding	2

Table 3: Parameters for NSEA [7, Section 2.6]

3.2.1 Evolutionary Loop

The steps of the evolutionary loop are as follows[7, p. 3]:

1. Initialize population at generation 0.
2. set the current front number to 1.
3. Identify non-dominated individuals and place these non-dominated individuals in the current front.
4. increment the current front number.

5. If there are non-classified feasible individuals then go to 3.
6. Place infeasible individuals after the last front.
7. Ranking selection, sampling, crossover and mutation.
8. Generational replacement according to crowding factor model
9. If current generation < maximum generation then increment the generation number and go to 2, otherwise stop.

3.2.2 Operators

NSEA relies on operators for selecting, recombining and mutating individuals. For selection, the algorithm uses an operator originally described in Michalewicz [12, p. 60]. However, Jiménez and Verdegay [7]'s description differs from the original source. The probability of selecting a single individual in Jiménez and Verdegay [7, p. 3] is described in eq. (14), where the variable q is one of the parameters listed in table 3, X_s is the individual's genotype and s is a range defined as $s = 1, .., popsize$. The variable $rank(X_s)$ is defined as either the front the individual is in or the individual's position in the sorted population, depending on whether the individual is feasible or not. The selection scheme consists of applying eq. (14) to every individual and using this probability to determine whether to select them or not. The application of this selection scheme always leads to a situation where all individuals in the first Pareto-front have a probability of selection of 0.9.[2]

$$prob(X_s) = q(1-q)^{rank(X_s)-1} \tag{14}$$

The algorithm uses two crossover operators and two mutation operators [7, p. 4]. All of these rely on the predefined probability parameters shown in table 3.

2-point crossover: NSEA uses the same 2-point crossover as described in section 2.3

Arithmetical crossover: The algorithm has its own implementation of an arithmetical crossover [7, p. 4]. The operator takes two individuals $X_s = (x_1^s, ..., x_n^s)$ and $X_r = (x_1^r, ..., x_n^r)$ and determines the genotype of the offspring X'_s and X'_r as the sequence defined in eq. (15)

$$\begin{aligned} x_k'^s &= c \cdot x_k^s + (1-c) \cdot x_k^r \\ x_k'^r &= (1-c) \cdot x_k^s + c \cdot x_k^r \end{aligned} \tag{15}$$

Where $k = 1, ..., n$ and n is the length of the genotype

Uniform mutation: NSEA uses the same uniform mutation operator described in section 2.3.

Non-uniform mutation: The non-uniform operator in NSEA depends on the current generation number. Given an individual to mutate $X_s = (x_1^s, ..., x_k^s, ..., x_n^s)$ the algorithm generates a new individual $X'_s = (x_1'^s, ..., x_s'^k, ..., x_n'^s)$ using the following pair of functions:

[2] This differs from the source (Michalewicz [12]) cited in Jiménez and Verdegay [7].

$$x_k'^s = \begin{cases} x_k^s + (u_k - x_k^s) \cdot r \cdot (1 - t/T)^c, & \text{if a random digit is 0.} \\ x_k^s - (x_k^s - l_k) \cdot r \cdot (1 - t/T)^c, & \text{if a random digit is 1.} \end{cases} \tag{16}$$

where $[l_k, u_k]$ is the minimum and maximum values of the variable for a given problem, r is a random number, either 0 or 1, T is the maximum number of generations and t is the current generation number (starting from 1)[7, p. 4].

The standard probability of not applying an operator is 0.8. The probability of none of them being used is ≈ 0.41, shown in eq. (17). This means that some of the population will pass directly from the old generation into the new, without being altered.

$$0.8^4 = 0.4096 \tag{17}$$

3.2.3 Elitism

NSEA defines elitism as [7, p. 4]:

> Elitism: One individual chosen at random from the first Pareto-optimal front is copied into the new population in each generation.

This usage of elitism is equivalent to the definition in section 2.3.4.

3.2.4 Crowding Factor

When replacing the population with a newly generated one, the algorithm applies a crowding factor model. This model selects C random individuals from the old population and exchanges the one with the lowest distance to the new individual, with the new individual. The distance between the individuals ($X_s = (x_1^s, ..., x_n^s)$ and $X_r = (x_1^r, ..., x_n^r)$) is determined by:

$$distance(X_s, X_r) = \sqrt{\sum_{k=1}^{n} (x_k^s - x_k^r)^2} \tag{18}$$

where n is the length of the genotypes.

3.2.5 Implementation

The current implementation of the NSEA algorithm is based solely on the descriptions of Jiménez and Verdegay [7]. Although there is a discrepancy concerning the selection operator between Jiménez and Verdegay [7] and Michalewicz [12], this was disregarded because the authors might have intentionally modified the operator.

In the current implementation, selection is done by selecting an individual at random and generating a random number in the range from zero to the value of q. If the random number is lower than the selected individual's given probability value (as detailed in section 3.2.2), the individual is selected for evolution.

3.3 Ray's Algorithm

This algorithm is proposed in Ray, Tai, and Seow [8], and is referred to as Ray's. Unlike the other algorithms, it can operate in an *unconstrained*, *moderately constrained* and *highly constrained* manner[8, pp. 6-7]. How constrained the algorithm should be is decided upon usage. The algorithm is designed to work on constrained problems where at least one objective function f_1 and one constraint function c_1 is defined. Furthermore, the solution population size M and maximum amount of generations G_m needs to be decided upon as well [8, pp. 7-8]. The algorithm differs from the others, in that it works with matrices, as defined in eq. (19), eq. (20) and eq. (21).

$$\text{OBJ} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} \end{bmatrix} \tag{19}$$

$$\text{CONS} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} \tag{20}$$

$$\text{COMBINED} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} & c_{11} & c_{12} & \cdots & c_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} & c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} & c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} \tag{21}$$

3.3.1 The algorithm

It starts by randomly generating M solutions. Then a matrix, eq. (19), is computed, where each row is a solution's phenotype. Then another matrix, eq. (20), is computed where each row is a solution for which each of the s number of constraint satisfactions c is calculated by applying c to the solutions' genotype i.e. each row denotes for every solution, which of the constraints it violates. Then the matrices are combined into a single matrix, eq. (21). After the matrices are constructed, the pseudo-code in listing 1 is executed.

Listing 1: Ray's Algorithm in pseudo-code

```
Do {
    RankObj = Compute Pareto Ranking based on OBJ matrix
    RankCon = Compute Pareto Ranking based on CON matrix
    RankCom = Compute Pareto Ranking based on COMBINED matrix
    New Population = Individuals from the population
        where RankCom = 1 and the individual is Feasible.
    Do {
        Generate remaining members of new population:
                Select two individuals, reproduce them
                Put parents and children in New Population
    } Until population is full.
} Until maximum number of generations is reached.
```

First, the algorithm ranks the individuals in descending non-dominated order based on the different matrices, giving each individual a *RankObj*, *RankCon* and *RankCom* rank. Then a new population is created from the feasible individuals in the population. Finally, the remaining population is created by mating the original individuals. The algorithm keeps looping until the maximum number of generations is reached.

3.3.2 Operators & Models

Ray's algorithm relies on operators for selection and variance. These operators have different implementations, depending on how constrained the algorithm is. The algorithm also relies on Adaptive Niche Count and Non-overlapping Constraint Satisfaction models.

Selection Operator: The selection operator for the algorithm is different, depending on how constrained the algorithm is. Generally, if the algorithm is unconstrained, the selection operator simply bases selection upon an individual's *RankObj* ranking. Otherwise, the selection operator takes *RankCon* and *RankCom* into account. When selecting mating partners, the algorithm takes adaptive niche count and non-overlapping constraint satisfaction into account, as described later.

Variance Operator: During reproduction, every mating operation generates three additional individuals. One is generated by a uniform crossover between the parents. The other two is generated using random mix and move as described in Ray, Tai, and Seow [8, pp. 410-411]. Generally, random mix and move combines the two individuals' genotypes based on stochastic behaviour. This behaviour is achieved by selecting random numbers and determining how to combine the individuals based on the values. It creates solutions that are both crossed from parents and mutated by a random factor. It should be noted that this might be a disadvantage, since there is a chance that good genes are eliminated through mutation.

Adaptive Niche Count: The adaptive niche count[8, p. 411] of an individual A is the number of individuals that are within the average distance metric of A. The procedure for computing the adaptive niche count can be found in listing 2.

Listing 2: Adaptive Niche Count

```
For every individual I in the population M:
  Compute the euclidean distances between I and all other M-1
      individuals.
  Compute the average Euclidean distance.
  Count the number of solutions that are within the average
      distance.
```

In the algorithm, the individuals that have a low niche count are preferred over others, since this promotes a greater diversification[8, p. 411]. Computing the adaptive niche count however, is expensive and is estimated to run in at least $O(M^2)$.

Non-overlapping Constraint Satisfaction: Non-overlapping Constraint Satisfaction is a strategy where solutions are allowed to mate if they complement each other in their satisfaction of constraints[8, p. 412]. This strategy consists of computing the sets of constraints that each solution to be made satisfies and then finding the intersection. It is described in the following procedure: Given three individuals, A, B and C, the selection of a partner for A is decided by:

if $(S_A \cap S_B > S_A \cap S_C)$, then the partner is C.

if $(S_A \cap S_B < S_A \cap S_C)$, then the partner is B.

if $(S_A \cap S_B = S_A \cap S_C)$, then the partner is randomly chosen between B & C.

Where S_A, S_B and S_C denotes the sets of satisfied constraints for individual A, B and C, respectively.

3.3.3 Implementation

In the implementation, ranking was calculated using a custom implementation of JMetal's Ranking interface. This meant that instead of calculating the matrices OBJ, CONS and COMBINED for each individual every generation, a non-dominated ranking was created based on the individual's objective value, constraint violation and a combination of these. This gives us a difference in data structure, from the unorthodox matrices to the standard JMetal implementation, but it does not affect the logic of the algorithm. Otherwise, the algorithm follows the descriptions in Ray, Tai, and Seow [8].

3.4 NSFI-2POP: Non-dominated Sorting Feasible-Infeasible 2 Populations

Non-Dominated Sorting Feasible-Infeasible 2 Populations (NSFI-2POP) is an algorithm proposed in Scirea, Togelius, Eklund, *et al.* [1]. It uses ideas from the algorithms FI-2pop[14] and NSGA-II[5], the idea being that by not deleting the infeasible solutions, but evolving them with the sole goal of becoming feasible. This increases novelty in the system because the infeasible solutions are not restricted by the fitness function, but only by constraint(s), thus allowing them to explore the entire solution space[1, section 4]. Because the results only consist of feasible solutions, the population size of NSFI-2POP is twice the size, compared to the other algorithms. The size is n for the feasible population and also n for the infeasible, since members of the infeasible population will never be included as a result. NSFI2POP works as follows:

1. Initialize the infeasible population by generating random solutions.
2. Initialize the feasible population with 0 solutions.
3. Evaluate both populations, i.e. calculate their phenotype and check their constraints.
4. Find all infeasible solutions in the feasible population and move them to the infeasible population[3]

[3]This is not necessary in the first generation, because the feasible population contains zero individuals.

5. The reverse of the above. Move feasible solutions from the infeasible population to the feasible population
6. Evolve the feasible population based on NSGA-II
7. Evolve the infeasible population based on FI-2pop
8. If the maximum number of generations has not been reached, go to 3

The selection, mating, mutation and replacement of the feasible population is done just as if it was NSGA-II. For details see section 3.1. It is worth noting that NSGA-II also looks at the constraints, which differs from FI-2pop where the feasible population is only evaluated based on objective. [14, Appendix A, page 325]. The evolving of the infeasible population was done as described in Kimbrough, Koehler, Lu, *et al.* [14, Appendix A]. Fitness-Proportionate Selection is used to select two parents, as described in Lipowski and Lipowska [21]. Single-point Crossover is then used to generate two children, and these are mutated using Non-Uniform Mutation, as implemented in the JMetal Framework.

3.5 Summary

This section introduces the four different algorithms chosen for comparison, NSGA-II, NSEA, Ray's and NSFI-2POP. NSGA-II is the most widely used of the algorithms and is included in many genetic algorithm libraries. NSEA is an algorithm that includes infeasible solutions in a separate front and applies ranking differently on feasible and infeasible solutions. Ray's algorithm uses three different modes of computation, determining how it should approach solutions. It also mutates and crosses solutions using the same variance operator. NSFI-2POP is a combination of NSGA-II and FI-2Pop, which is an algorithm that evolves feasible and infeasible solutions in different populations. Each algorithm has been examined to give an understanding of how it works and what operators it utilizes. Each algorithm has been implemented in Java using the JMetal framework.

4 Data Collection

The data collection is done via JMetal's Measure system and a MySQL database. The data collection results are contained in a data entity aptly named an experiment. An experiment contains the results from all the algorithms NSGA, NSEA, Rays unconstrained, Rays moderately constrained, Rays highly constrained and NSFI-2POP applied to the problems CTP1-7. After the completion of an experiment, it is committed to the database. The solutions saved in the database are only the non-dominated individuals for each generation. This approach was chosen because the measures only measure performance based on the non-dominated individuals, see section 5 and section 2.9 for more details. Ten experiments have been conducted for each of the parameter sets seen in table 4. The parameter sets have been chosen to demonstrate how the algorithms perform under different circumstances. For instance, some algorithms might perform better with less solution variables.

Population Size	Number of Solution Variables	Number of Generations
50	5	100
50	50	100
250	5	500

Table 4: Parameter sets for experiments

5 Data Analysis

In this section, we analyse the data obtained from the algorithms. The purpose of the project is to evaluate the algorithms and to do this we will apply different measures to the data collected from running the algorithms. First, the measures used will be described. Then, the results of the analysis will be presented.

5.1 Measures

Three measures have been applied:

- Distance Measure
- Distribution Measure
- Convergence Measure

5.1.1 Distance Measure

The distance measure is based on the measure in Deb, Agrawal, Pratap, *et al.* [5] as described in section 2.9.2. The purpose of the measure is to get the average distance from the solutions to the global Pareto-optimal front. This is done by finding the nearest point on the front from a given solution point, and then measure the Euclidean distance between the two points. The formula for the Euclidean distance between a point (x, y) and a point (x_0, y_0) is seen in eq. (22)

$$d(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} \tag{22}$$

Hence the distance from the point (x_0, y_0) to a point $(x, y) \equiv (x, f(x))$, which means the distance to a point on the curve described by the function $f(x)$, can be computed by eq. (23)

$$d(f(x)) = \sqrt{(x - x_0)^2 + (f(x) - y_0)^2} \tag{23}$$

We are interested in the nearest point on the curve from the point (x_0, y_0). Said in another way, we wish to find the point $p_{min} = (x, f(x))$ where the distance from (x_0, y_0) to p_{min} is minimal. The minimum of a function $d(x)$ can be found where the derivative is zero, written as $d'(x) = 0$. Hence, we are looking for eq. (24).

$$x_{min} = d'(x) = 0 \tag{24}$$

We will then have eq. (25)

$$d_{min}(x_0, y_0) = d(f(x_{min})) = \sqrt{(x_{min} - x_0)^2 + (f(x_{min}) - y_0)^2} \tag{25}$$

The distance measure, describing the average distance from points in a generation to the global Pareto-optimal front, will then be as in eq. (26), where s_i is a solution in the first non-dominated front with index i and n is the number of solutions in the first non-dominated front.

$$\Delta = \sum_{i=1}^{n} \frac{d_{min}(s_i)}{n} = \sum_{i=1}^{n} \frac{\sqrt{(x_{min} - x_i)^2 + (f(x_{min}) - y_i)^2}}{n} \tag{26}$$

The measure has been developed in the programming language Python, where the built-in optimize function has been applied to obtain d_{min}. The solution described by eq. (26) can be used for any function $f(x)$, but if the function f is a first-order linear function that can be described by the formula in eq. (27)

$$ax + by + c = 0 \tag{27}$$

then the function f can be written as in eq. (28).

$$y = -\frac{a}{b}x - \frac{c}{b} \tag{28}$$

If the formula of the function f can be written as in eq. (28), then the distance from a point to the function f can be found with eq. (29). [22]

$$\Delta_2 = \frac{\mid ax_0 + by_0 + c \mid}{\sqrt{a^2 + b^2}} \tag{29}$$

The distance measure in eq. (29) is more efficient when implemented in Python, hence it has been used when the function f is a first-order linear function.

5.1.2 Distribution Measure

The distribution measure used in Deb, Agrawal, Pratap, *et al.* [5] as described in section 2.9.3 resembles the distribution measure used in this project. It is basically the standard deviation of the distance matrix, where the distance matrix describes the Euclidean distances between the points in the first non-dominated front. Unlike the solution in Deb, Agrawal, Pratap, *et al.* [5] the distances between all points in the non-dominated front is calculated. The measure is expressed mathematically in eq. (30), where m_i is an element of the distance matrix, $\overline{m}$ is the mean of the distances and n is the number of elements in the matrix.

$$d = \sqrt{\frac{\sum_{i=0}^{n} \mid m_i - \overline{m} \mid^2}{n}} \tag{30}$$

5.1.3 Convergence Measure

Convergence time is a measure that describes the development of the solution set through all generations of all experiments that have the same parameters. Here, it is defined as the number of generations before the average distance reaches a plateau. The plateau can be defined in various ways, but generally it is the place where the average distance from the solutions to the global Pareto-optimal front is not improved or only slightly improved in future generations.

Our convergence measure is to plot the generations with the average distance to the Pareto front and look for the plateau in the plots. This option was selected for the project since the distance measure provides a clear picture of the effectiveness of the algorithm. The distance measure, combined with a measure showing the average size of the Pareto front allows for easy and quick assessment. Assessing the plots in person also enables a better flexibility towards edge cases, such as an algorithm reaching a plateau, but worsening distance later in the run. When observing the plots, the observer still needs to be aware that the distribution of solutions is not visible. This means that the plots could have a plateau in distance to global Pareto-optimal front, but still has not found a plateau in the distribution of solutions.

5.2 Experiment Results

By applying the measures to the data obtained from running the algorithms, an understanding of the performance of the algorithms can be attained. Each algorithm has been run ten times for each parameter setting. The results of the ten experiments are then aggregated in a way that makes it possible to comment on the general performance of the algorithm. Descriptions of the performance of each algorithm will be provided in this section followed by a comparison of the algorithms.

5.2.1 NSGA-II

5 Variables, 50 Population, 100 Generations

NSGA-II quickly converges towards a solution in all test problems. The algorithm finds good solutions to most CTPs, but in some cases, it does not find perfect

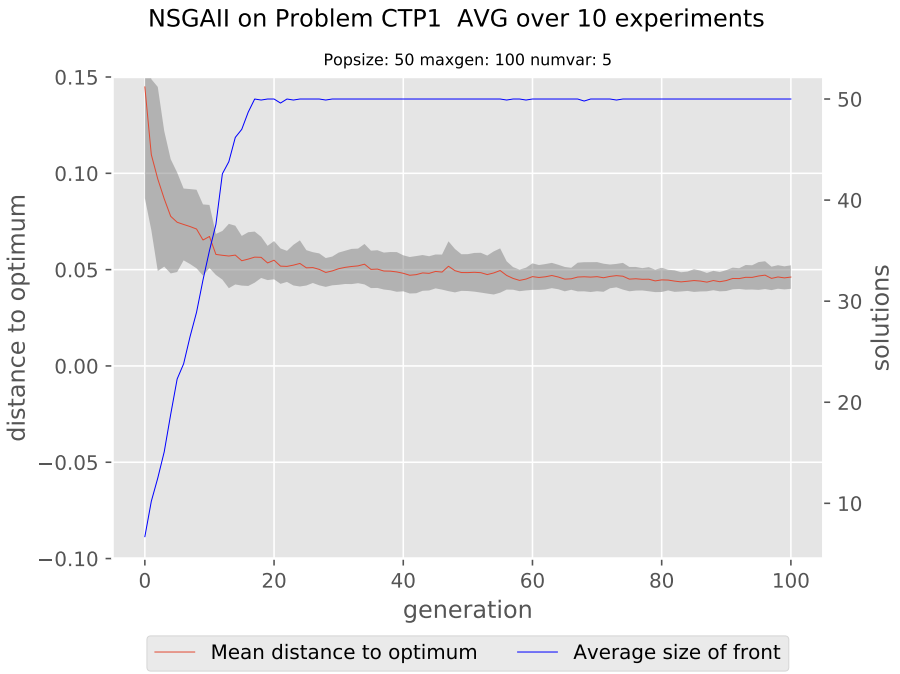


Figure 11: Global Pareto front distance development of CTP1 with NSGA-II with 5 solution variables

solutions. In CTP1 and CTP4, NSGA-II finds a good solution within the first 20 generations, but afterwards it only slowly converges towards a better solution in CTP4 and does not find better solutions in CTP1. The plot of CTP1 and 4 can be seen in fig. 11 and fig. 12 respectively.

CTP5 demonstrates a situation where NSGA-II finds a set of almost perfect solutions, which can be seen in fig. 13.

The most challenging problem is CTP4, which is the CTP where the solutions must travel through "tunnel-like" feasible regions to find the perfect solution. The algorithm finds it difficult to maintain diversity while converging towards a solution. It seems like solutions in different feasible-region tunnels share genomes, which makes them infeasible. Only solutions mating with other solutions in the same subregion will stay in the population and potentially lower the distance to the global Pareto-optimal front. This would explain the slow convergence towards the global Pareto-optimal front after generation 20, because when the algorithm reaches generation 20, the solutions have reached the beginning of the tunnels. see fig. 14.

Considering the distribution of the solutions over the Pareto-optimal front, NSGA-II performs better than most algorithms, but still have trouble in some test problems, for instance, in CTP1 the algorithm finds more solutions with low $f1$ values than solutions with higher values. In CTP1, it still finds solutions spread over the entire global Pareto-optimal front, meaning that the distribution is still acceptable. The plot can be seen in fig. 15.

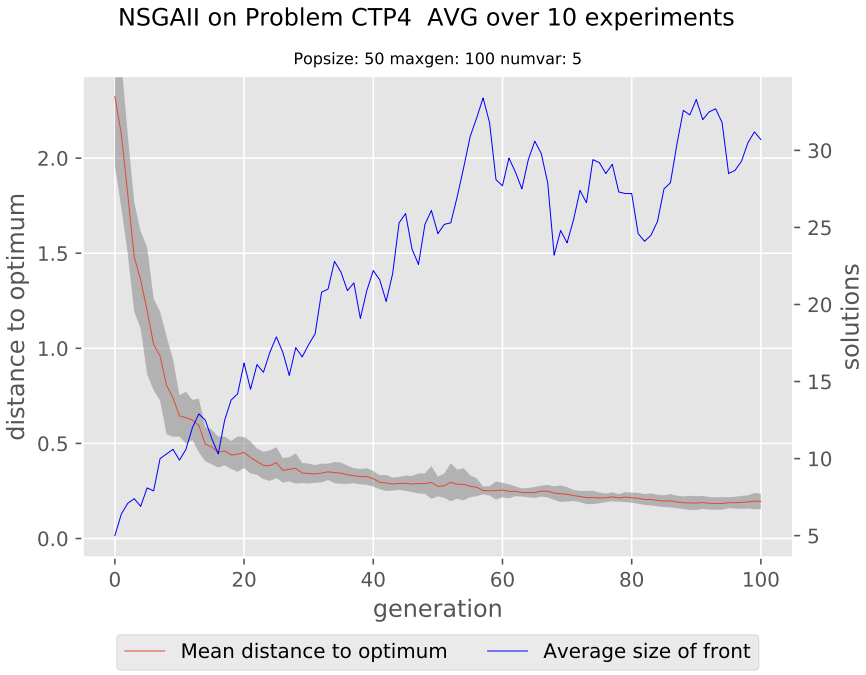


Figure 12: Global Pareto front distance development of CTP4 with NSGA-II with 5 solution variables

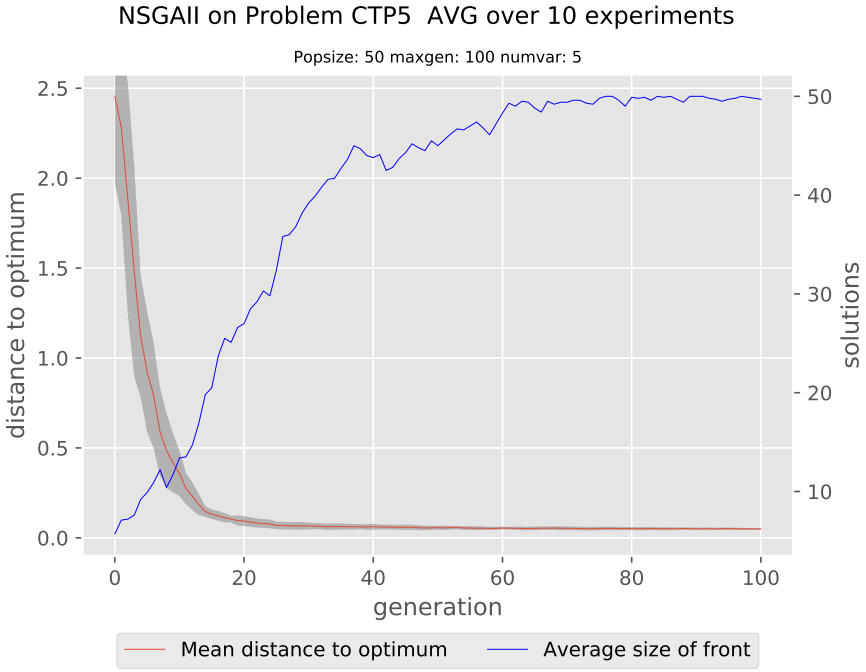


Figure 13: Global Pareto front distance development of CTP5 with NSGA-II with 5 solution variables

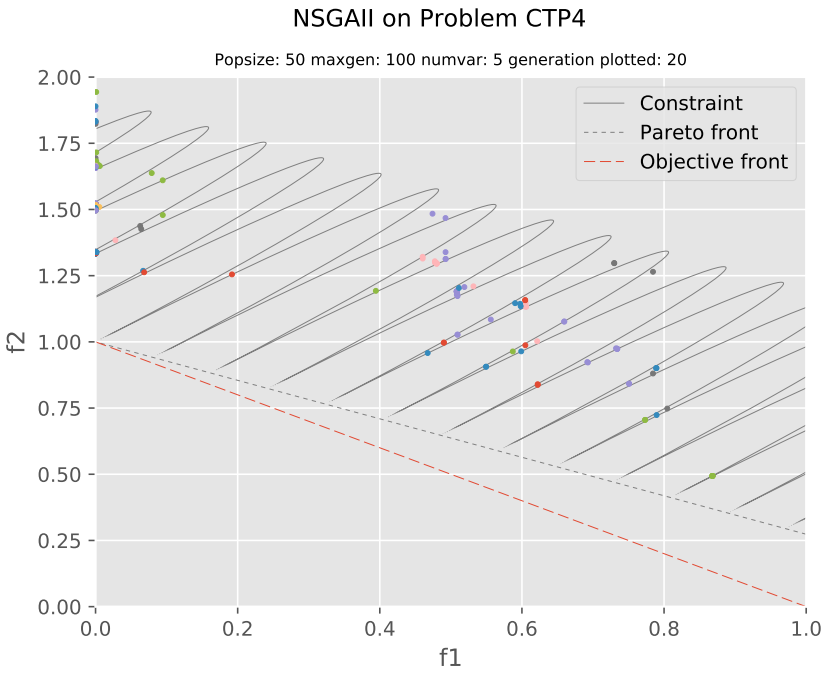


Figure 14: CTP4, NSGAII, generation 20 of 100 generations, 5 variables, 50 population

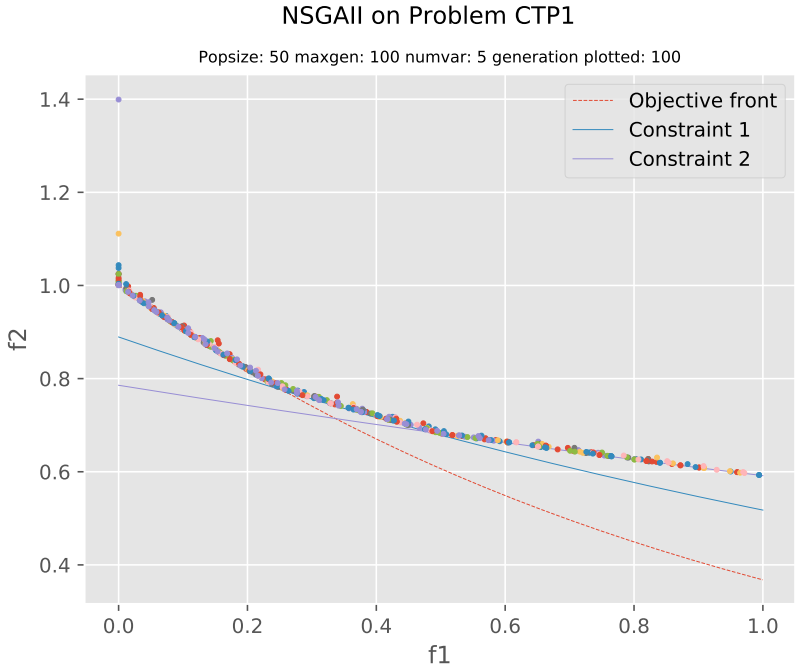


Figure 15: CTP1, NSGAII, 100 generations, 5 variables, 50 population

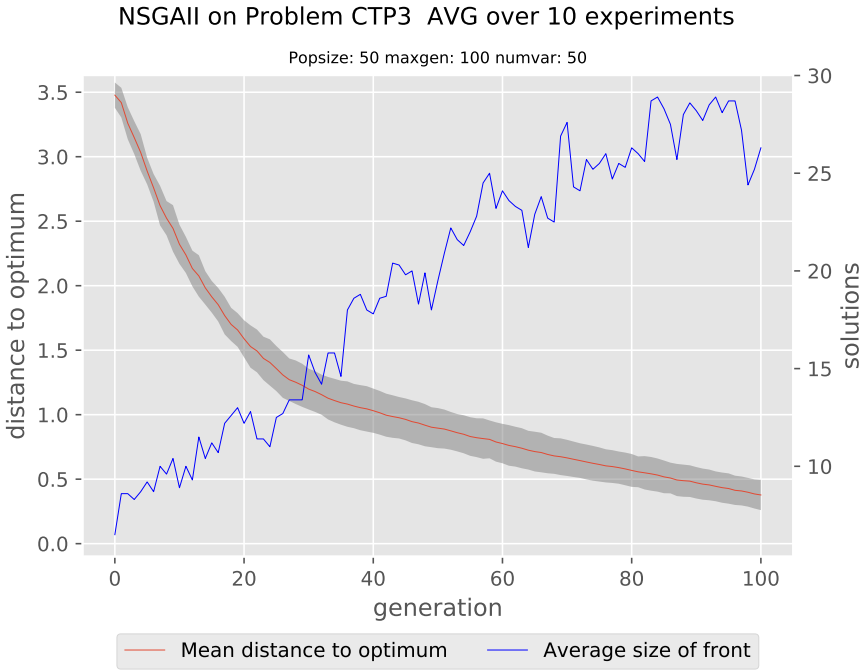


Figure 16: Global Pareto front distance development of CTP3 with NSGA-II with 50 solution variables

50 Variables, 50 Population, 100 Generations

When the number of variables in the solutions is increased, it is more difficult for NSGA-II to find solutions. NSGA-II still converges towards a solution, but the rate of convergence is lower. A good example is CTP3, where the algorithm finds a better solution every generation, but never finds a perfect solution. CTP3 can be seen in fig. 16. The variance between the different experiments is greater in some of the test problems, for instance CTP1 clearly has a large standard deviation, which is represented by the grey area in fig. 17. Furthermore, it is difficult for the algorithm to find solutions on the right-hand side of the global Pareto-optimal front in CTP1, as seen in fig. 18.

5 Variables, 250 Population, 500 Generations

The algorithm finds good solutions in almost all problems. The solutions are close to the global Pareto-optimal front and in several of the problems all solutions in the population are placed in the first non-dominated front. CTP4 is again the most difficult problem, but the algorithm nevertheless finds good solutions placed close to the global Pareto-optimal front with a good distribution. The CTP4 problem is seen in fig. 19. CTP7 is a good example of a difficult problem, where NSGA-II finds a perfect solution.

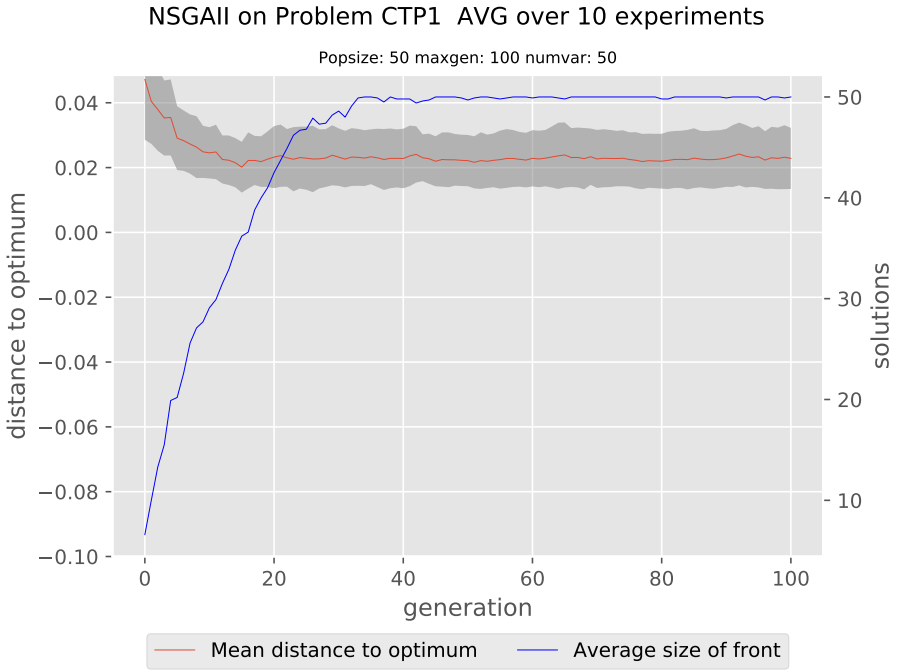


Figure 17: Global Pareto front distance development of CTP1 with NSGA-II with 50 solution variables

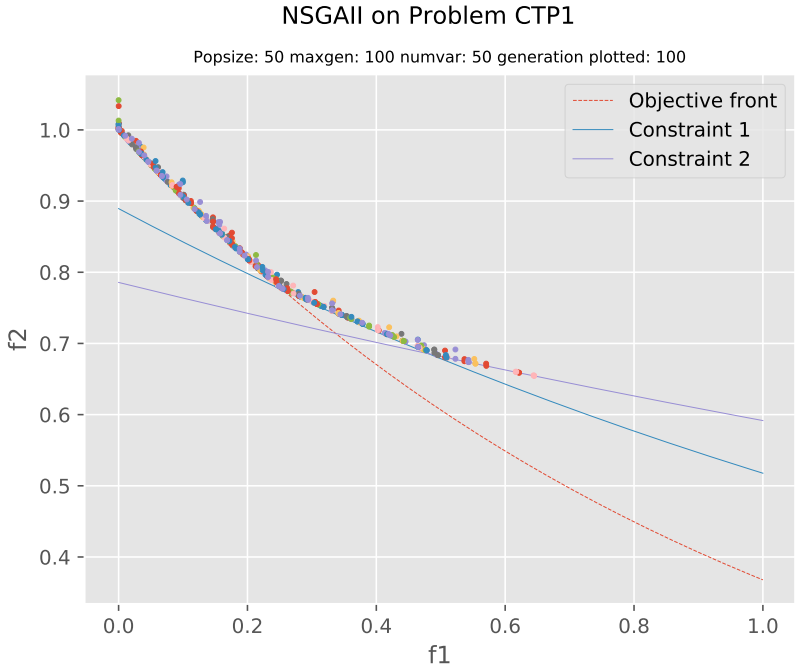


Figure 18: Objective plot for NSGAII with 50 variables

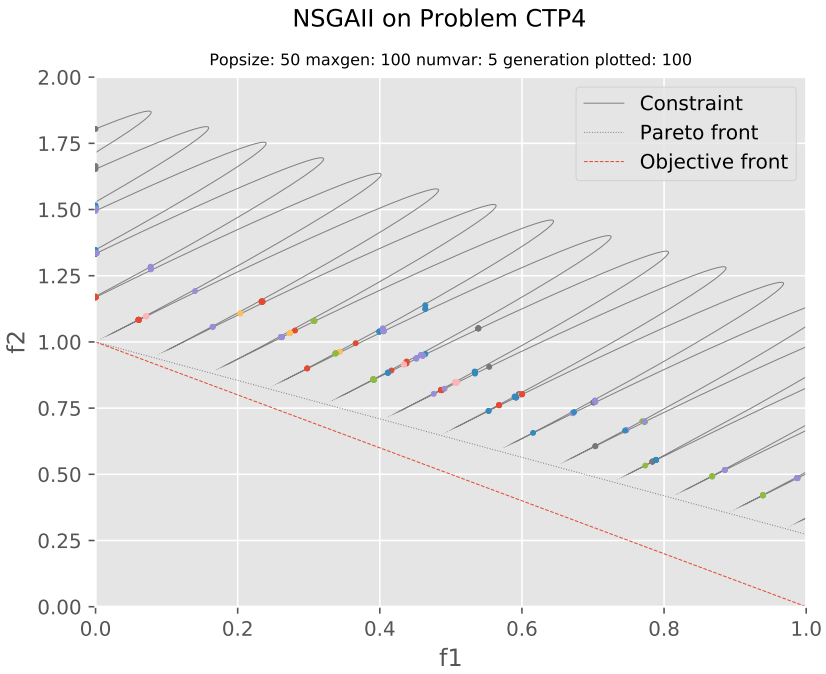


Figure 19: CTP4, NSGAII, 100 generations, 5 variables, 50 population

NSGA-II Conclusion

NSGA-II finds good solutions to most CTPs, but only nearly-perfect solutions on few CTPs if the number of generations is 100. If the number of generations is increased, the algorithm only finds nearly-perfect and perfect solutions. NSGA-II is good at finding diverse solution sets, but sometimes the diversity can cause problems when converging towards a solution.[4] The convergence is nevertheless not so slow that it never finds good solutions, hence it can be concluded that NSGA-II is a good algorithm for the test problems given in this project.

5.2.2 NSEA

5 Variables, 50 Population, 100 Generations

NSEA never comes near a perfect solution, except in CTP5 where a few solutions are near the Pareto-optimal front (see fig. 20). When looking at the objective values in fig. 21 it can clearly be seen that the distribution of the solutions is unsatisfactory, as only few solutions move into the "tunnels".

In most of the CTPs, NSEA randomly converges towards the global Pareto-optimal front, meaning that the distance to the global Pareto-optimal front oscillates from generation to generation. This is most clearly observed in fig. 22. In most of the CTPs, there is a weak tendency of converging towards a perfect solution. For instance, the distance to the global Pareto-optimal front oscillates in CTP3, but it continually approaches the perfect solution.

[4] As can be seen in CTP4 with 100 generations, where the diversity causes problems when converging towards the global Pareto-optimal front

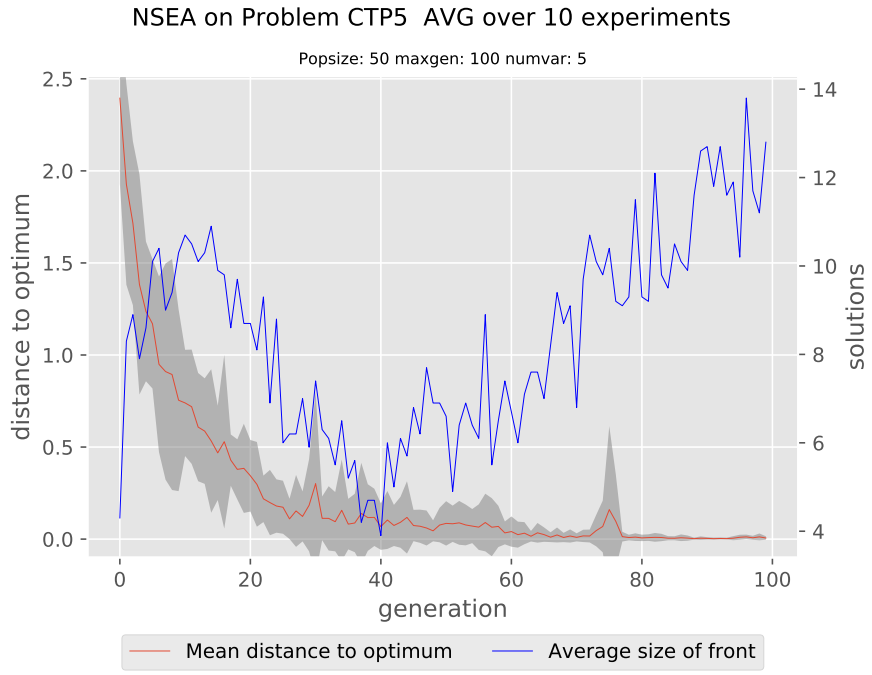


Figure 20: Global Pareto front distance development of CTP5 with NSEA with 5 solution variables

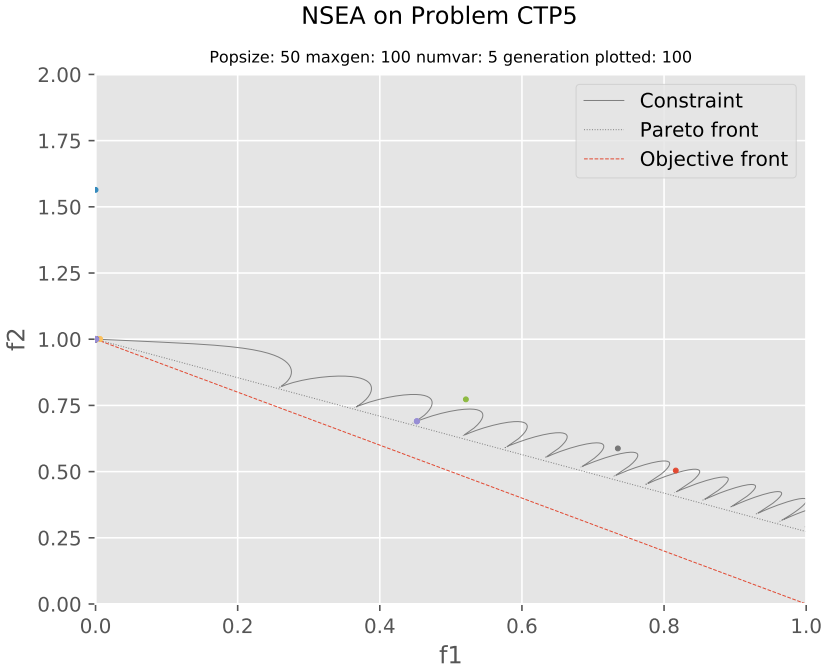


Figure 21: Objective front distance of CTP5 with NSEA with 50 solution variables generation 100

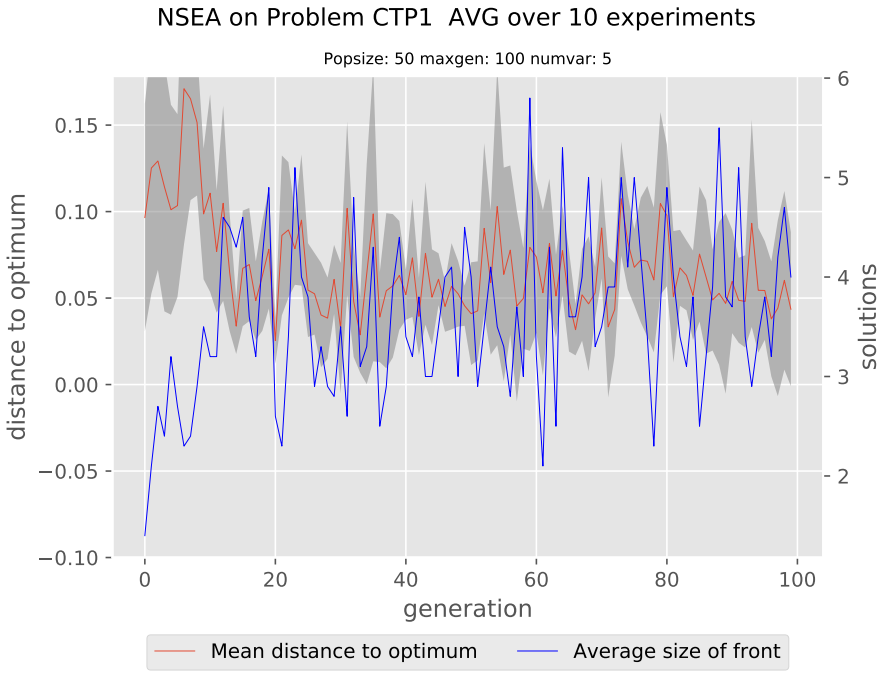


Figure 22: Global Pareto front distance development of CTP1 with NSEA with 5 solution variables

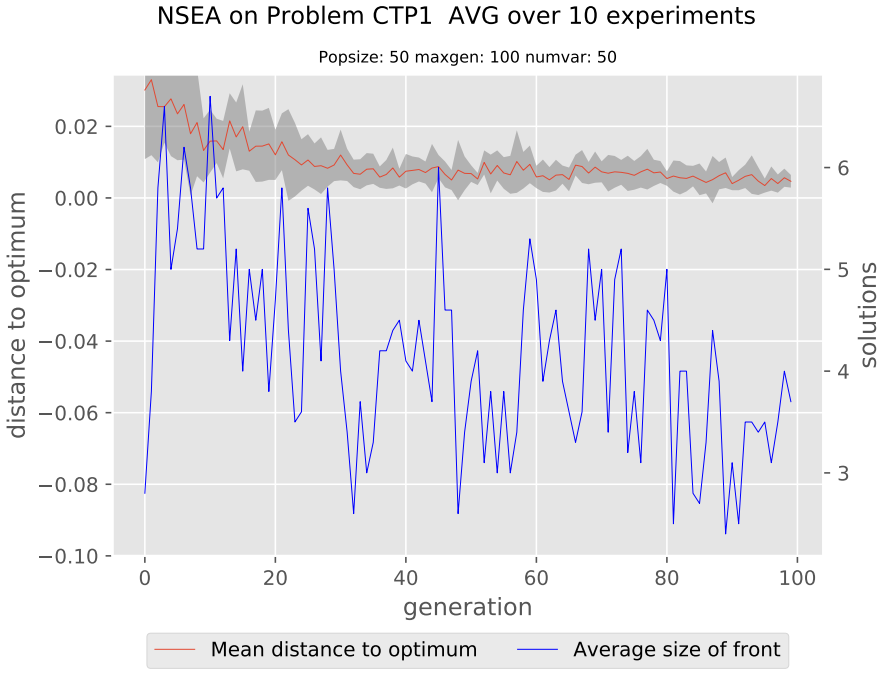


Figure 23: Global Pareto front distance development of CTP1 with NSEA with 50 solution variables

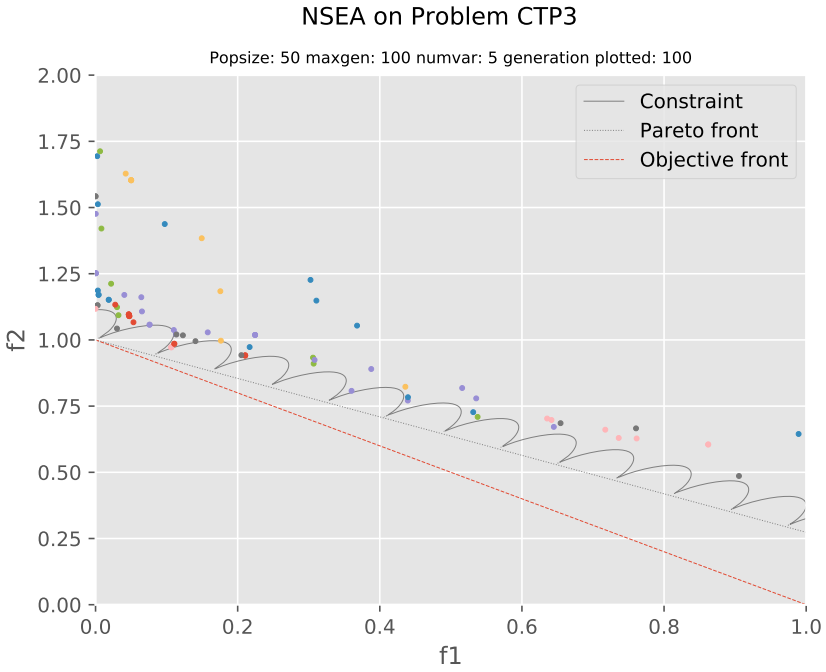


Figure 24: Objective value plot of NSEA on CTP3 with 5 solution variables

However, the final set of solutions are still far from the global Pareto-optimal front as seen in fig. 24. Worse yet, on average, only about a third of the solutions are in the Pareto-front at the final generation.

NSEA provides a good set of solutions for CTP6 and CTP7. The final solutions are close to the constraint boundary that constitute the global Pareto-optimal front in CTP6 and several of the solutions lie on the global Pareto-optimal front in CTP7. The plots of the final solutions of CTP6 and CTP7 can be seen in fig. 26 and fig. 27, respectively.

50 Variables, 50 Population, 100 Generations

When the number of variables in the solutions is increased, the same situation as with NSGA-II appears. The algorithm never really finds a plateau on any of the problems. It continually improves the distance to the global Pareto-optimal front, but never achieves the goal. Looking at CTP3, the solutions are nearing a value of zero, but in generation 100 the distance is still between 1.0 and 1.5, as seen in fig. 25. Compared to NSGA-II in fig. 16, this is an inferior result. When comparing the solution distances in CTP1, NSEA actually has a better solution most of the time, even though it oscillates more than NSGA-II, fig. 23 and fig. 17. NSEA could still have worse distribution than NSGA-II.

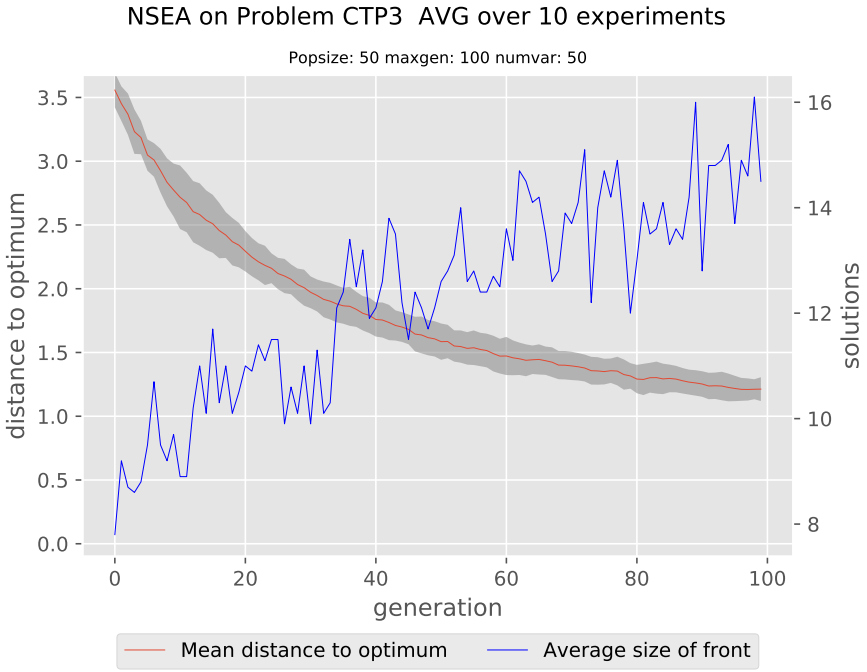


Figure 25: Global Pareto front distance development of CTP3 with NSEA with 50 solution variables

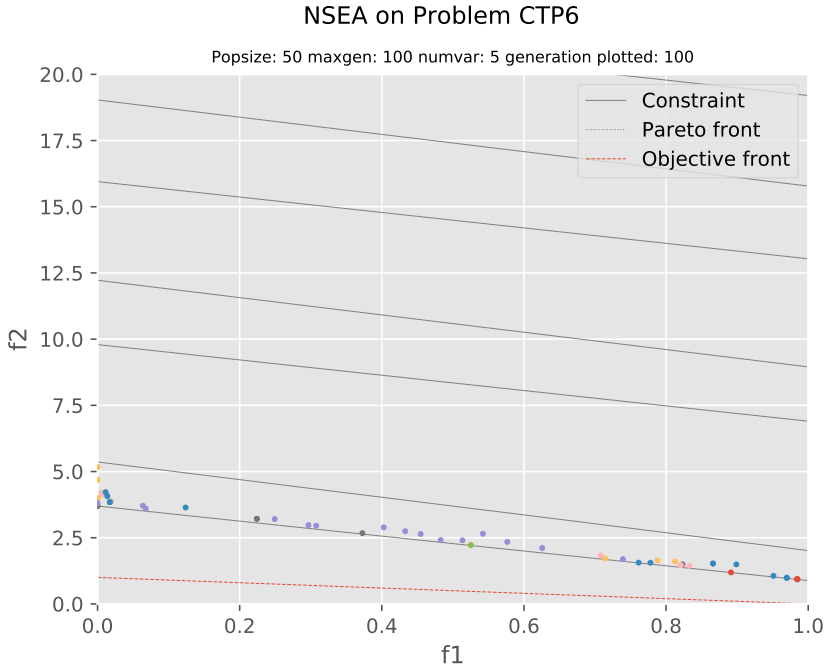


Figure 26: Objective value plot of CTP6 with NSEA with 5 solution variables

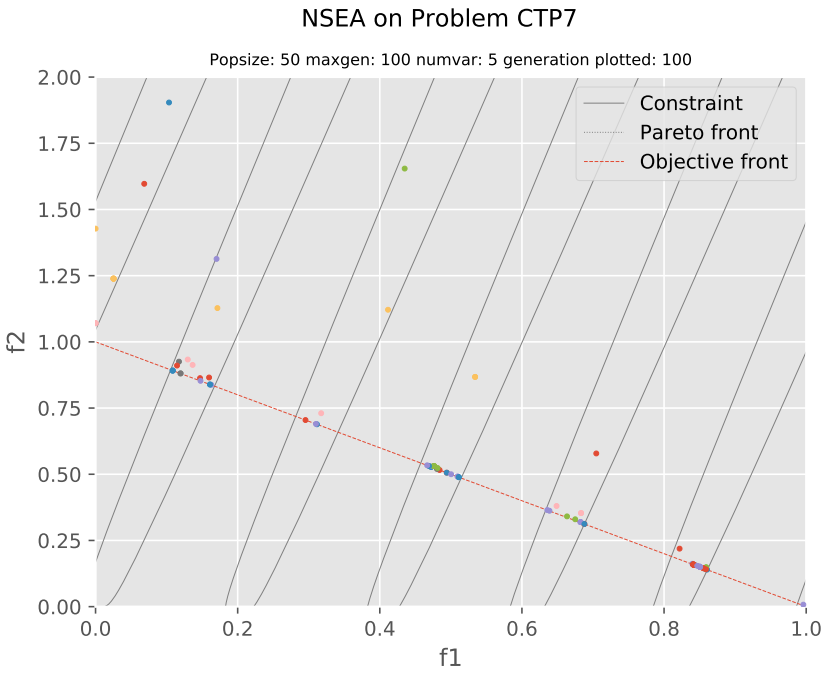


Figure 27: Objective value plot of CTP7 with NSEA with 5 solution variables

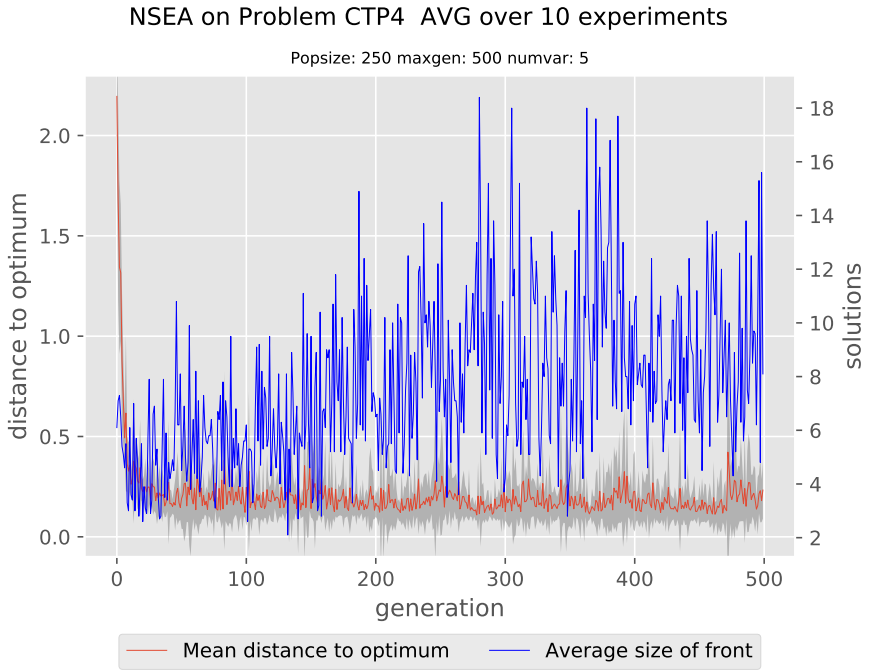


Figure 28: Global Pareto front distance development of CTP4 with NSEA with 5 solution variables and 500 generations

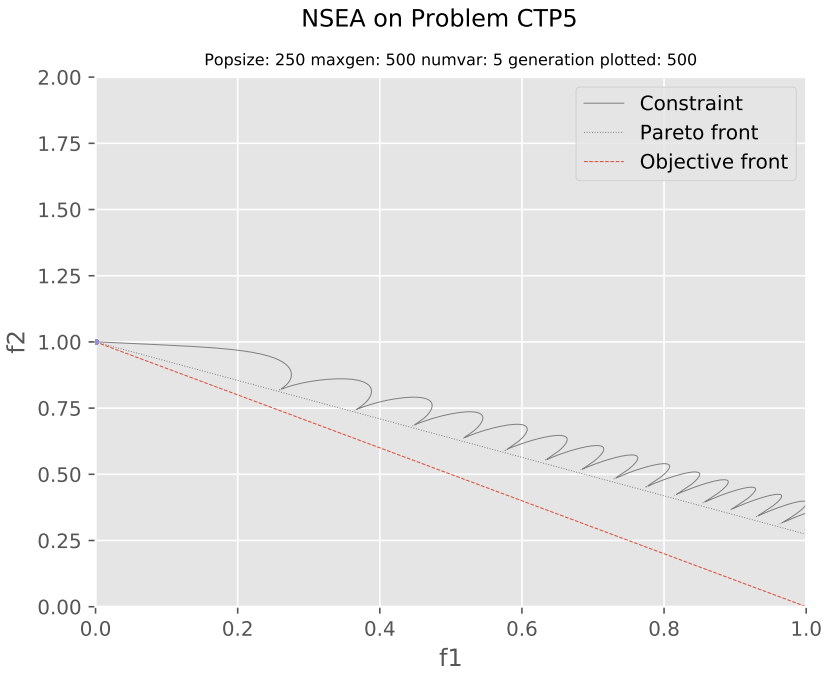


Figure 29: Objective value plot of CTP5 with NSEA with 5 solution variables and 500 generations. Note: all solutions lie in $\approx (0, 1)$.

5 Variables, 250 Population, 500 Generations

The first few generations, NSEA oscillates towards a better solution set, but afterwards it stops improving and oscillates approximately around the same distance value. An example of this situation can be seen in fig. 28. In some of the CTPs, for instance CTP5 fig. 30 and fig. 29, it finds a perfect solution according to the distance to the global Pareto-optimal front in less than 100 generations. This does not mean that it is a completely perfect solution, which can be seen when looking at the objective value plot, fig. 29. Most of the population are placed directly on the global Pareto-optimal front, but the solutions are placed in the exact same position. It can be concluded that this is a bad result, because in multi-objective algorithms a set of solutions spread over the whole Pareto-optimal front is usually desired. Generally, with these settings, NSEA have few solutions in the first non-dominated front, the distribution is poor and the solutions are far from the global Pareto-optimal front.

NSEA Conclusion

NSEA's solutions oscillate greatly during the evolution. It slowly finds better solutions, but rarely gets as close to the global Pareto-optimal front as NSGA-II, and generally performs worse than NSGA-II and NSFI-2POP.

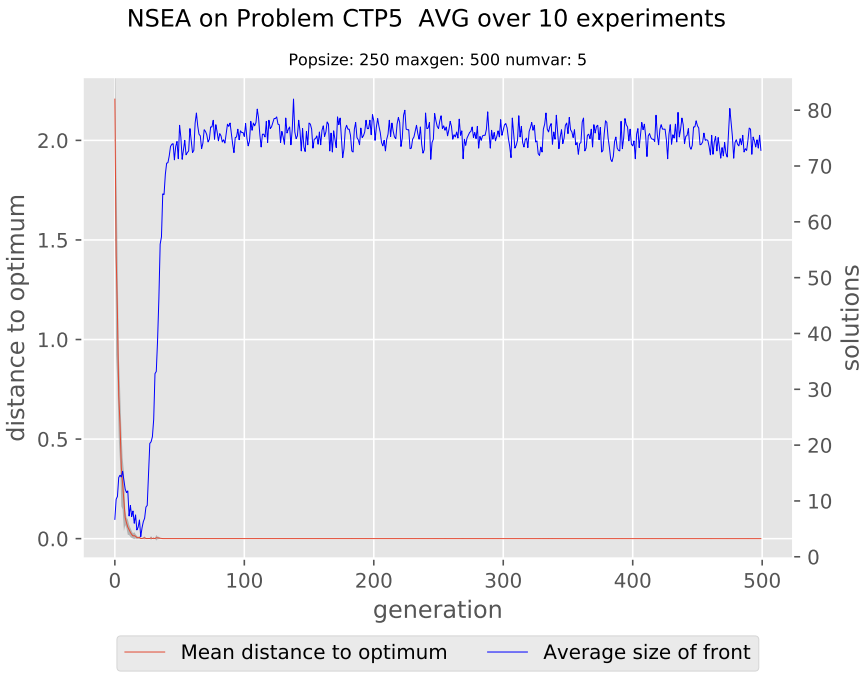


Figure 30: Global Pareto front distance development of CTP5 with NSEA with 5 solution variables and 500 generations.

5.2.3 Ray's Algorithm

5 Variables, 50 Population, 100 Generations

When looking at the convergence graphs, certain reservations need to be made. The unconstrained Ray's is actually not a constrained multi-objective optimization algorithm, which means that the solutions found can be positioned in the infeasible area. The convergence graph will show good results for this version of Ray's, but in reality, the solutions could be placed on the wrong side of the global Pareto-optimal front. This can be seen in fig. 31. When looking at the solution plots of CTP7 for the three different Ray's implementations, the unconstrained version actually gets closer to the global Pareto-optimal front without violating the constraints. This could be due to the unconstrained nature of the algorithm, which means that the "tunnels" do not cause the algorithm problems, see fig. 32. Ray's algorithm never reaches a plateau in any of the implementations, because the distance to the front varies a lot between generations and it is not necessarily in the direction of the global Pareto-optimal front. This is likely due to the variance operator used in Ray's algorithm (see section 3.3.2).

50 Variables, 50 Population, 100 Generations

An example showing the development of the distance to the Pareto-optimal front can be seen in fig. 33. The algorithm still retains a tendency to decrease the distance to the Pareto-optimal front, despite the increase in problem variables. The ideal distance being 0, indicates that it still does not perform ideally.

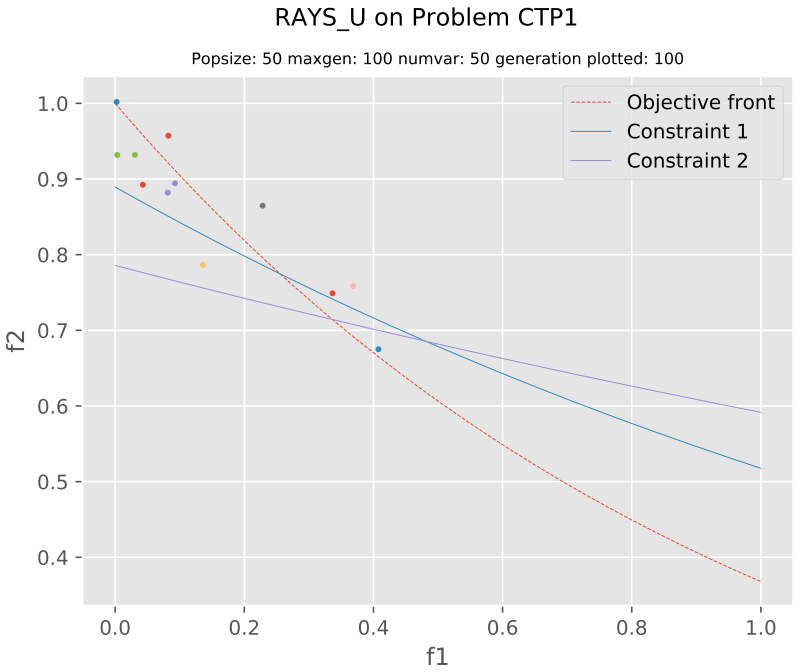


Figure 31: Solution plot of Ray's unconstrained CTP1 with 50 variables, 100 generations and 50 population

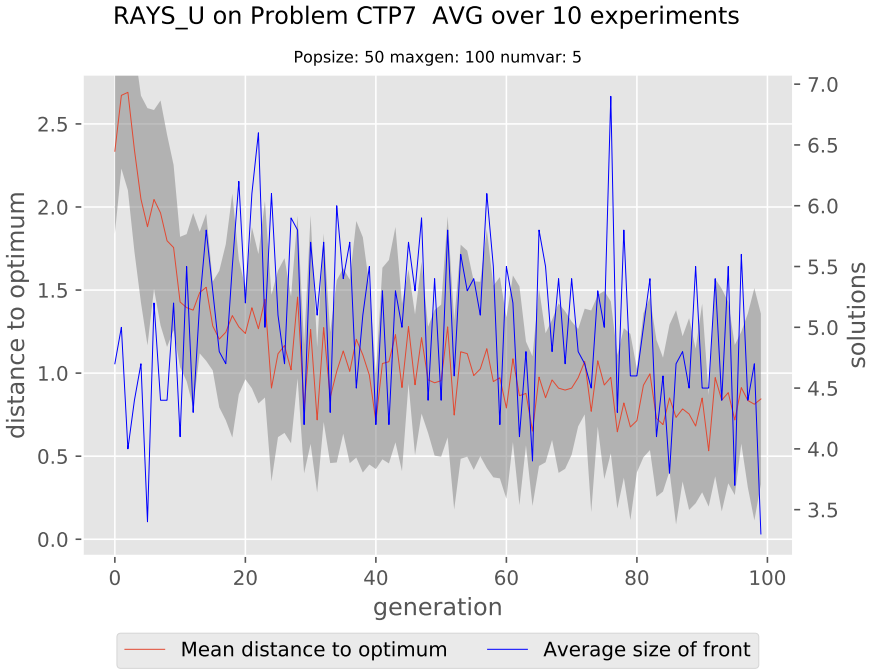


Figure 32: Global Pareto front distance development of CTP7 with Ray's unconstrained with 5 solution variables

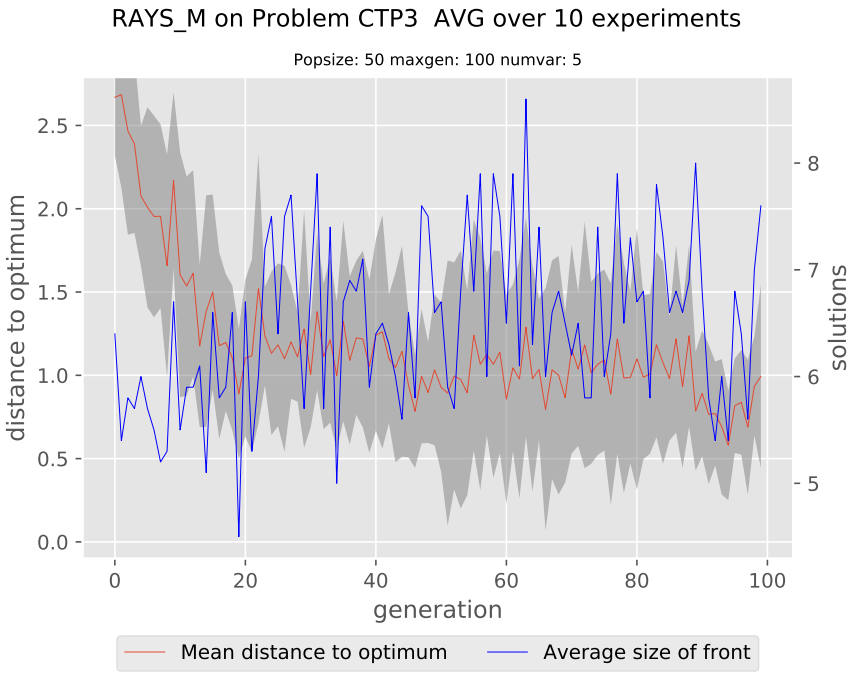


Figure 33: Global Pareto front distance development of CTP3 with Ray's moderately constrained with 50 solution variables

5 Variables, 250 Population, 500 Generations

Looking at the convergence plots, the set of solutions mostly does not improve after generation 100, hence 500 generations does not improve the final result of the algorithm.

Ray's conclusion

Ray's algorithm is one of the worst performing algorithms when tested on the problems of this project. The convergence of the solutions oscillates throughout the experiments, meaning that the convergence is very slow and uncertain. The final non-dominated fronts are usually far from the global Pareto-optimal fronts, hence the algorithm is not recommendable when solving any of the test problems.

5.2.4 NSFI-2Pop

5 Variables, 50 Population, 100 Generations

The NSFI-2pop algorithm converges towards the global Pareto-optimum quickly in all of the problems. In general, the performance of the algorithm is very much like NSGA-II's. NSFI-2Pop produces solutions that are, on average, at the same distance from the global Pareto-optimal front as NSGA-II's solutions. However, the average standard deviation across the experiments is higher. This difference could be attributed to NSFI-2Pop's conservation of infeasible genes, since it is possible for infeasible solutions to keep evolving, we can imagine that

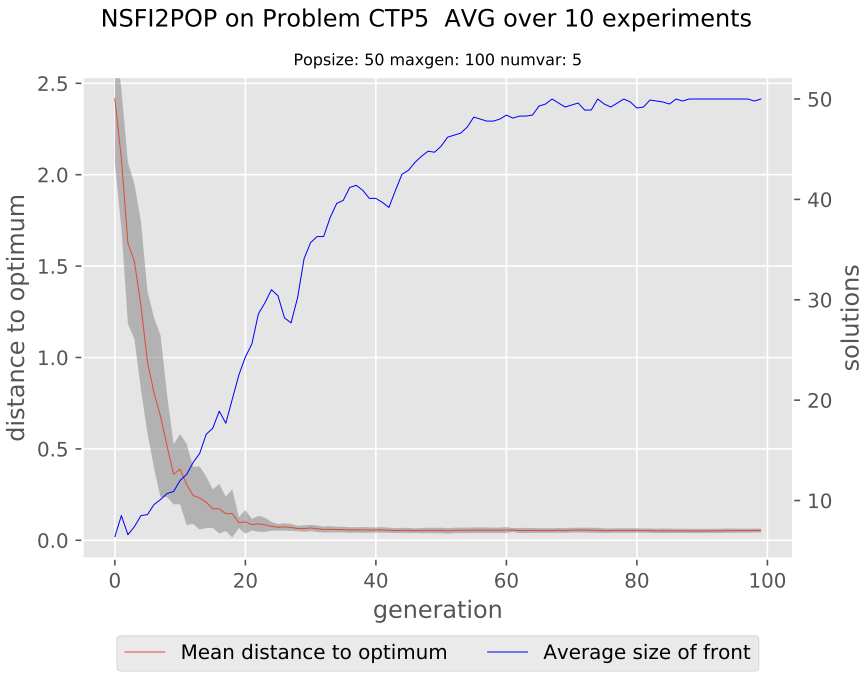


Figure 34: Global Pareto-optimal front distance development of CTP2 with NSFI-2Pop with 5 solution variables

their offspring become feasible but less or more fit than the feasible solutions' offspring. An example of this difference can be seen in fig. 34 and fig. 35.

50 Variables, 50 Population, 100 Generations

The algorithm generally handles the increase in solution variables no different than NSGA-II. It neither performs better or worse.

5 Variables, 250 Population, 500 Generations

The increase in the population does not adversely affect the performance of the algorithm either.

NSFI-2Pop Conclusion

NSFI2-pop has good performance in finding the global Pareto-optimal front. It struggles when faced with 50 variable solutions, but so does the other algorithms in the same experiments. The difference in convergence time with a larger population is minimal. In these tests a population increase of 500% results in a convergence time that is the same, making the algorithm quite effective with large populations as well. In general, it performs almost equivalently with NSGA-II, but does have a slightly larger difference between best and worst performance of a generation.

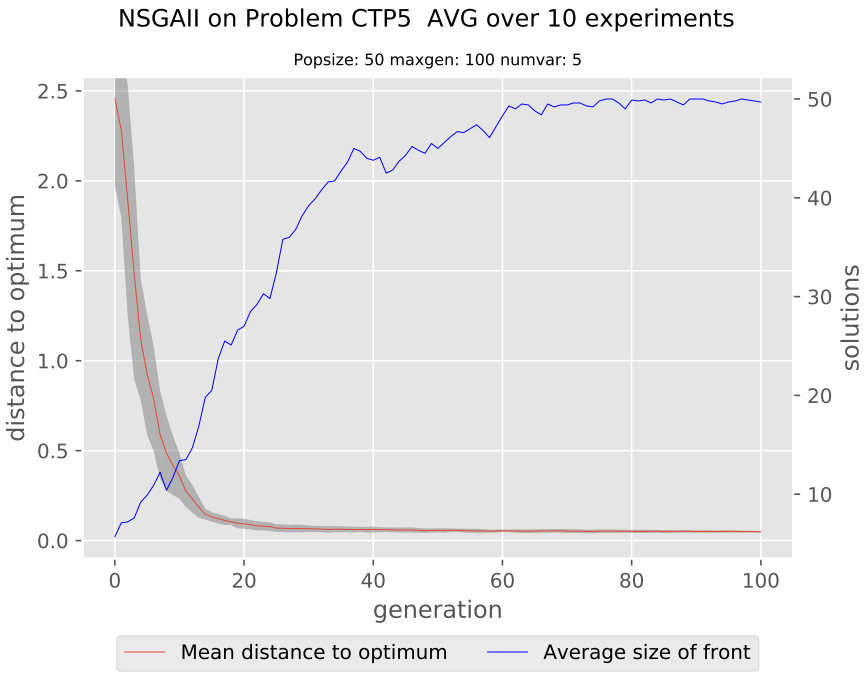


Figure 35: Global Pareto-optimal front distance development of CTP2 with NSGA-II with 5 solution variables

5.2.5 Distribution Measure Analysis

The distribution of NSFI-2Pop and NSGA-II in most of the test problems are very similar. This is expected because NSFI-2Pop is partly based on NSGA-II, but also because it is seen in the convergence graphs and solution plots that the final generations are similar. The distribution of Ray's is mostly greater than the rest of the algorithms.

The NaN values in the table are because there only is one value in the Pareto-front. It is not possible to calculate the distance between one point and nothing. This analysis is based on table 5, table 6 and table 7.

		NSEA	NSFI 2POP	NSGAII	RAYS H	RAYS M	RAYS U
CTP1	std	NaN	0.238	0.237	0.257	0.287	NaN
	var	NaN	0.001	0.001	0.005	0.004	NaN
CTP2	std	0.393	0.257	0.302	1.008	0.887	1.037
	var	0.099	0.003	0.001	0.555	0.443	0.657
CTP3	std	0.555	0.283	0.279	0.985	1.190	0.797
	var	0.123	0.005	0.002	0.244	0.253	0.471
CTP4	std	0.152	0.265	0.280	1.093	1.038	0.774
	var	0.038	0.009	0.007	0.136	0.839	0.094
CTP5	std	0.133	0.244	0.268	1.197	1.032	0.696
	var	0.020	0.008	0.003	0.267	0.218	0.138
CTP6	std	NaN	0.735	0.729	0.673	0.828	0.614
	var	NaN	0.000	0.003	0.091	0.026	0.152
CTP7	std	0.621	0.264	0.265	1.056	1.506	NaN
	var	0.139	0.005	0.006	0.363	0.462	NaN

Table 5: Overview of the mean of the standard deviation and their variances for all experiments with a population size of 50, a generation count of 100 and the number of variables 5

		NSEA	NSFI 2POP	NSGAII	RAYS H	RAYS M	RAYS U
CTP1	std	NaN	0.150	0.147	0.175	0.160	NaN
	var	NaN	0.000	0.001	0.003	0.002	NaN
CTP2	std	0.301	0.324	0.280	0.499	0.555	0.451
	var	0.002	0.004	0.005	0.022	0.014	0.007
CTP3	std	0.343	0.288	0.330	0.628	0.558	0.493
	var	0.004	0.003	0.007	0.066	0.011	0.011
CTP4	std	0.271	0.306	0.265	0.517	0.576	0.547
	var	0.003	0.005	0.007	0.022	0.002	0.022
CTP5	std	0.288	0.291	0.309	0.492	0.534	0.495
	var	0.005	0.006	0.003	0.008	0.006	0.014
CTP6	std	0.469	NaN	0.208	0.196	0.276	0.303
	var	0.011	NaN	0.008	0.024	0.019	0.023
CTP7	std	0.321	0.402	0.322	0.535	0.434	0.465
	var	0.016	0.017	0.020	0.058	0.021	0.015

Table 6: overview of the mean of the standard variation and their variances for all experiments with a population size of 50, a generation count of 100 and the number of variables 50

		NSEA	NSFI 2POP	NSGAII	RAYS H	RAYS M	RAYS U
CTP1	std	NaN	0.261	0.260	0.293	0.312	0.216
	var	NaN	0.000	0.000	0.000	0.001	0.025
CTP2	std	0.038	0.295	0.295	0.981	0.788	1.188
	var	0.006	0.000	0.000	0.295	0.110	0.361
CTP3	std	0.222	0.349	0.327	1.130	0.792	0.860
	var	0.020	0.002	0.001	0.168	0.151	0.097
CTP4	std	NaN	0.336	0.327	1.036	1.042	0.872
	var	NaN	0.001	0.002	0.289	0.100	0.082
CTP5	std	0.000	0.285	0.292	1.465	1.191	1.072
	var	0.000	0.000	0.000	0.616	0.156	0.342
CTP6	std	NaN	0.710	0.709	0.820	0.840	0.945
	var	NaN	0.000	0.000	0.011	0.016	0.021
CTP7	std	0.713	0.315	0.315	1.380	1.293	1.014
	var	0.170	0.000	0.000	0.361	0.317	0.178

Table 7: overview of the mean of the standard variation and their variances for all experiments with a population size of 250, a generation count of 500 and the number of variables 5

5.3 Comparisons

In this subsection the biggest differences in the algorithms' performances will be investigated.

5.3.1 CTP1

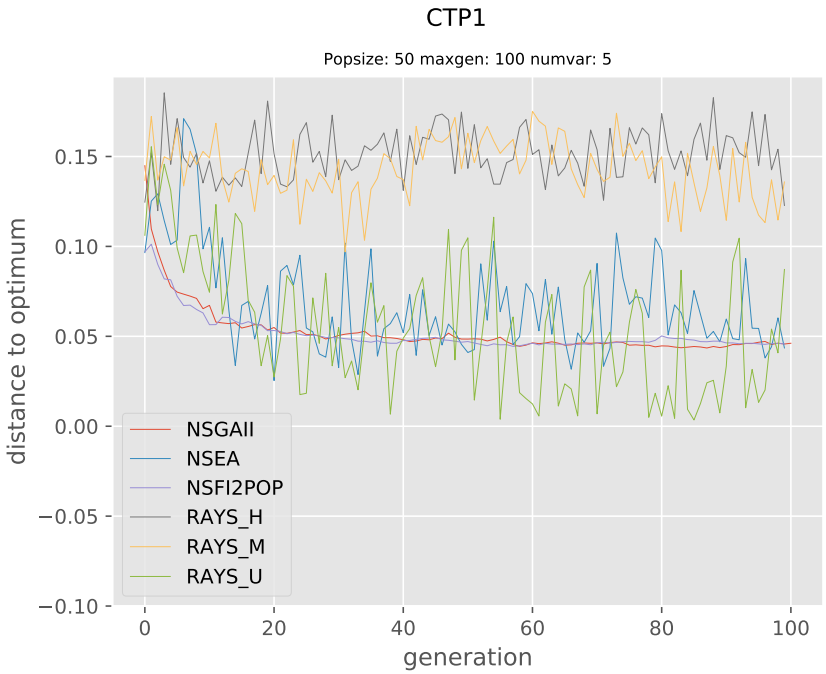


Figure 36: The average distance to the global Pareto-optimal front for CTP1 with 5 variables, 100 generations and 50 population

In fig. 36, all algorithms are plotted for CTP1. Ray's unconstrained is not as good as the figure says, because some of the solutions could be placed in the infeasible region as described in section 5.2.3. NSGA-II and NSFI-2Pop performs well in CTP1, but NSEA also comes near the global Pareto-optimal front. However, NSEA does not provide as stable evolution as NSGA-II and NSFI-2Pop, which can be concluded from the oscillating behaviour of NSEA. Furthermore, the number of solutions found by NSEA is much less than for the two other algorithms and the solutions are not as distributed. When looking at the plot where the number of solution variables is 50 (fig. 37), it can be seen that NSEA actually comes closer to the global Pareto-optimal front, but it is still not as distributed over the front as NSGA-II and NSFI-2Pop.

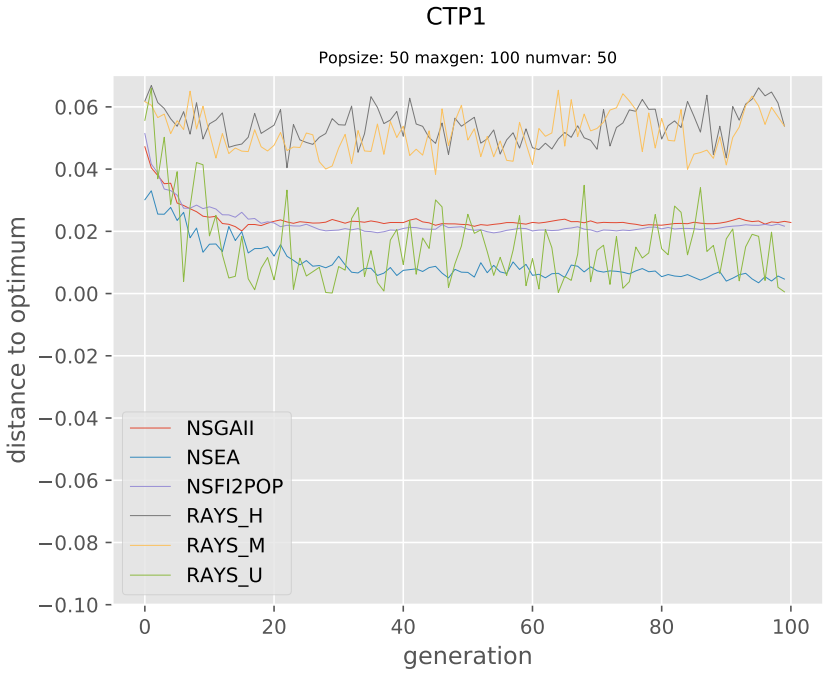


Figure 37: The average distance to the global Pareto-optimal front for CTP1 with 50 variables, 100 generations and 50 population

5.3.2 CTP2, CTP3 & CTP5

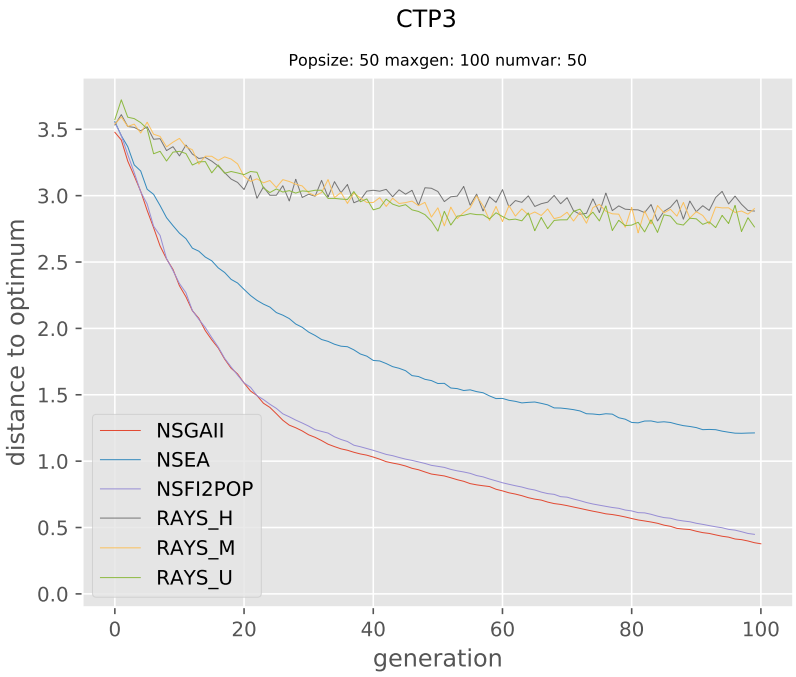


Figure 38: The average distance to the global Pareto-optimal front for CTP3 with 50 variables, 100 generations and 50 population

Across these 3 problems, the performance of all algorithms is very similar. NSGA-II and NSFI-2Pop have an almost identical performance, but they are also the best performing algorithms. NSEA has a better performance than Ray's, but a worse performance than the two other algorithms. This again shows that NSGA-II and NSFI-2Pop are better than the rest, but NSEA is also keeping up although it is more uncertain and has worse distribution. The same situation appears in the experiment with 50 solution variables. Generally, CTP3 provides difficulty in that the Pareto-optimal front's sections are smaller and harder to reach, resulting in a bit more difficulty for NSEA than in CTP2. This is also reflected in NSEA's standard deviation being higher than in CTP2, see table 5. When the number of solution variables is increased, NSEA's distance from the Pareto-optimal front does not show signs of oscillation as is otherwise the case, see fig. 38. This suggests that NSEA oscillates when it is nearing an optimal solution. This is further backed by the standard deviation and variance values, which are very similar to NSGA-II and NSFI-2Pop for the same problem, see table 6.

5.3.3 CTP4

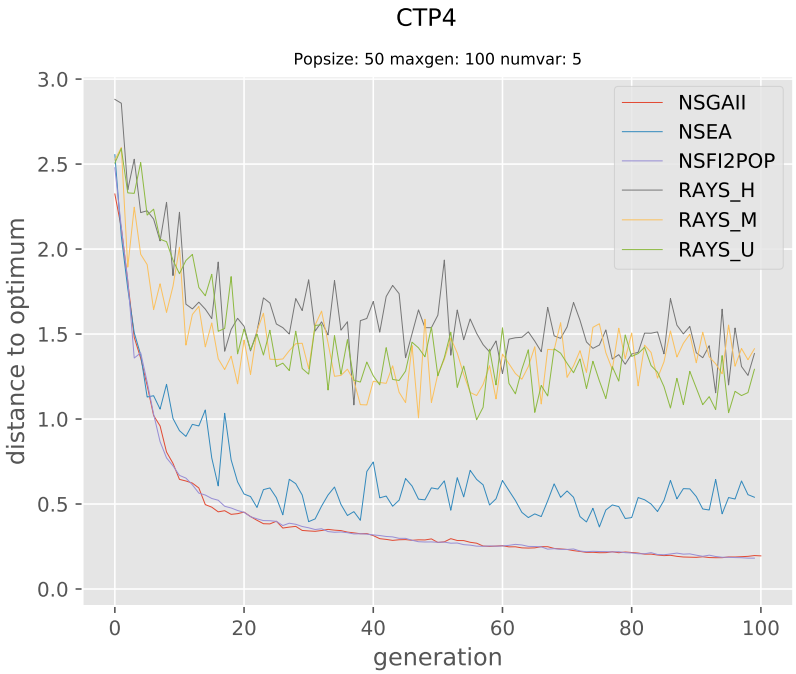


Figure 39: The average distance to the global Pareto-optimal front for CTP4 with 5 variables, 100 generations and 50 population

Looking at fig. 39 the same situation is conveyed as in the previous test problems. It is still NSGA-II and NSFI-2Pop that performs best. When the number of solution variables is increased, NSFI-2Pop in one experiment has a final result that is worse than any of the NSGA-II results (NSFI-2Pop result can be seen in fig. 40 and NSGA-II result can be seen in fig. 41). In general, all algorithms find CTP4 with 50 solution variables too difficult to find solutions on the global Pareto-optimal front.

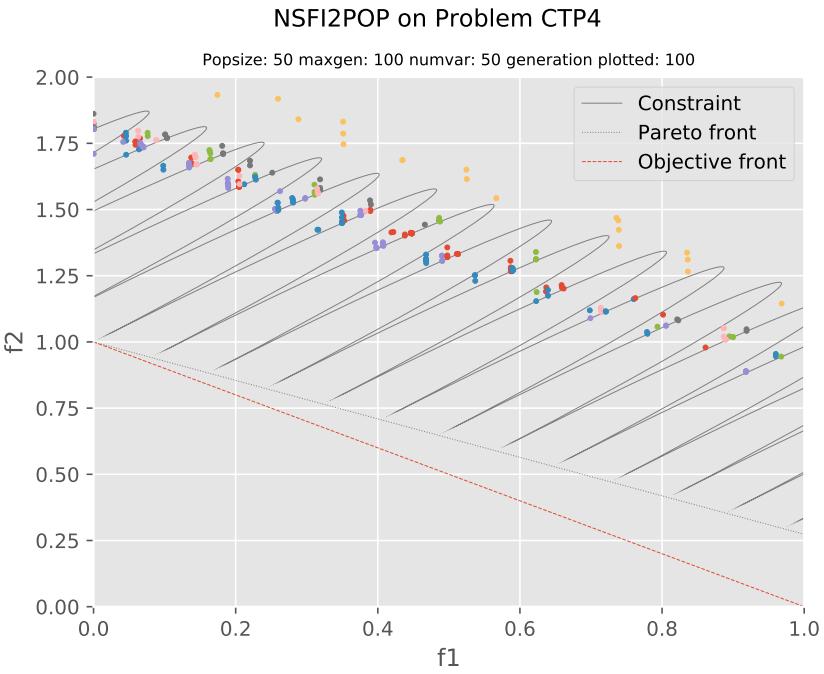


Figure 40: Solution plot of NSFI-2Pop with 50 variables on CTP4

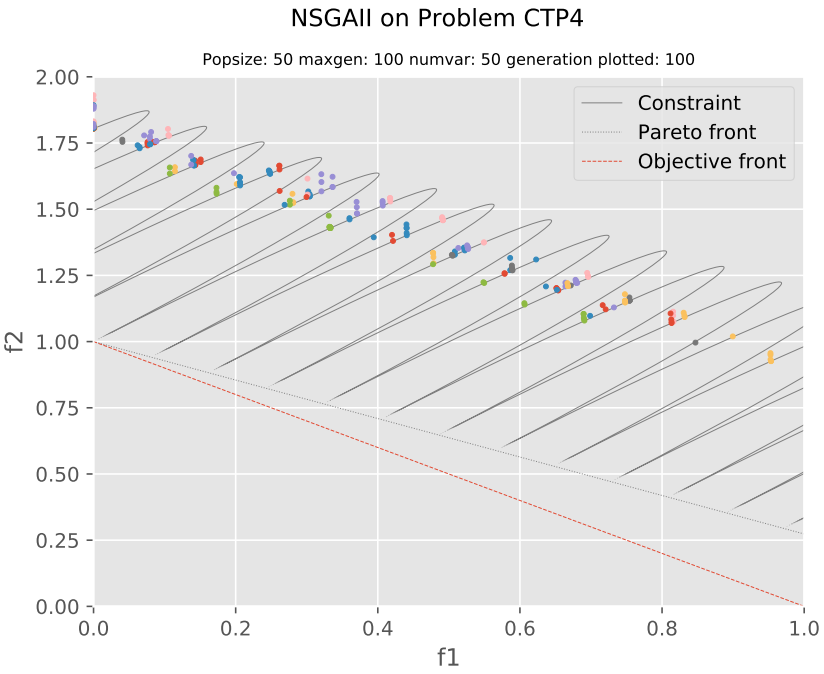


Figure 41: Solution plot of NSGA-II with 50 variables on CTP4

5.3.4 CTP6

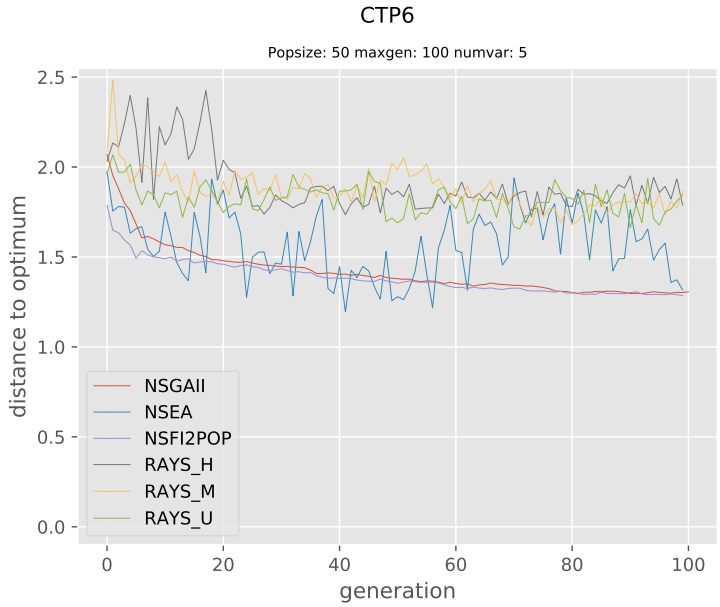


Figure 42: The average distance to the global Pareto-optimal front for CTP6 with 5 variables, 100 generations and 50 population

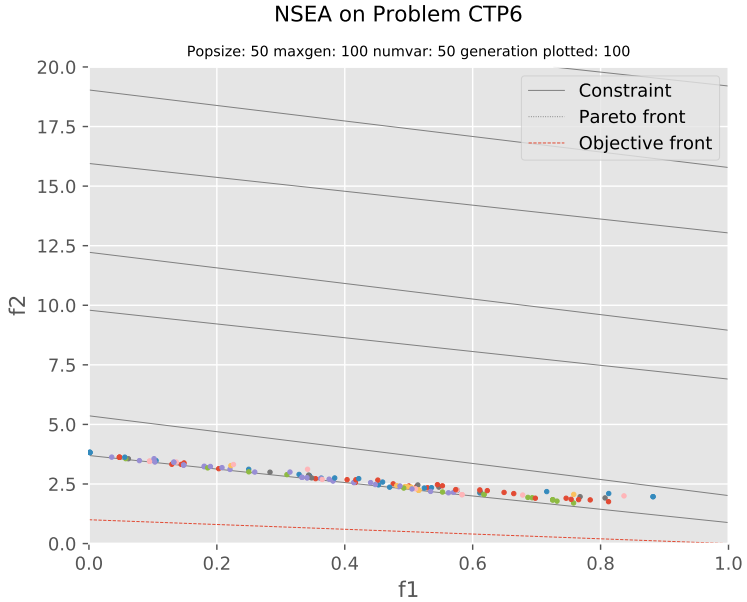


Figure 43: Solution plot of NSEA on CTP6, with 50 variables, 50 population size and 100 generations, at the last generation

NSEA finds really good solutions when looking at fig. 42, but it oscillates a lot, hence it "loses" the good solutions again. NSGA-II and NSFI-2Pop maintain their good solutions, hence they are steadily achieving a better result. This is most likely because of their implementation of elitism. When the number of generations is larger, NSFI-2Pop and NSGA-II find a lot of solutions on the global Pareto-optimal front, but NSEA also finds several solutions in the same area. When the amount of solution variables is increased, NSEA comes closer to the Pareto-optimal front than NSGA-II and it has a better distribution (the standard deviation is twice as large as NSGA-II), as seen in fig. 43. This shows that NSEA is capable of finding better solutions when applied to problems with large infeasible regions that must be crossed to find an optimal solution.

5.3.5 CTP7

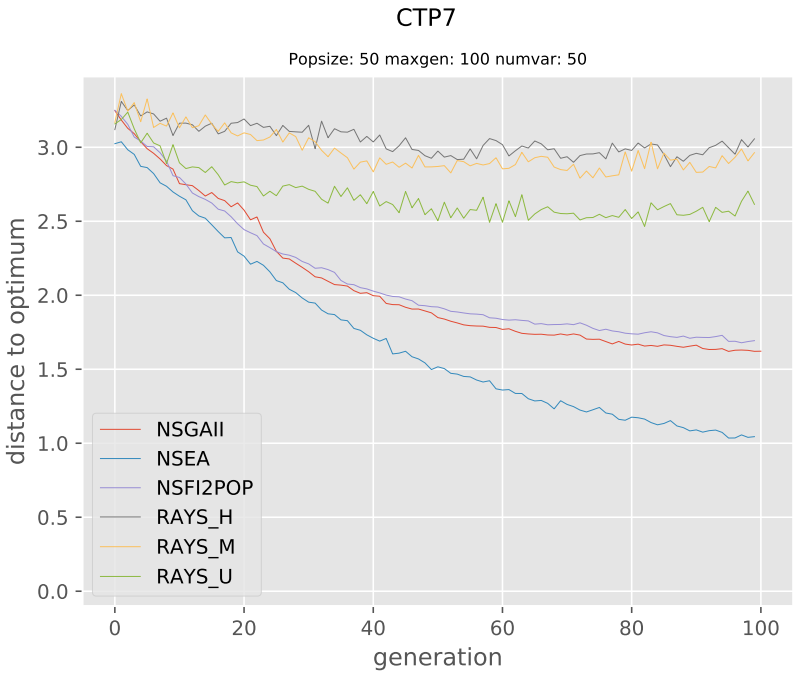


Figure 44: The average distance to the global Pareto-optimal front for CTP7 with 50 variables, 100 generations and 50 population

In fig. 44, NSEA again comes closer to the global Pareto-optimal front than NSGA-II and NSFI-2Pop. The solutions found by the algorithms are however all far from the global Pareto-optimal front. NSEA's solutions are also distributed, as seen in fig. 45. NSGA-II and NSFI-2Pop still finds good solutions when the number of variables is less, which for instance is seen in fig. 46.

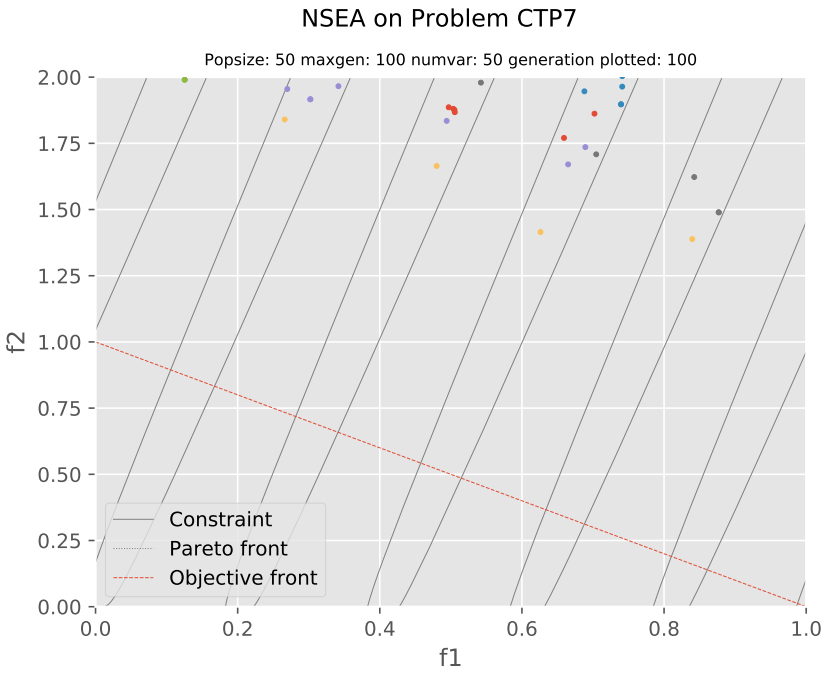


Figure 45: Solution plot of NSEA in CTP7

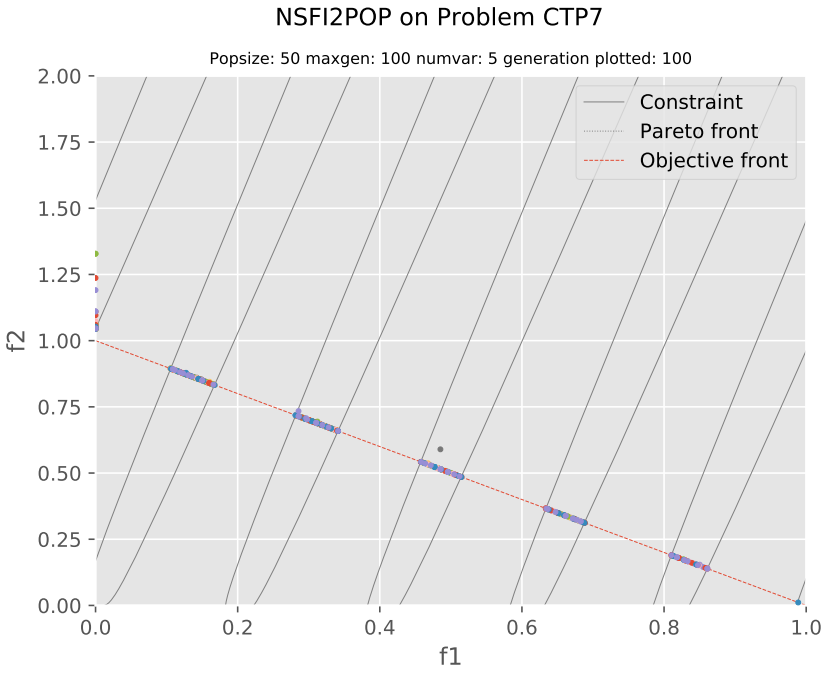


Figure 46: Solution plot of NSFI-2Pop in CTP7

6 Discussion

In this section, we will discuss the considerations implicit in our experiments and measures. Then interesting aspects of the project and subject, suitable for further work will be explored.

NSFI-2Pop, NSGA-II and NSEA have fixed algorithm parameters, but Ray's algorithm is implemented with three different settings. It might be possible to optimize each algorithm to the individual test problems by adjusting parameters in the algorithms. It would be necessary with specific knowledge of how each parameter will influence the performance of the algorithms in the problems, hence it is an entire research project by itself.

The scope of this project is to compare the performance of the four algorithms. The definition of performance influences the results obtained in the project. The performance mainly concerns how close the final Pareto-optimal front is to the global Pareto-optimal front, how good the distribution of the front is and how fast this convergence happens. The rate of convergence is related to the number of generations used and not the computational power or processing time necessary. It could be relevant to scrutinize and compare the performance with that perspective, particularly if a specific constrained multi-objective problem needs to be solved efficiently, but it is outside the scope of this project.

It would be interesting to combine the multiple measures of performance into a single measure of performance so that the algorithms could be compared more easily. The problem of combining the two measures into one coherent measure is in itself a multi-objective problem, which could be solved by scalarizing the objective value into a single objective, as described in Srinivas and Deb [6] and Jiménez and Verdegay [7]. The relative weights of the measures in the coherent measure would be decisive in describing whether a generation has found a good solution or a bad solution. To prevent some (possibly) arbitrary weights from influencing the outcome in a way that is not expected, it is a better decision to analyse each objective individually and in that way, get a more nuanced impression of the performance of the algorithms.

The experiments have been conducted with a various number of generations, population sizes and number of solution variables. The chosen values influence what we are able to see in the results. We might discover new things if the number of generations or population size are increased, but given that we have already tried with different values without a great difference, we might not find new information by trying even more experiment settings. The number of solution variables were maximum 50 in the experiments, which was enough to distinguish two of the algorithms from two of the other algorithms that performed poorly under these conditions. A further experiment with even more solution variables could be conducted to see if the two best performing algorithms would differ in performance. Another way to increase the difficulty of the problems is to implement a new $g(x)$ function in the constrained test problems. A more complex $g(x)$ function could be a function, where the individual weights of the solution variables were changed, for instance the solution variables could be passed through a sine-function before the mean value is found.

All experiments were conducted ten times and the average performances were scrutinized. This was done to make sure that a single "lucky" experiment would not influence the outcome of the experiments too much. Another approach to initial population of the algorithm could have been chosen, where all algorithms use the same initial population. This would reduce the influence of random values in the experiments. The algorithms are based on randomness during execution, but the initial population would not necessarily have to be random.

Three algorithms (NSGA-II, NSEA and Ray's) have been compared to the algorithm NSFI-2Pop. NSEA's and Ray's algorithm's performance is not good compared to NSGA-II and NSFI-2Pop. It would be relevant to compare NSGA-II and NSFI-2Pop to an algorithm like SPEA[5] or NSGA-III[23]. It is relevant to determine whether NSFI-2Pop performs better than the classical algorithms, but it would be even more relevant to compare it to new and popular algorithms. Other test problems could also have been used, which would possibly give different results. However, the CTPs provide a wide range of challenges and it is easy to apply all algorithms to the problems.

7 Conclusion

In this section, the goals of the project(See section 1) will be examined and the project will be evaluated in accordance to how it fulfilled these goals.

The purpose of this project was to compare the performance of the four algorithms by implementing them in the Java framework JMetal and applying our implementations to the constrained test problems described in Deb, Pratap, and Meyarivan [2]. This was done in a series of thirty experiments, divided on three different experiment types.

Our results have shown that different algorithms are best applied in different problems. To sum up our results:

- Jiménez and Verdegay's algorithm NSEA performed best in CTP6 and CTP7 compared to the other algorithms in our experiment with 50 solution variables. NSEA comes closer to the global Pareto-optimal front and has a better distribution.
- NSFI-2Pop performed almost identically to NSGA-II, but with an important difference. The result of NSFI-2Pop was more varied from experiment to experiment in regard to the mean distance from the set of solutions to the global Pareto-optimal front.
- NSGA-II and NSFI-2Pop generally performed better than NSEA and all versions of Ray's algorithm.
- The convergence of solutions towards the global Pareto-optimal front is more stable for NSGA-II and NSFI-2Pop compared to Ray's algorithm and NSEA.

References

[1] M. Scirea, J. Togelius, P. Eklund, and S. Risi, "MetaCompose: A compositional evolutionary music composer", in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, Springer International Publishing, 2016, pp. 202–217. DOI: 10.1007/978-3-319-31008-4_14. [Online]. Available: https://doi.org/10.1007%2F978-3-319-31008-4_14.

[2] K. Deb, A. Pratap, and T. Meyarivan, "Constrained test problems for multi-objective evolutionary optimization", in *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings*, E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 284–298, ISBN: 978-3-540-44719-1. DOI: 10.1007/3-540-44719-9_20. [Online]. Available: http://dx.doi.org/10.1007/3-540-44719-9_20.

[3] J. Durillo, A. Nebro, and E. Alba, "The jMetal framework for multi-objective optimization: Design and architecture", in *CEC 2010*, Barcelona, Spain, Jul. 2010, pp. 4138–4325.

[4] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization", *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011, ISSN: 0965-9978. DOI: DOI:10.1016/j.advengsoft.2011.05.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997811001219.

[5] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II", 2000.

[6] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms", pp. 221–248, 1994.

[7] F. Jiménez and J. L. Verdegay, "Constrained multiobjective optimization by evolutionary algorithms", 1998, pp. 266–271.

[8] T. Ray, K. Tai, and K. C. Seow, "Multiobjective design optimization by an evolutionary algorithm", pp. 399–424, 2000.

[9] A. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2003, ISBN: 978-3-662-05094-1.

[10] ——, "From evolutionary computation to the evolution of things", 2015.

[11] M. Schoenauer. (2002). Variation operators, [Online]. Available: http://eodev.sourceforge.net/eo/tutorial/html/eoOperators.html.

[12] Z. Michalewicz, *GeneticAlgorithms + Data Structures = Evolution Programs*. Springer, 1992, ISBN: 3-540-60676-9. [Online]. Available: http://zeus.inf.ucv.cl/~bcrawford/MII-748-METAHEURISTICAS/Michalewicz%20Z.pdf.

[13] S. Baluja and R. Caruana, "Removing the genetics from standard genetic algorithm", 1995.

[14] S. O. Kimbrough, G. J. Koehler, M. Lu, and D. H. Wood, "On a feasible–infeasible two-population (FI-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch", *European Journal of Operational Research*, vol. 190, no. 2, pp. 310–327, Oct. 2008. DOI: 10.1016/j.ejor.2007.06.028. [Online]. Available: https://doi.org/10.1016%2Fj.ejor.2007.06.028.

[15] A. J. Nebro, J. J. Durillo, M. Vergne, and J. Billingsley. (2017). Jmetal documentation, [Online]. Available: https://github.com/jMetal/jMetalDocumentation (visited on 05/06/2017).

[16] A. J. Nebro. (2017). Jmetal/abstractevolutionaryalgorithm.java, [Online]. Available: https://github.com/jMetal/jMetal/blob/b6cb51f77e4ce1cd5e0d1cc868e698d6b299c373/jmetal-core/src/main/java/org/uma/jmetal/algorithm/impl/AbstractEvolutionaryAlgorithm.java (visited on 04/22/2017).

[17] C.-S. Tsou. (2013). Elitist non-dominated sorting genetic algorithm based on r, [Online]. Available: https://cran.r-project.org/web/packages/nsga2R/nsga2R.pdf.

[18] F.-M. D. Rainville, F.-A. Fortin, M.-A. Gardner, M. Parizeau, and C. Gagné. (2012). Deap: A python framework for evolutionary algorithms, [Online]. Available: http://vision.gel.ulaval.ca/~cgagne/pubs/deap-gecco-2012.pdf.

[19] A. J. Nebro, J. J. Durillo, and M. Vergne. (2017). Jmetal 5 website, [Online]. Available: http://jmetal.github.io/jMetal/ (visited on 04/21/2017).

[20] A. J. Nebro. (2017). Jmetal/nsgaii.java, [Online]. Available: https://github.com/jMetal/jMetal/blob/90c8ced9e5a26c173f135bb558a9e15fda4bcbe8/jmetal-algorithm/src/main/java/org/uma/jmetal/algorithm/multiobjective/nsgaii/NSGAII.java (visited on 05/12/2017).

[21] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance", *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, Mar. 2012. DOI: 10.1016/j.physa.2011.12.004. [Online]. Available: https://doi.org/10.1016%2Fj.physa.2011.12.004.

[22] E. W. Weisstein. (2017). Point-line distance–2-dimensional, [Online]. Available: http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html.

[23] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints", *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, Aug. 2014. DOI: 10.1109/tevc.2013.2281535. [Online]. Available: https://doi.org/10.1109%2Ftevc.2013.2281535.

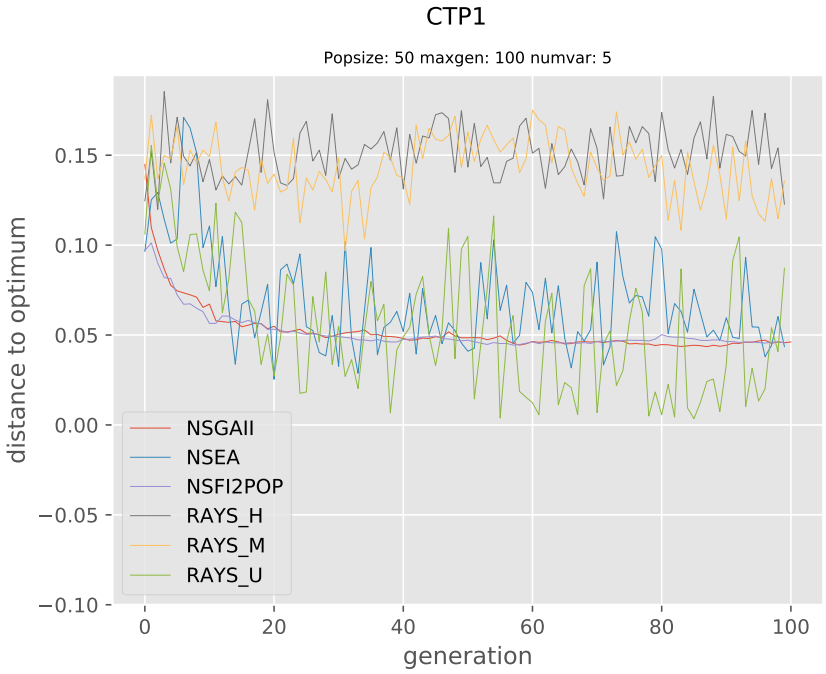
List of Figures

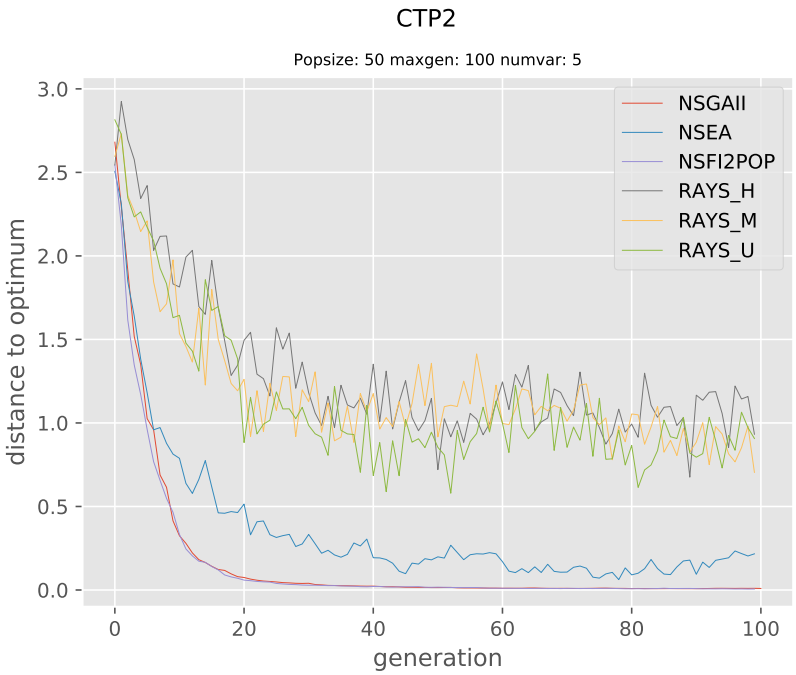
List of Tables

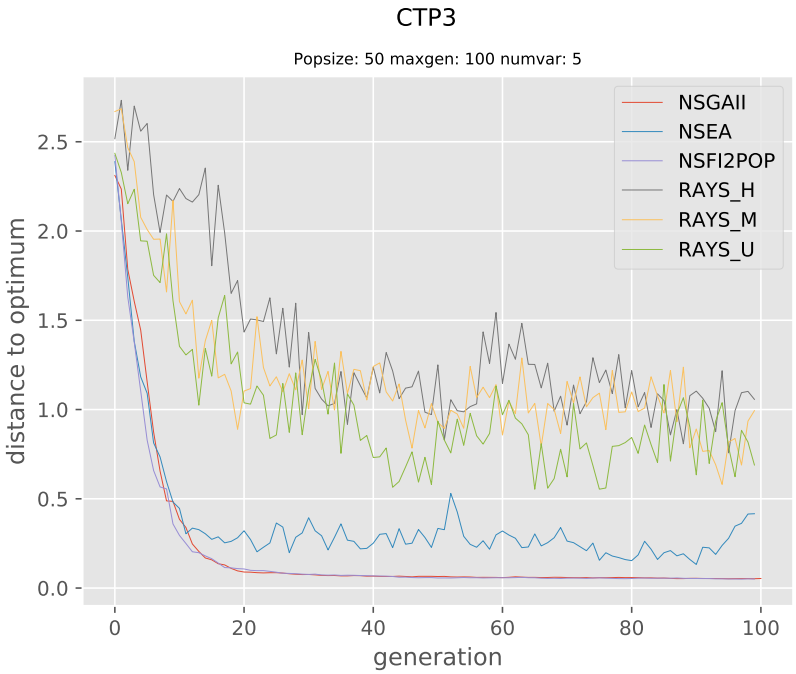
A Appendixes

A.1 Distance to Pareto-front comparisons

A.1.1 Variables 50 generations 100 population 50



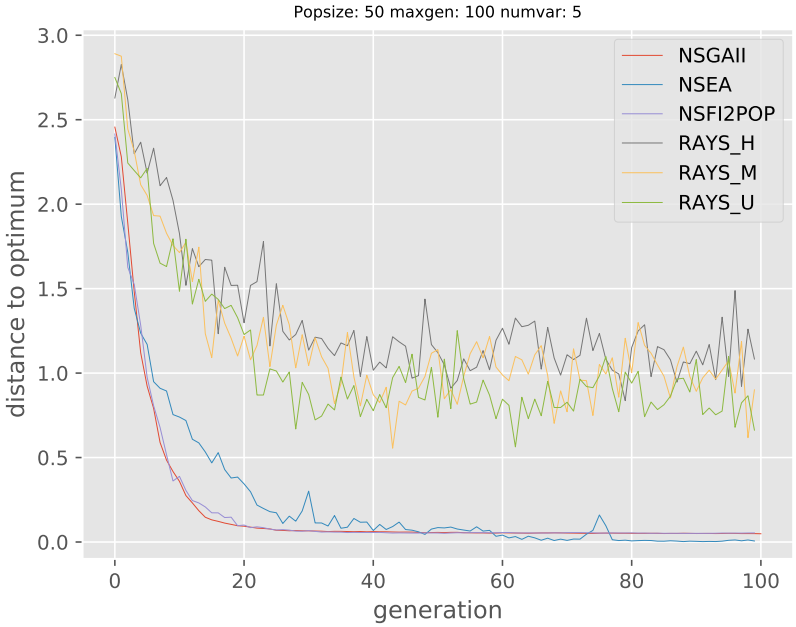

CTP2
Popsize: 50 maxgen: 100 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

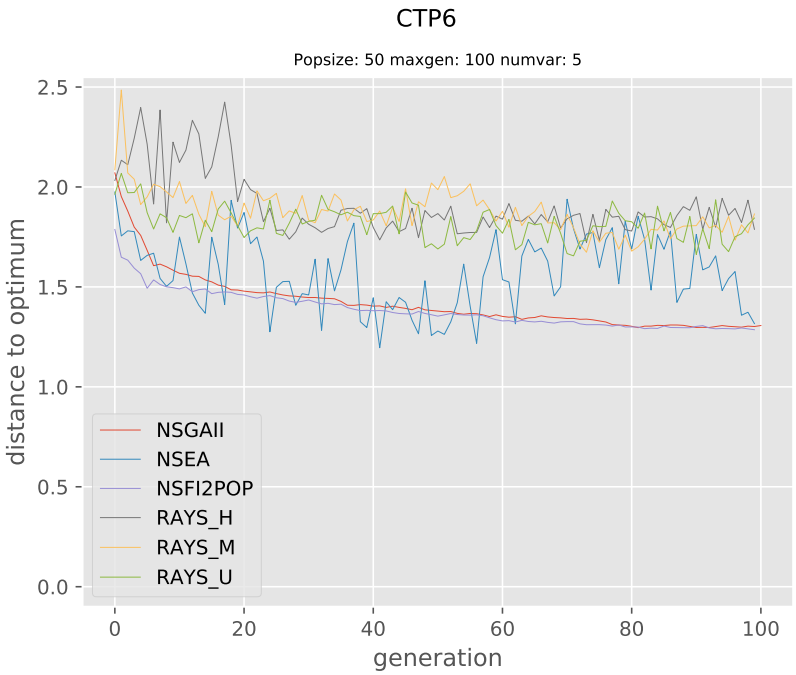

CTP3
Popsize: 50 maxgen: 100 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

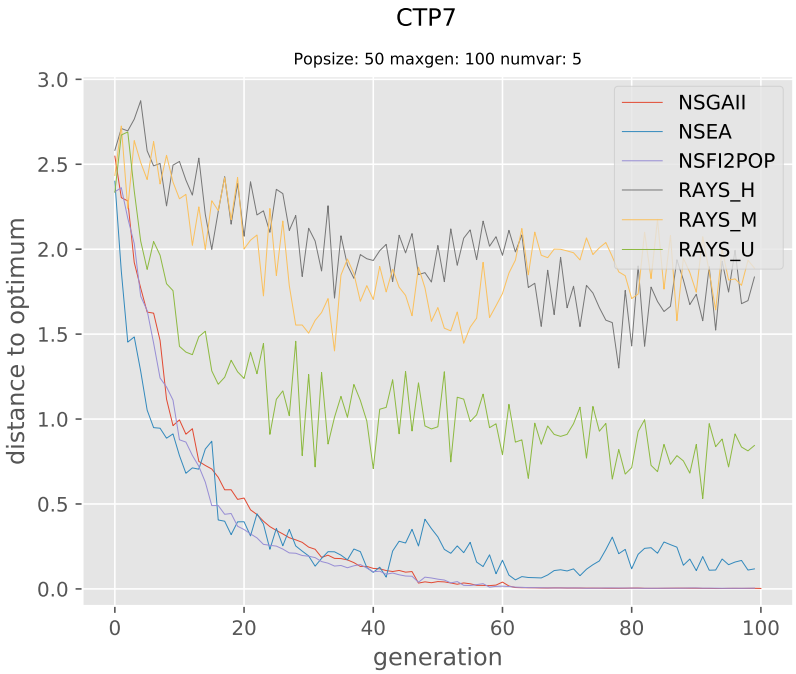
CTP4


Popsize: 50 maxgen: 100 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

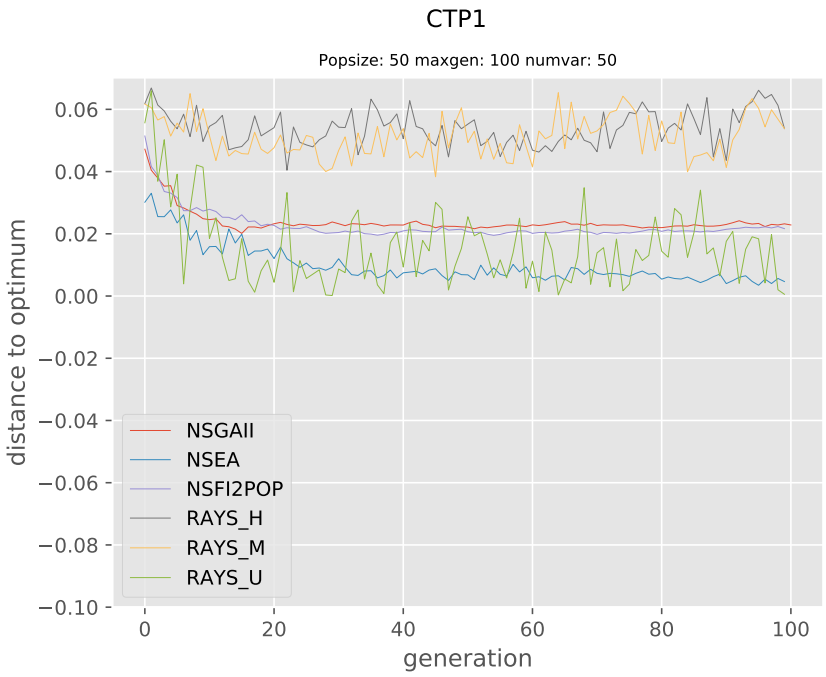
CTP5

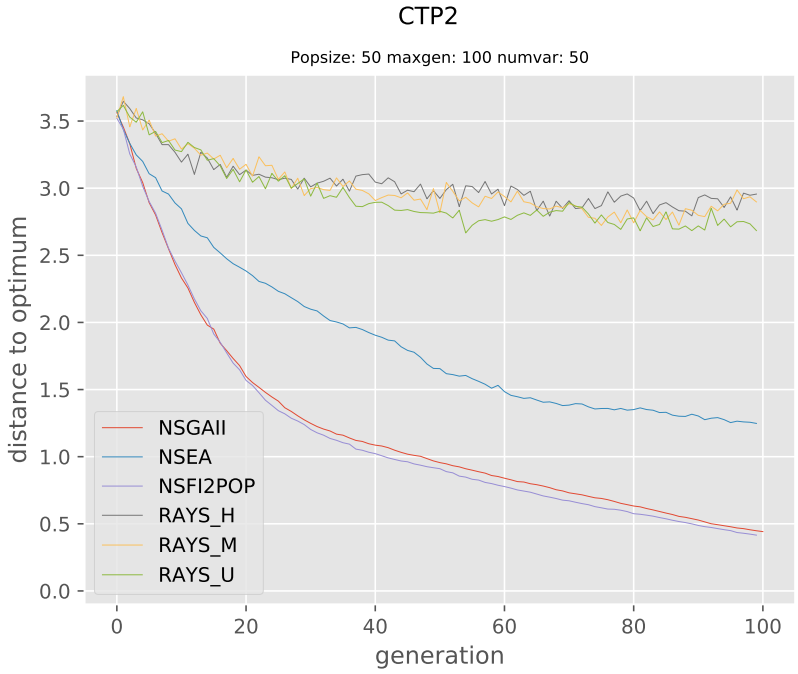

Popsize: 50 maxgen: 100 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100


CTP6
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U

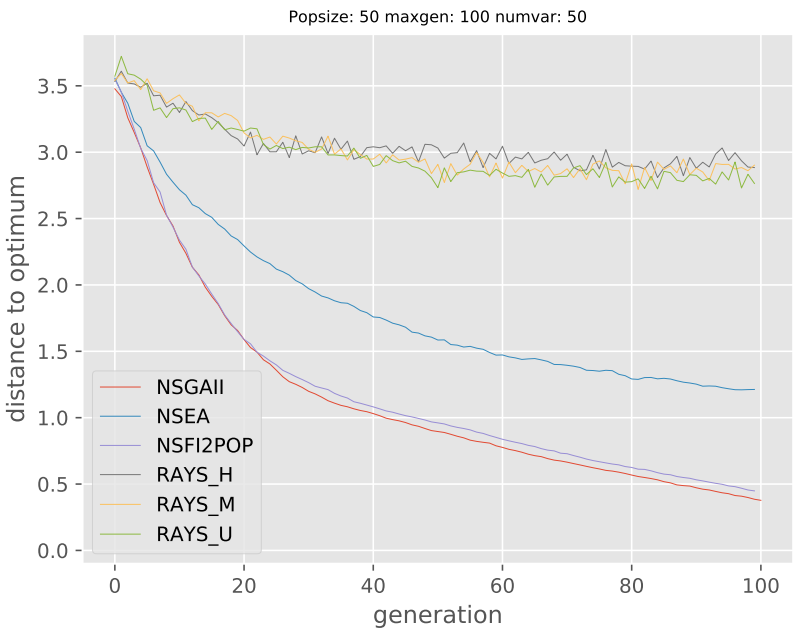

CTP7
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U

A.1.2 Variables 50 generations 100 population 50

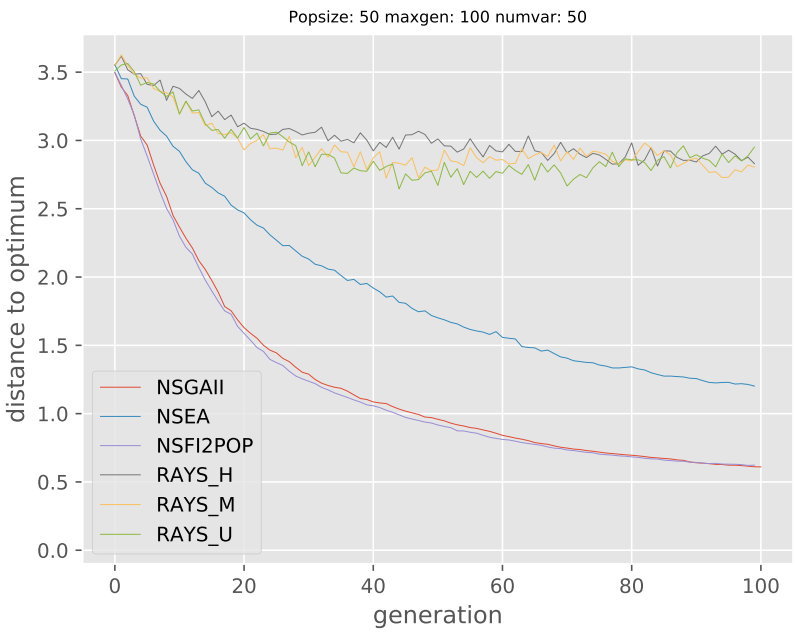


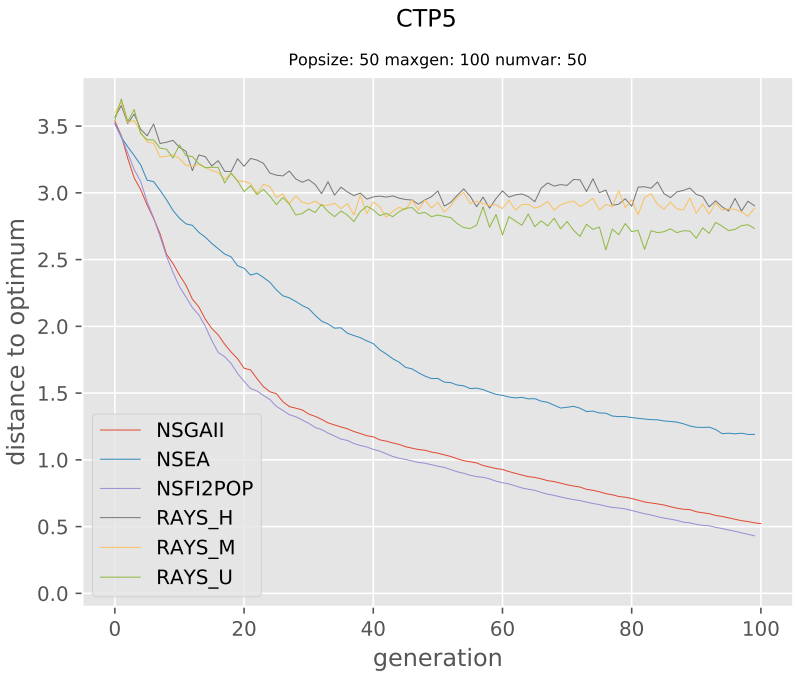


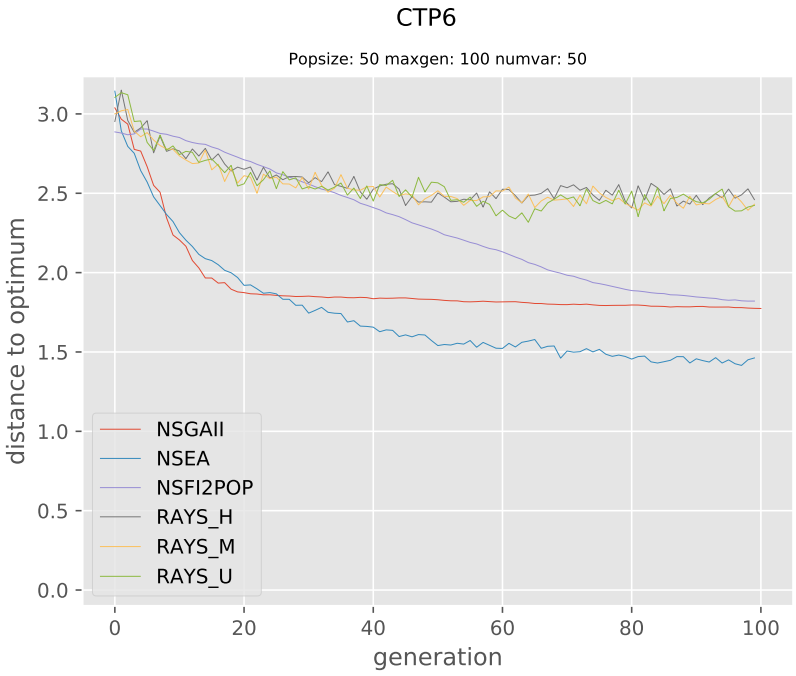
CTP3

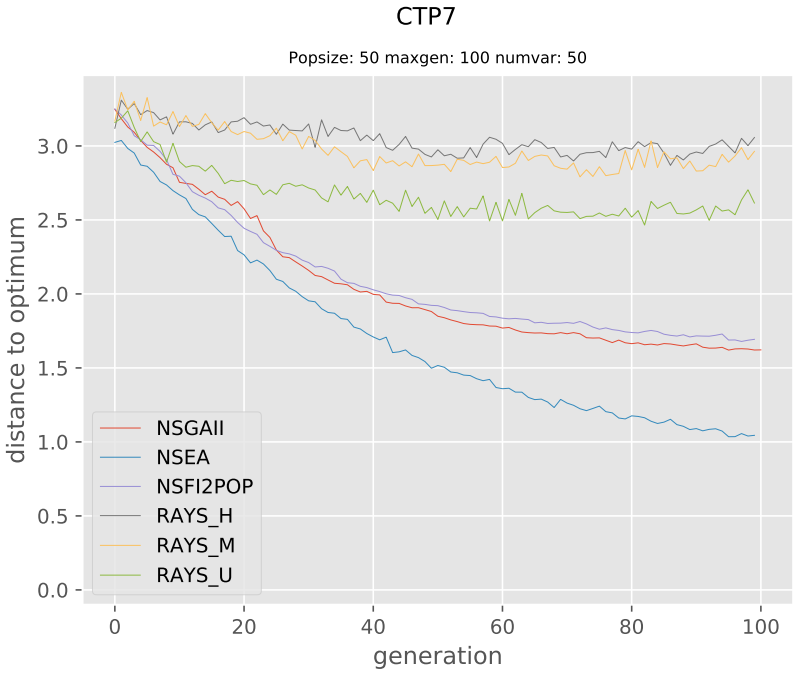

Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

CTP4

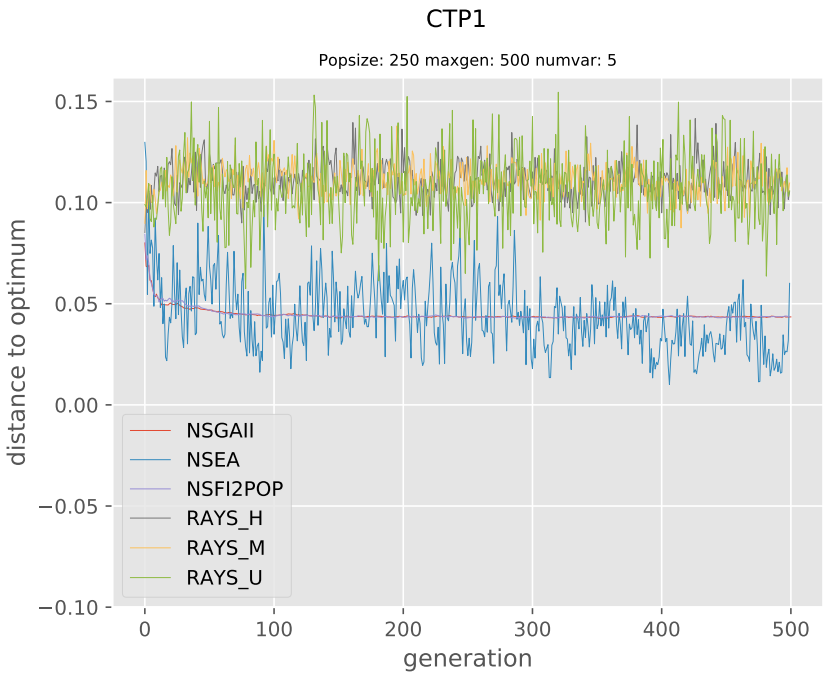

Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

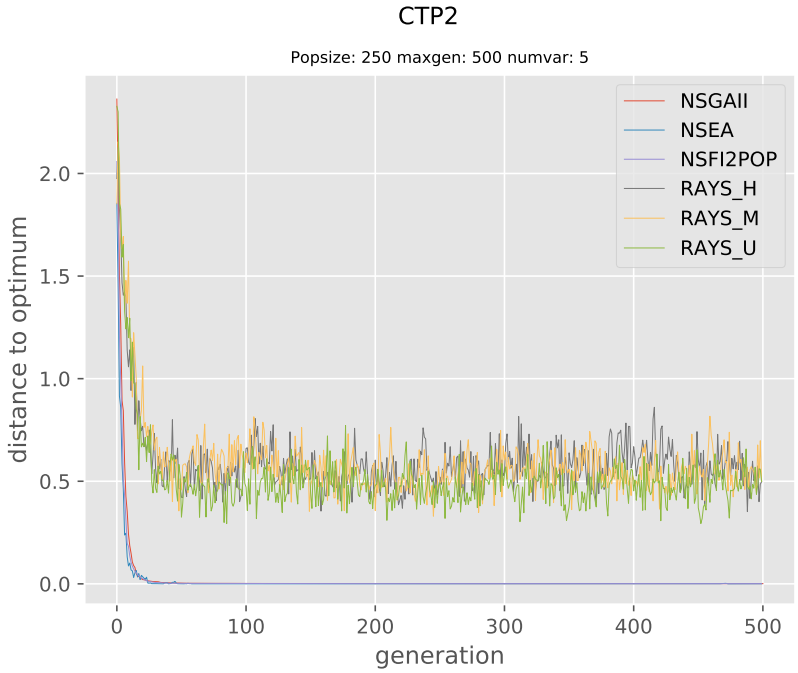

CTP5
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100


CTP6
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

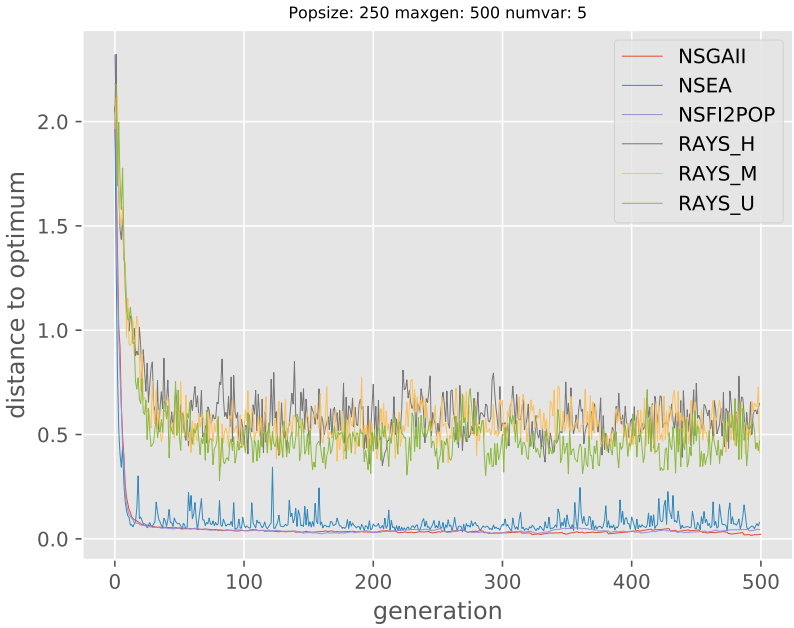

CTP7
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U

A.1.3 Variables 5 generations 500 population 250

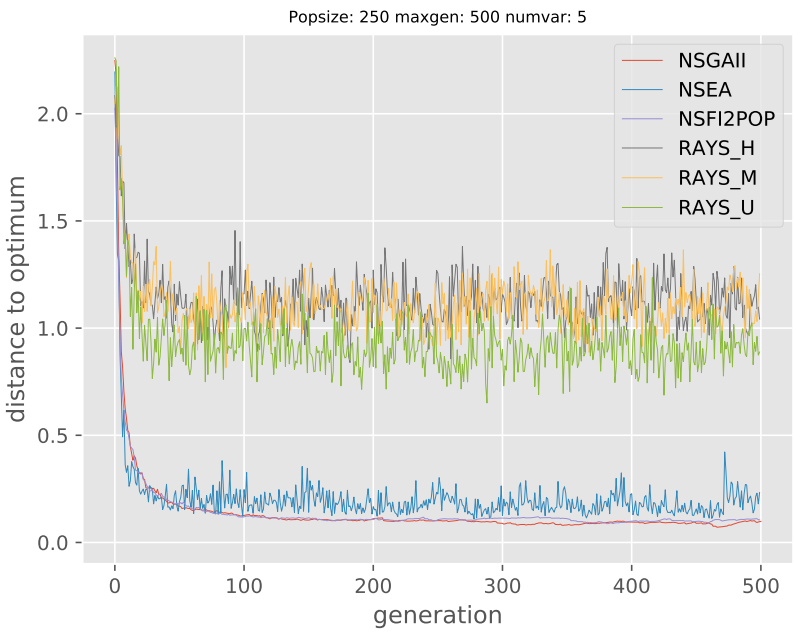


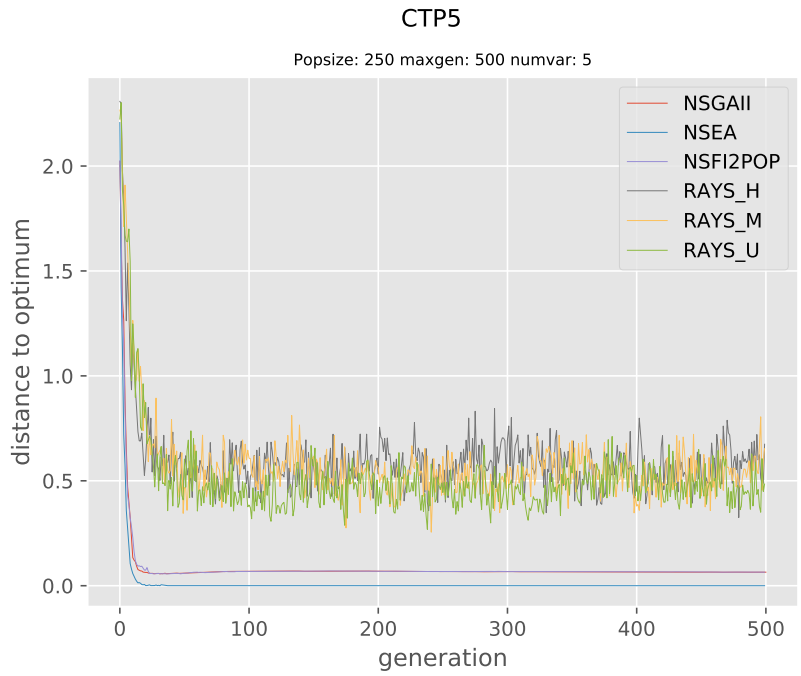


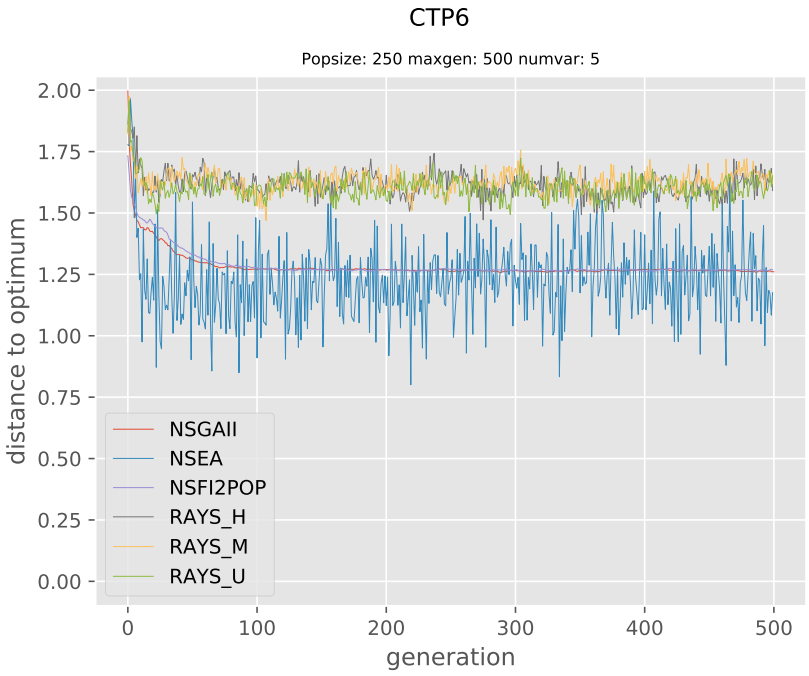
CTP3

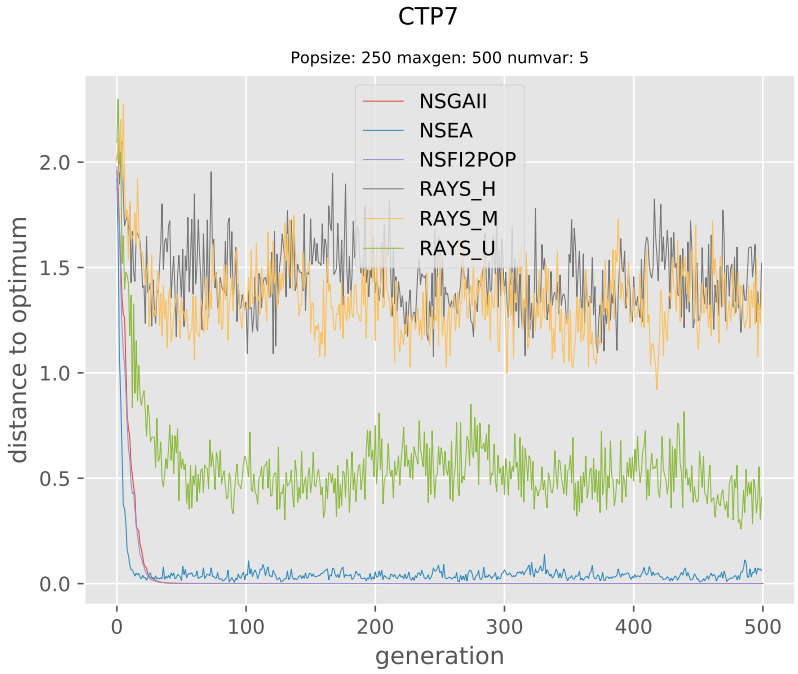

Popsize: 250 maxgen: 500 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
0.0
0.5
1.0
1.5
2.0
0
100
200
300
400
500
generation

CTP4


Popsize: 250 maxgen: 500 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
0.0
0.5
1.0
1.5
2.0
0
100
200
300
400
500
generation

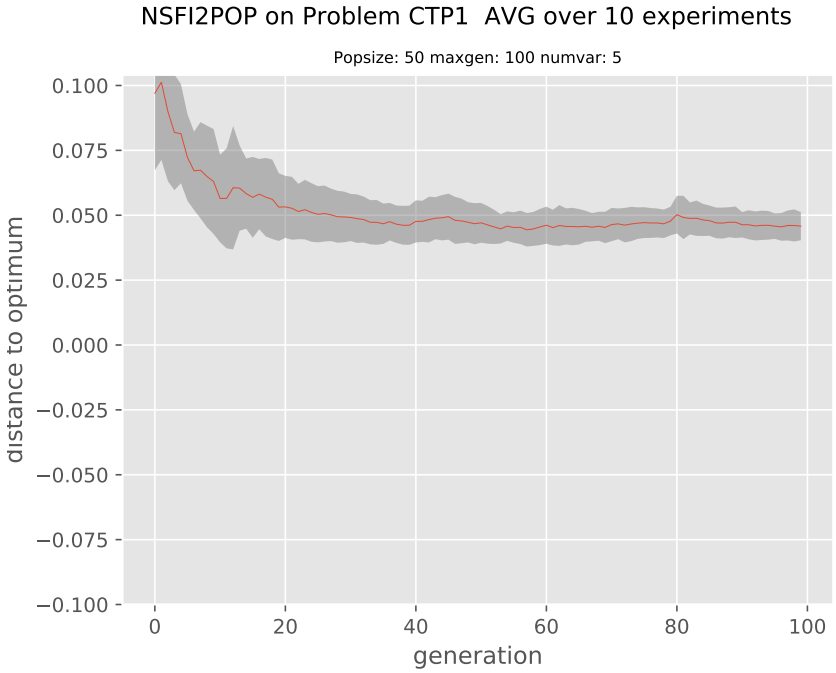

CTP5
Popsize: 250 maxgen: 500 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
0
100
200
300
400
500

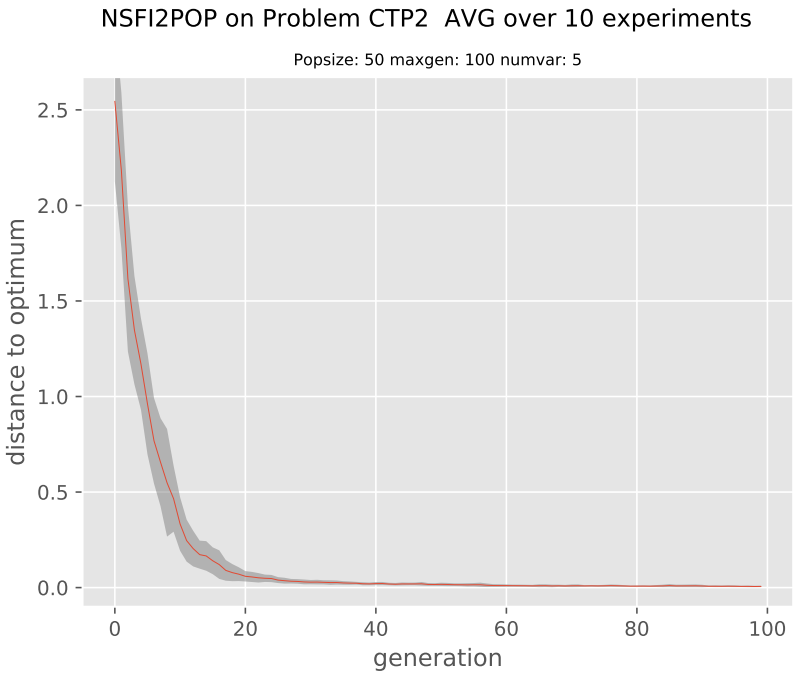

CTP6
Popsize: 250 maxgen: 500 numvar: 5
NSGAII
NSEA
NSFI2POP
RAYS_H
RAYS_M
RAYS_U
distance to optimum
generation
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
100
200
300
400
500

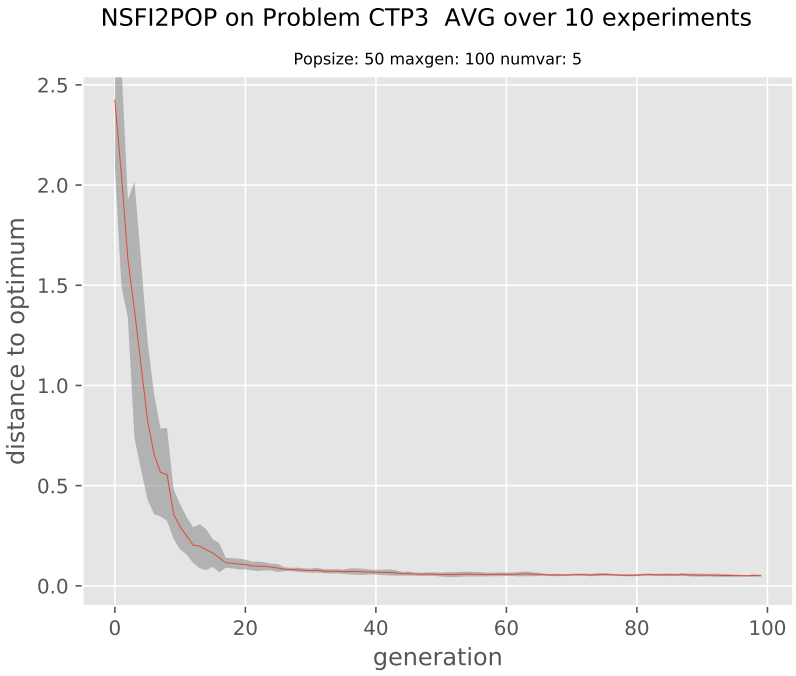


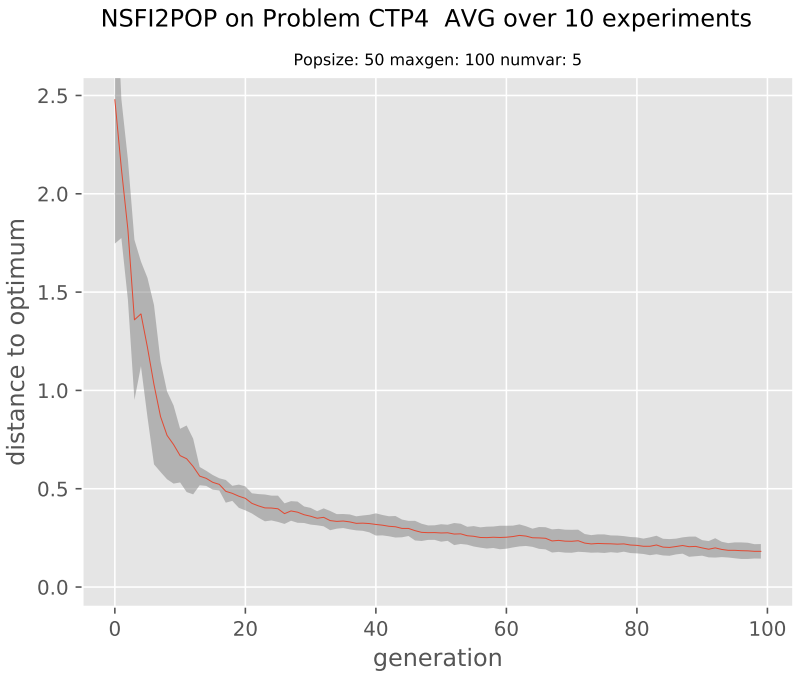
A.2 NSFI2POP plots

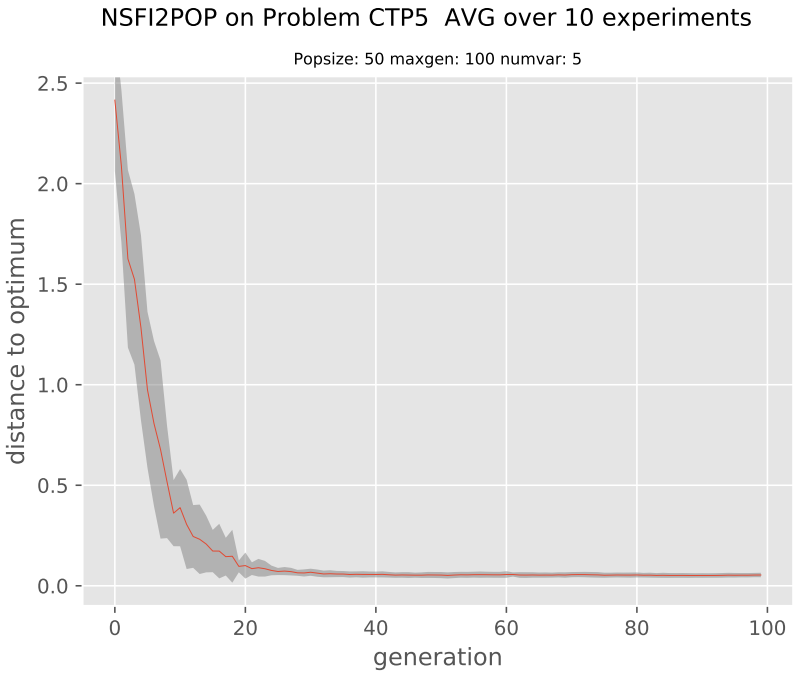
A.2.1 variables 5, generations 100, population 50

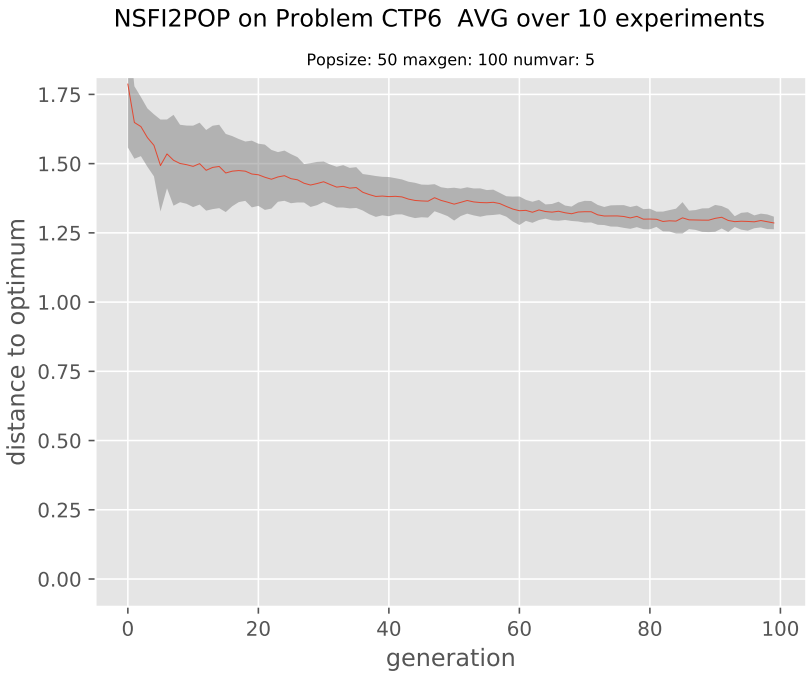


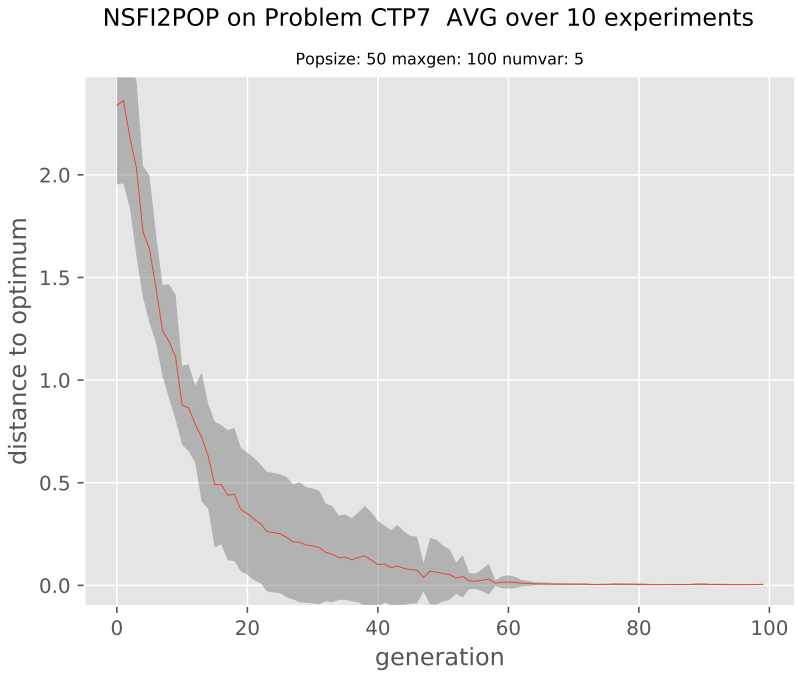

NSFI2POP on Problem CTP2 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation


NSFI2POP on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

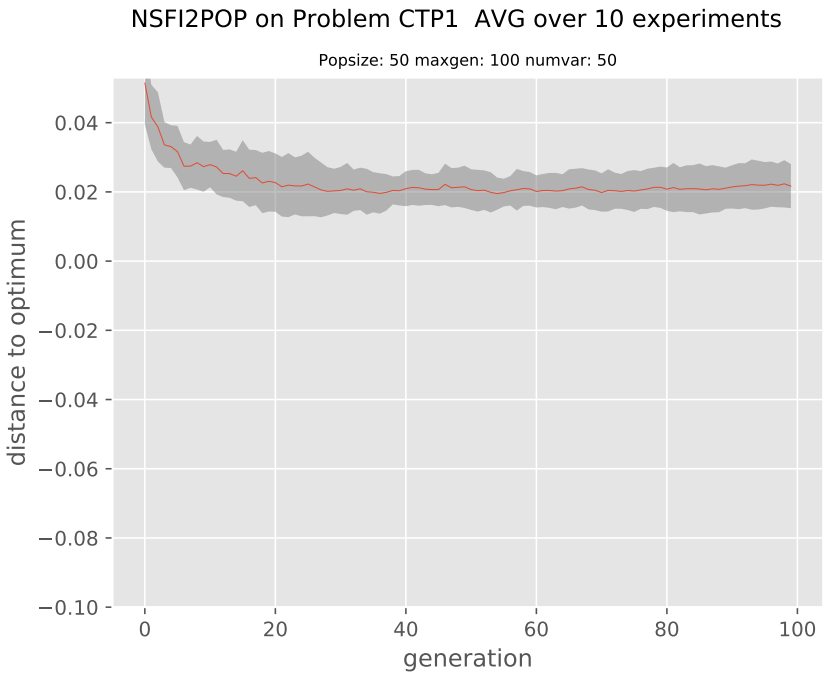

NSFI2POP on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

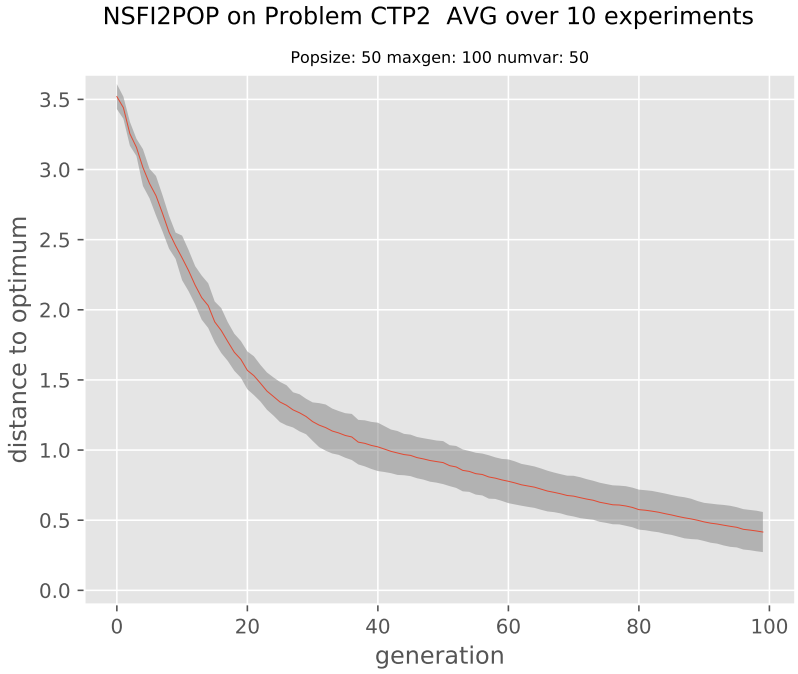

NSFI2POP on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

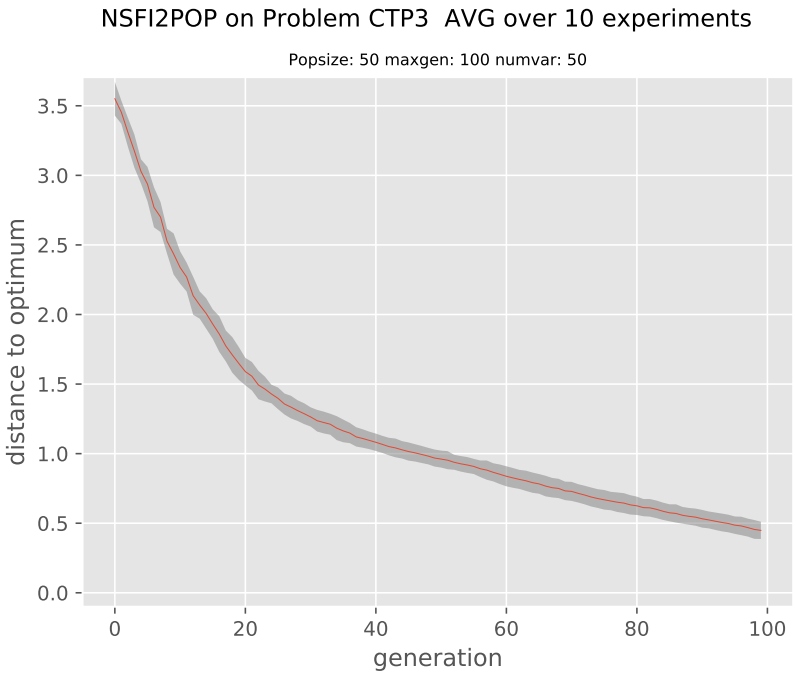

NSFI2POP on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
generation

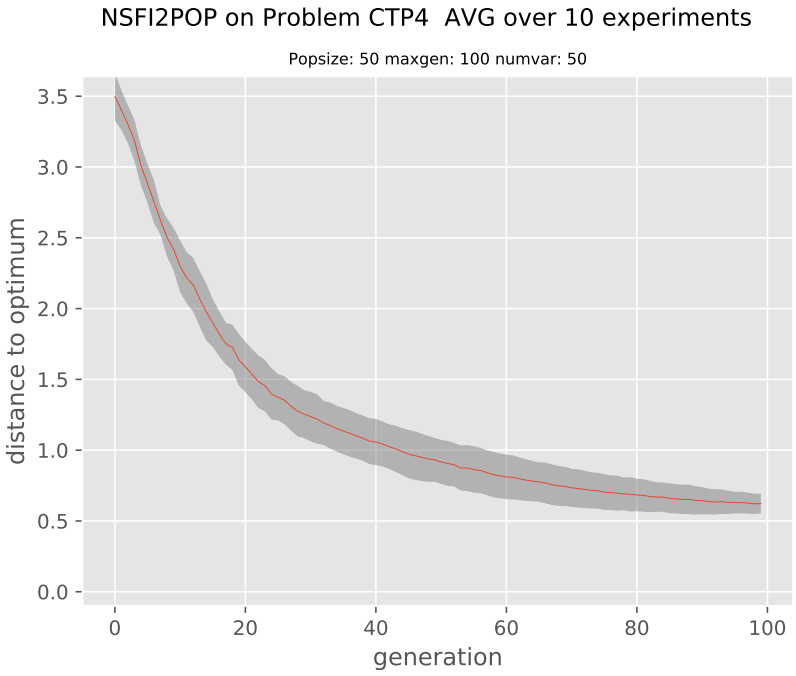

NSFI2POP on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

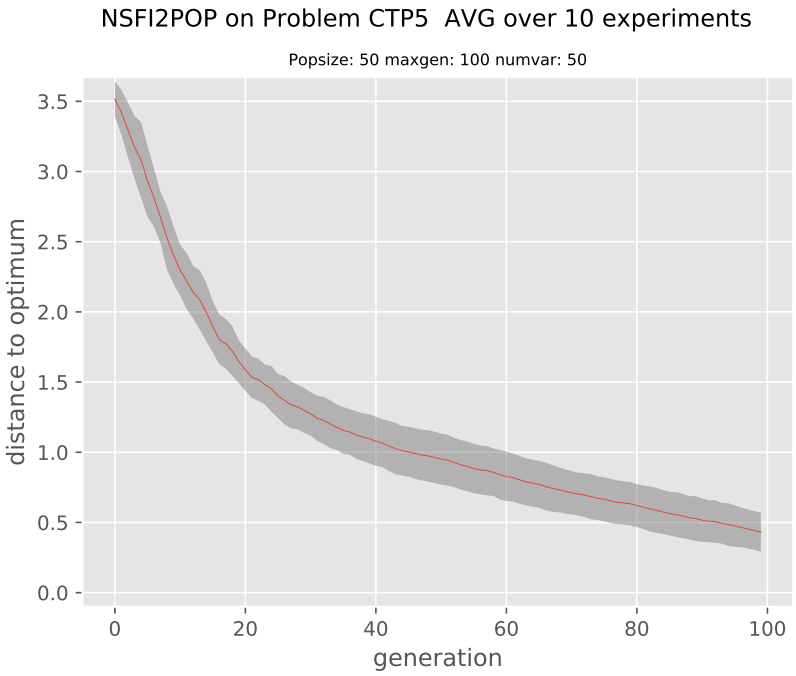
A.2.2 variables 50, generations 100, population 50

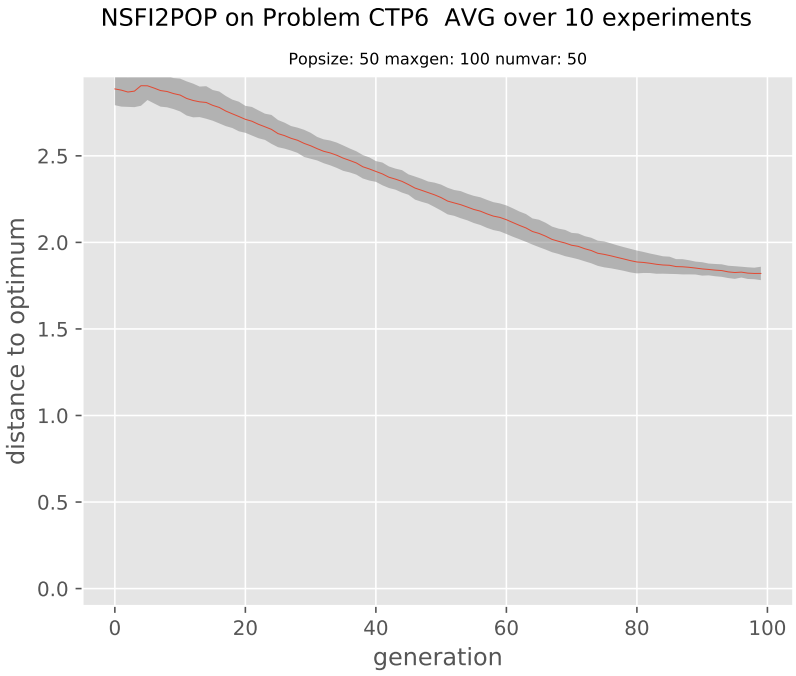


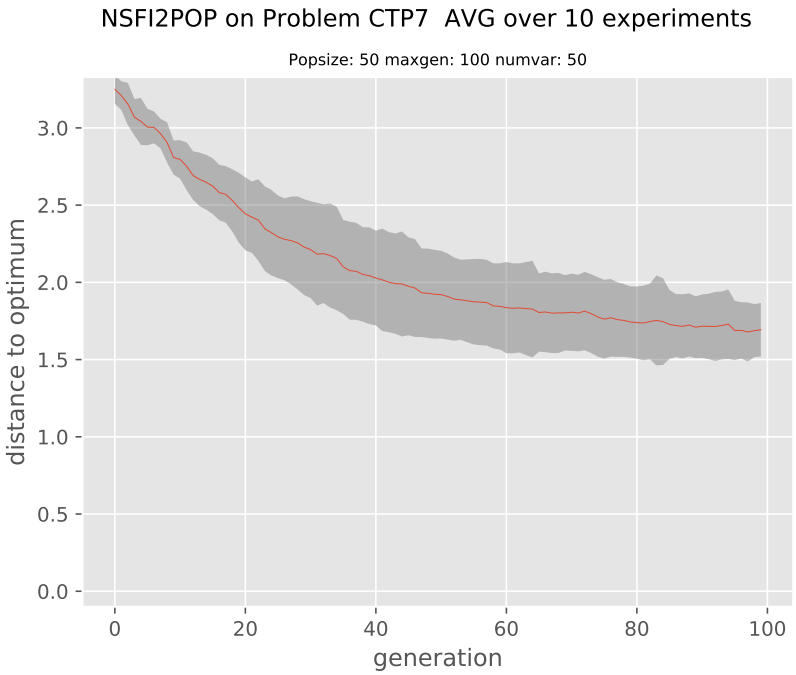



NSFI2POP on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

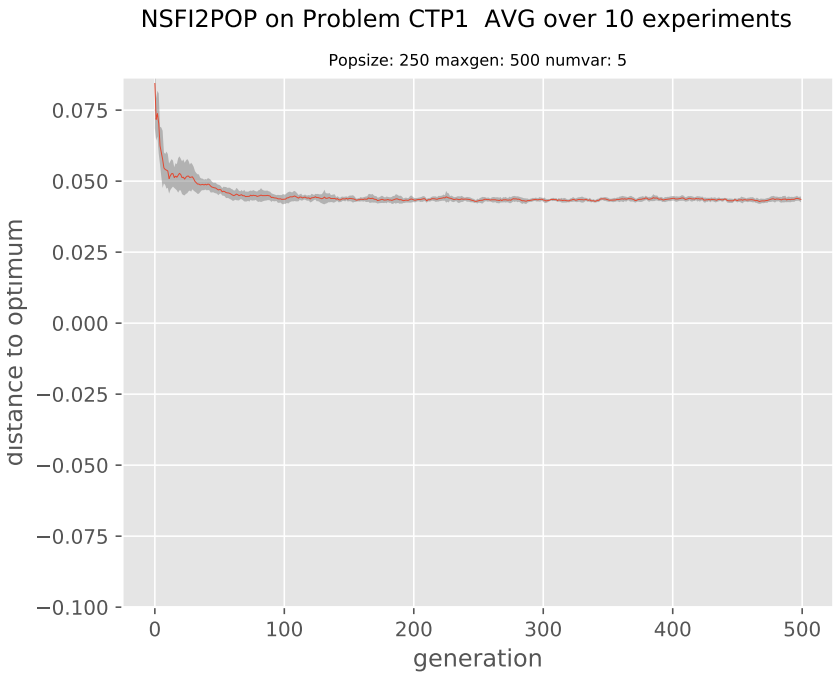

NSFI2POP on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

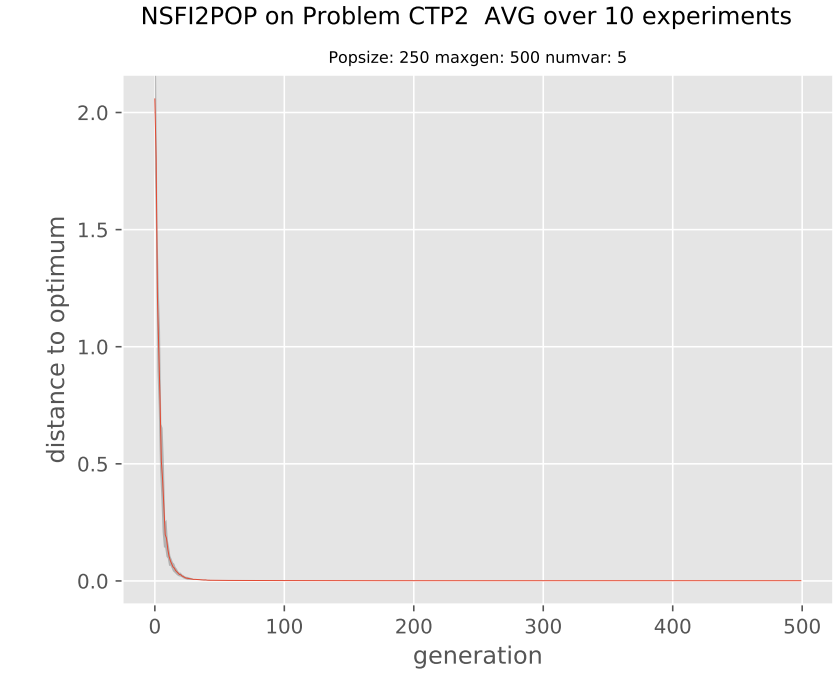

NSFI2POP on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

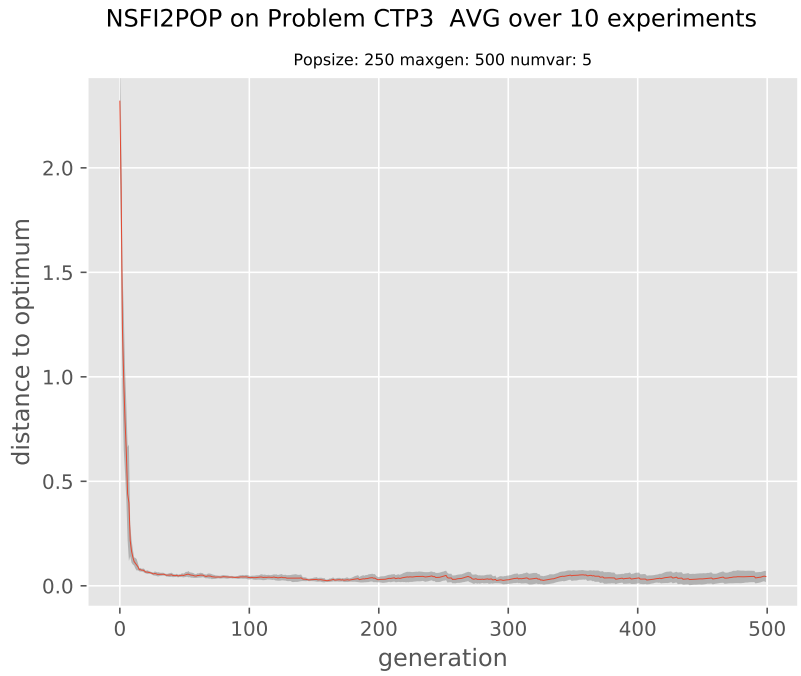

NSFI2POP on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

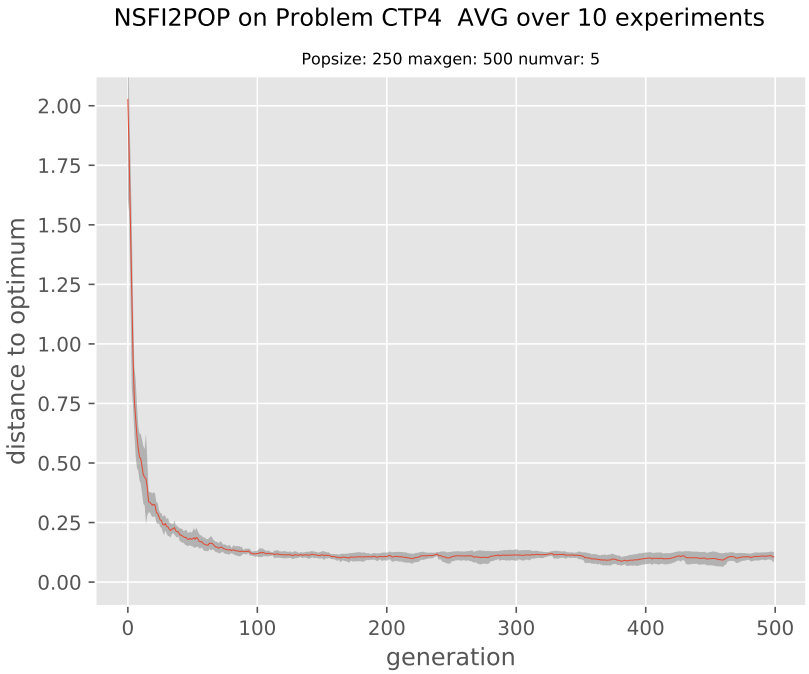

NSFI2POP on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

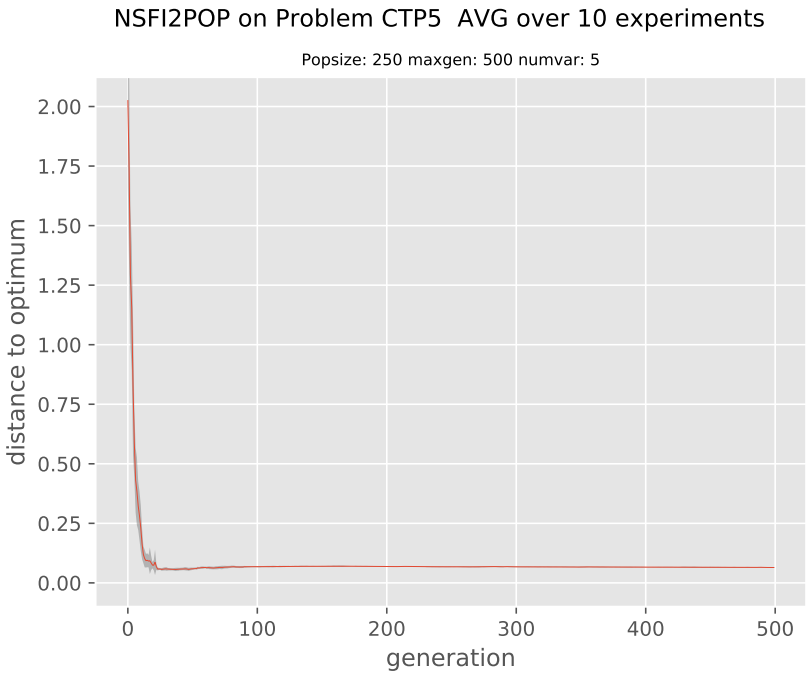
A.2.3 variables 5, generations 500, population 5

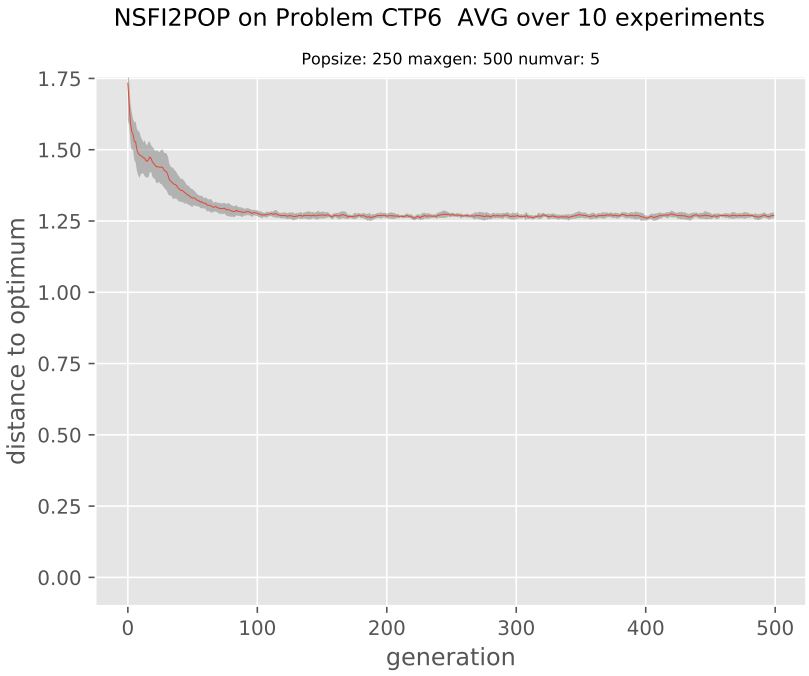


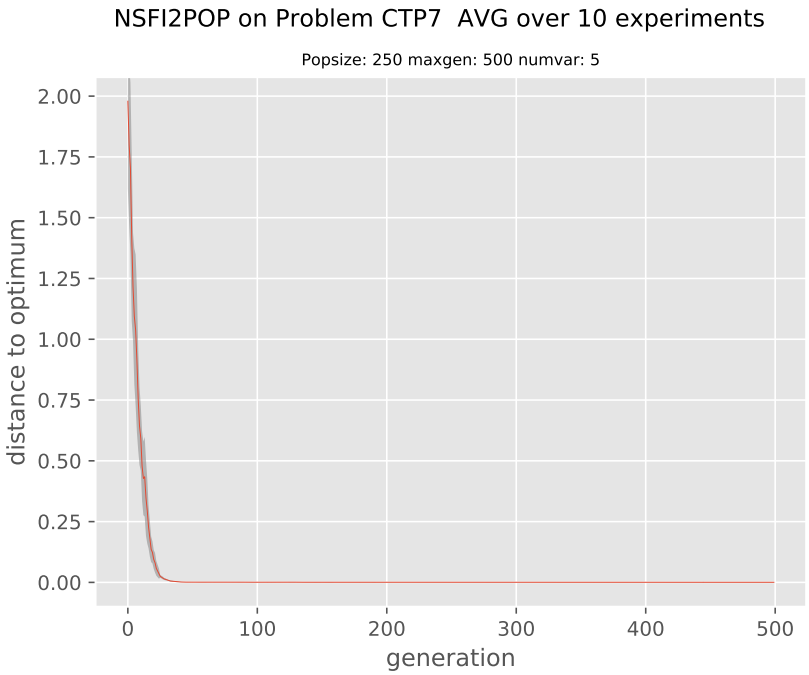



NSFI2POP on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation


NSFI2POP on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

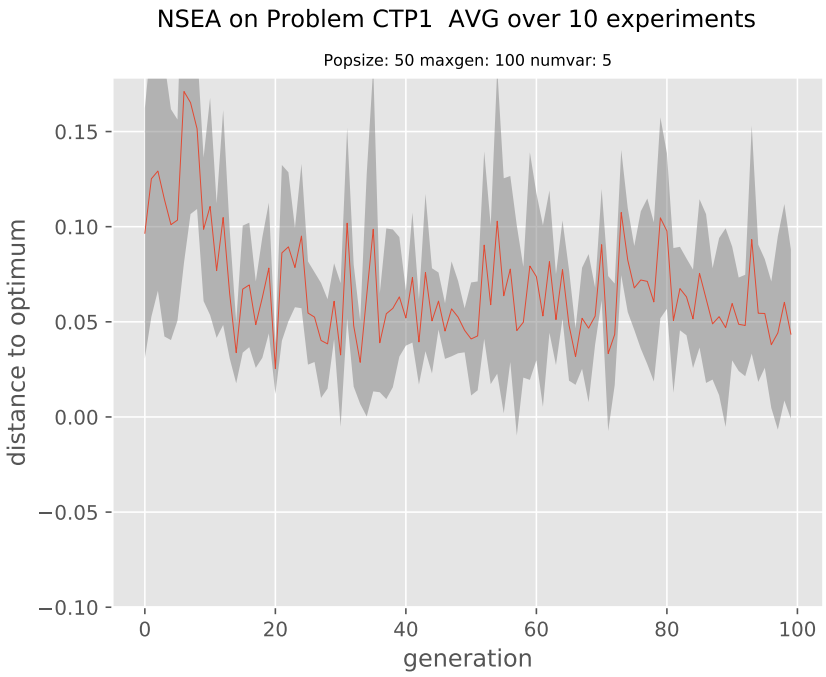

NSFI2POP on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

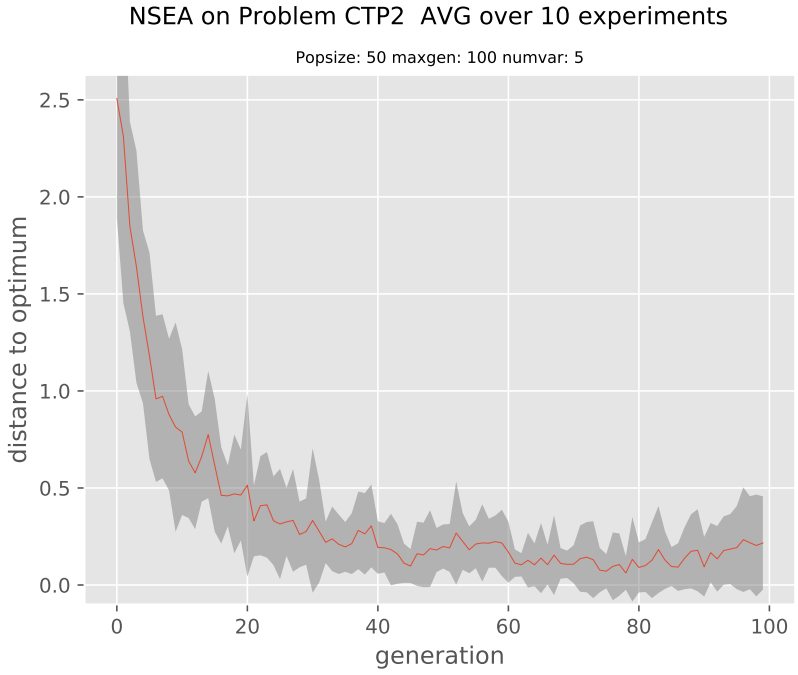

NSFI2POP on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

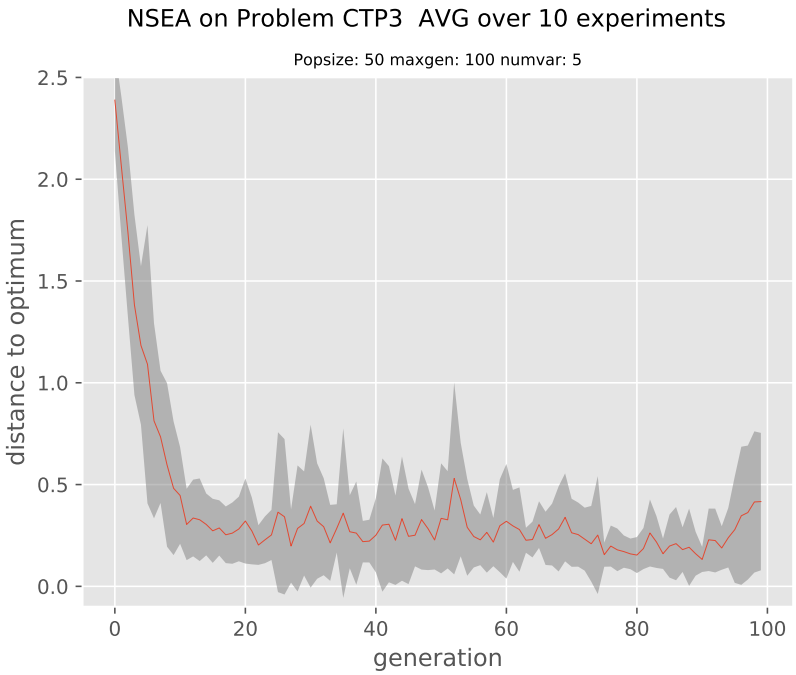

NSFI2POP on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

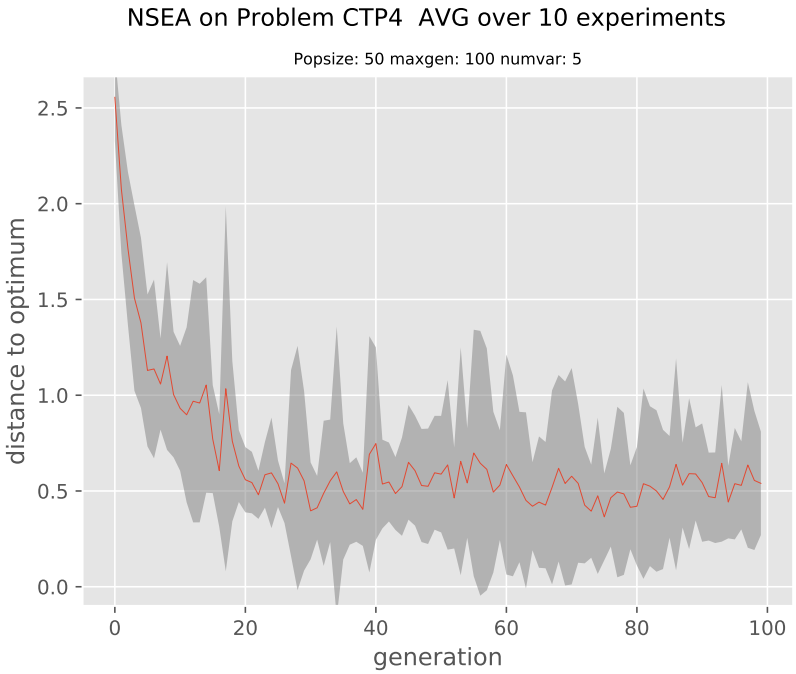
A.3 NSGA II Plots

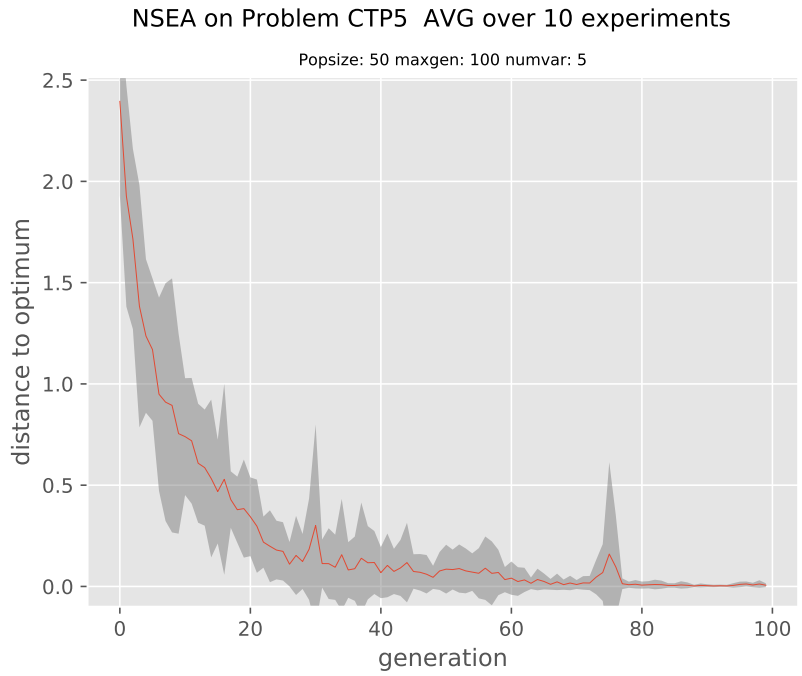
A.3.1 variables 5, generations 100, population 50

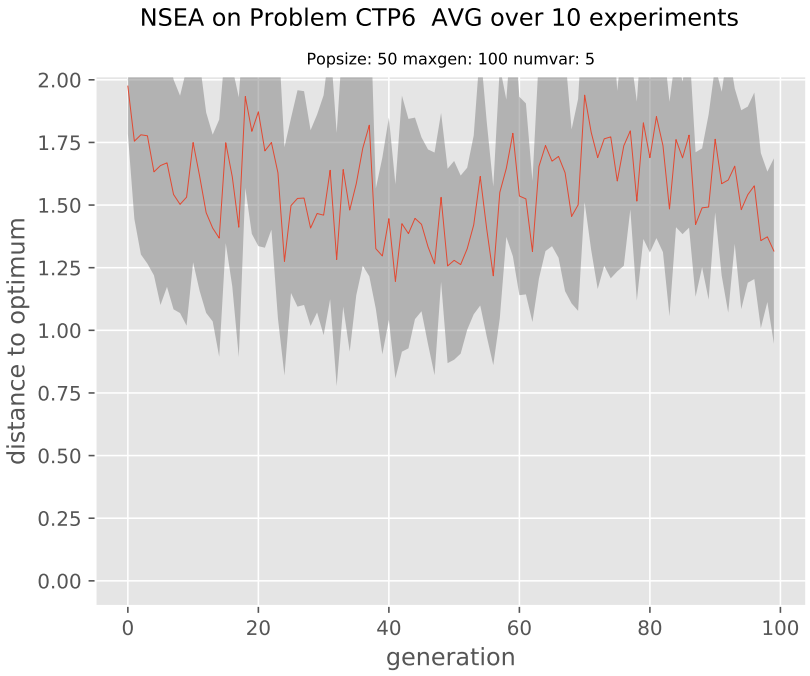


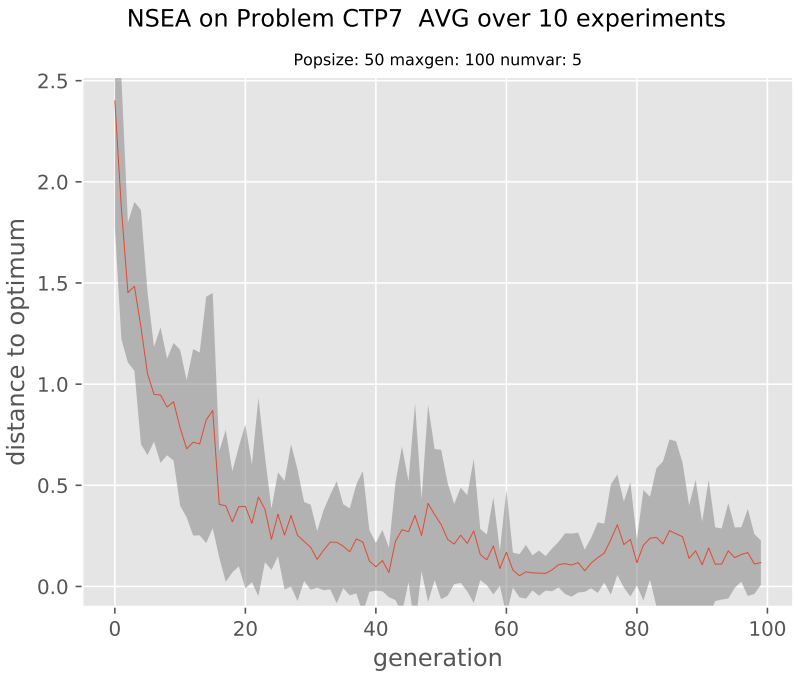



NSEA on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

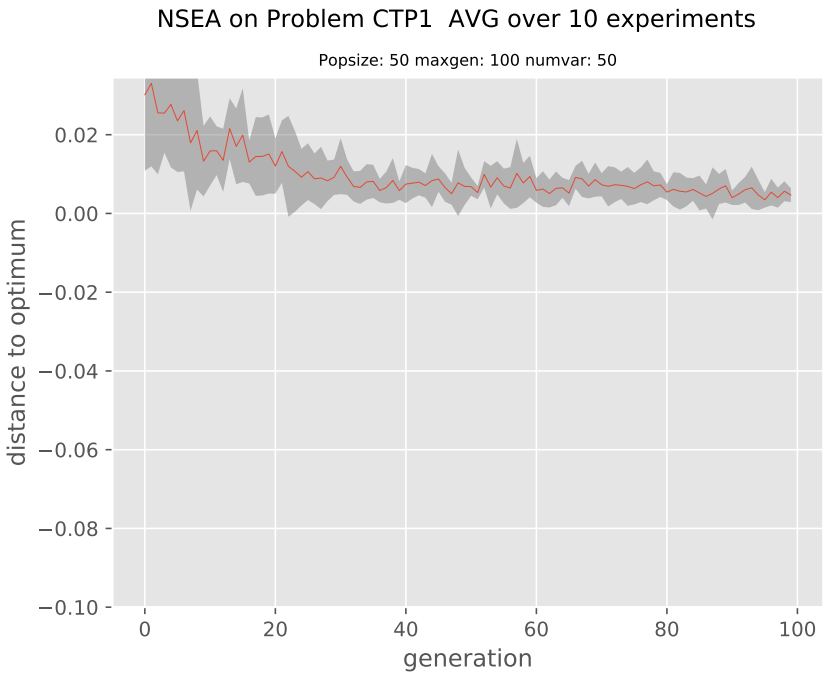

NSEA on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

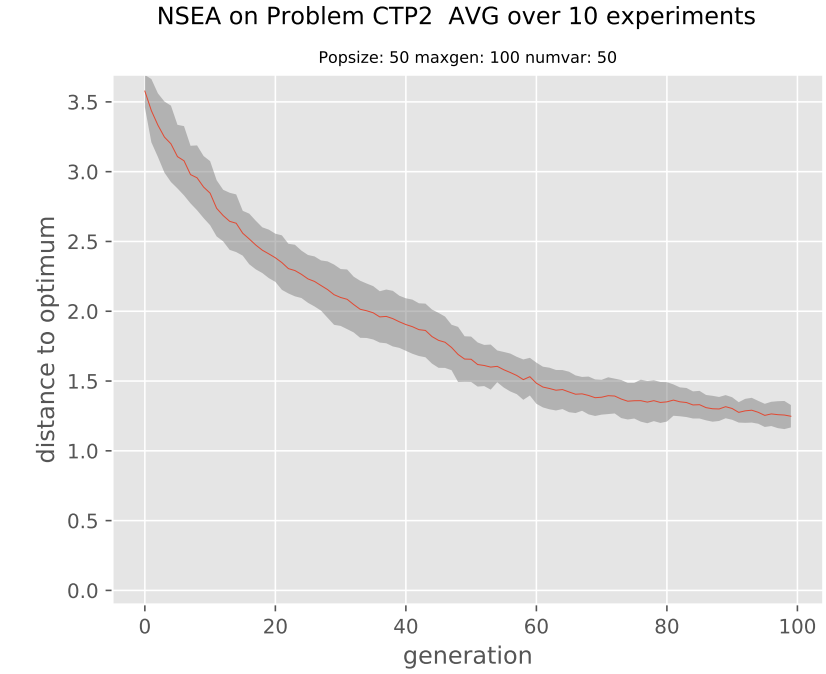

NSEA on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

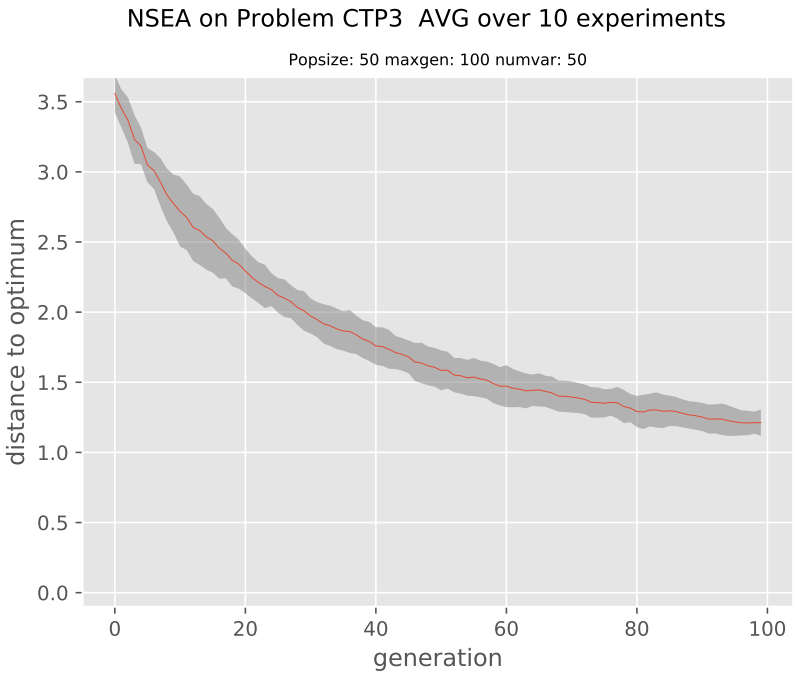

NSEA on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
generation

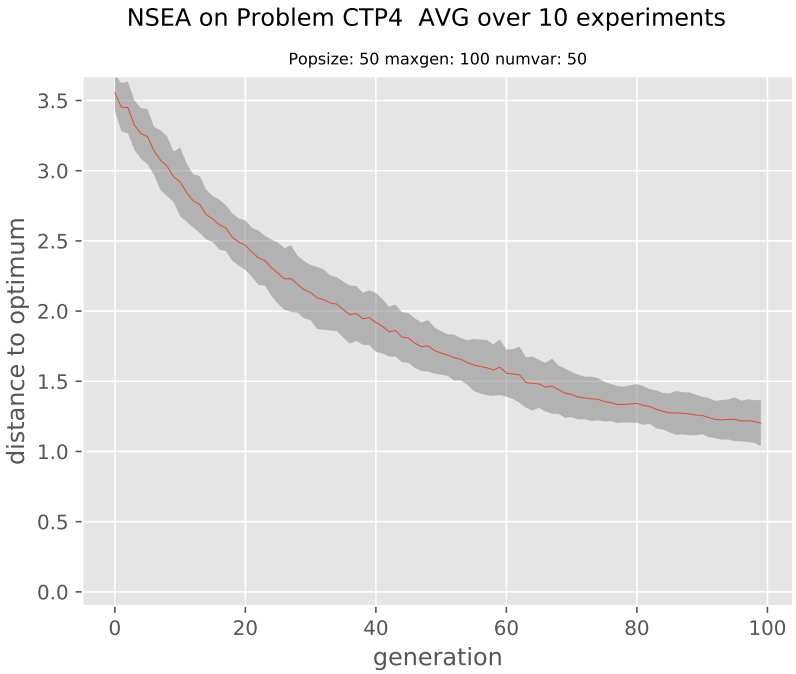

NSEA on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

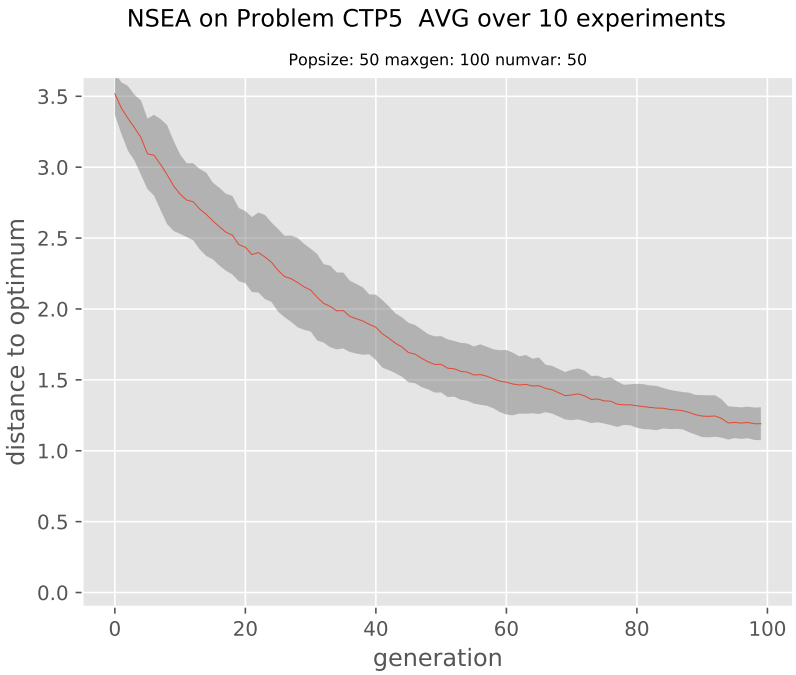
A.3.2 variables 50, generations 100, population 50

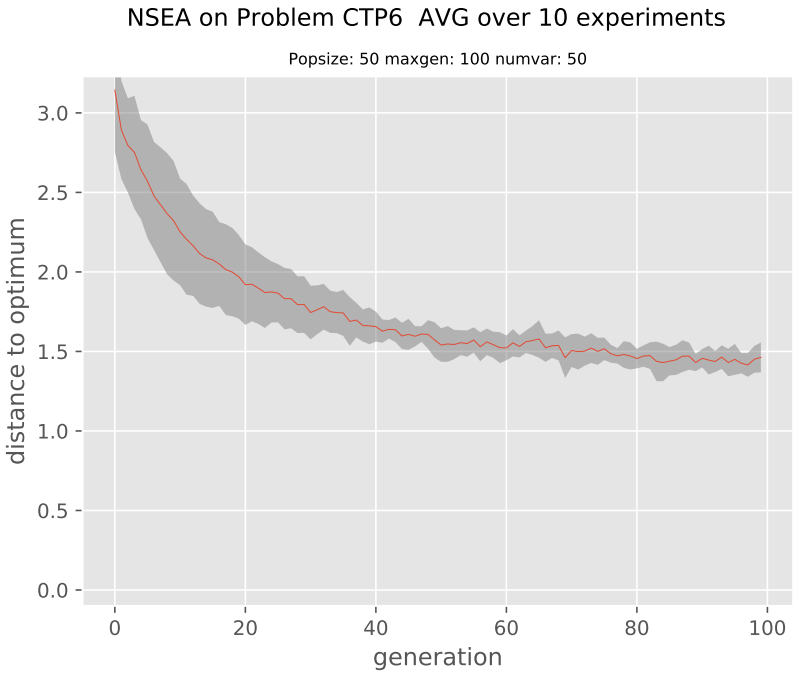


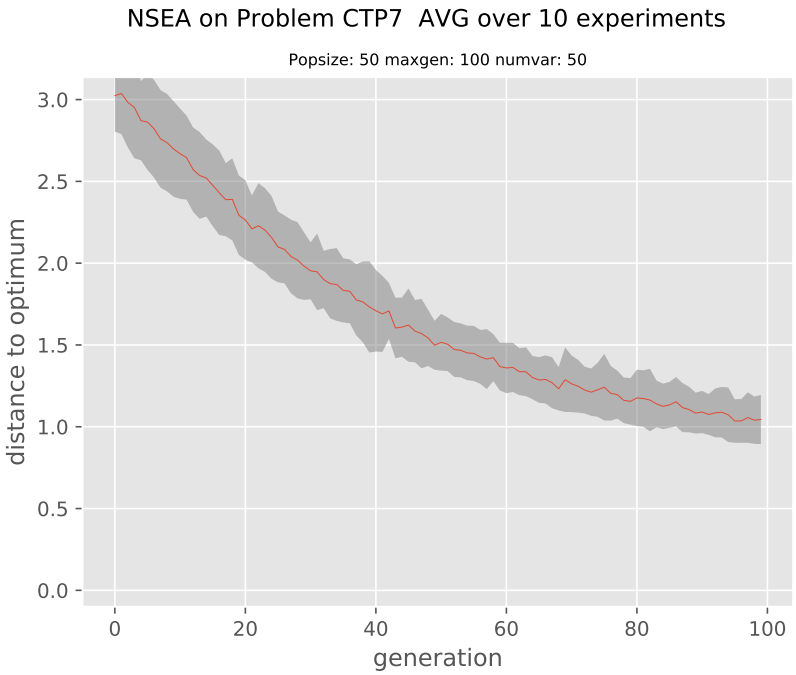



NSEA on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

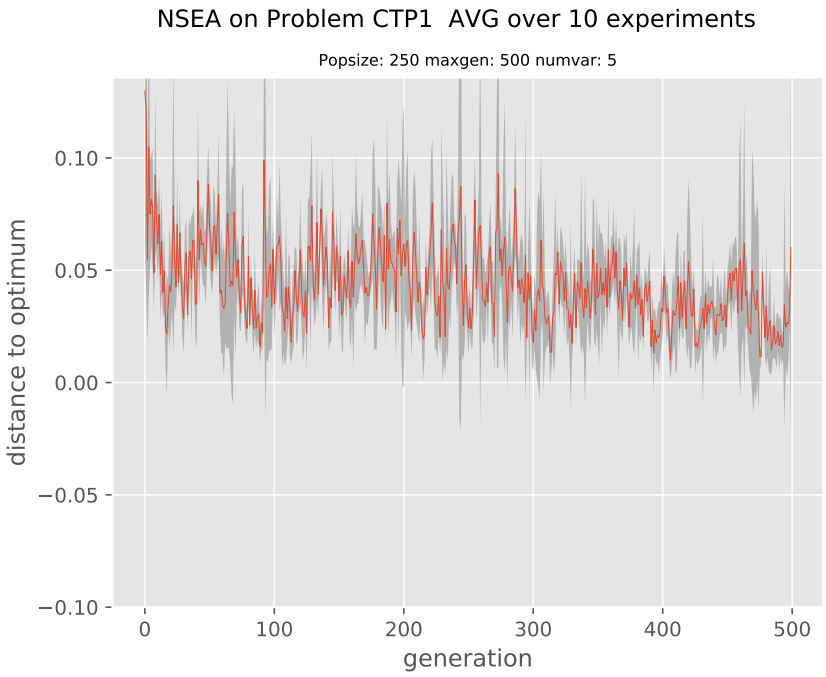

NSEA on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

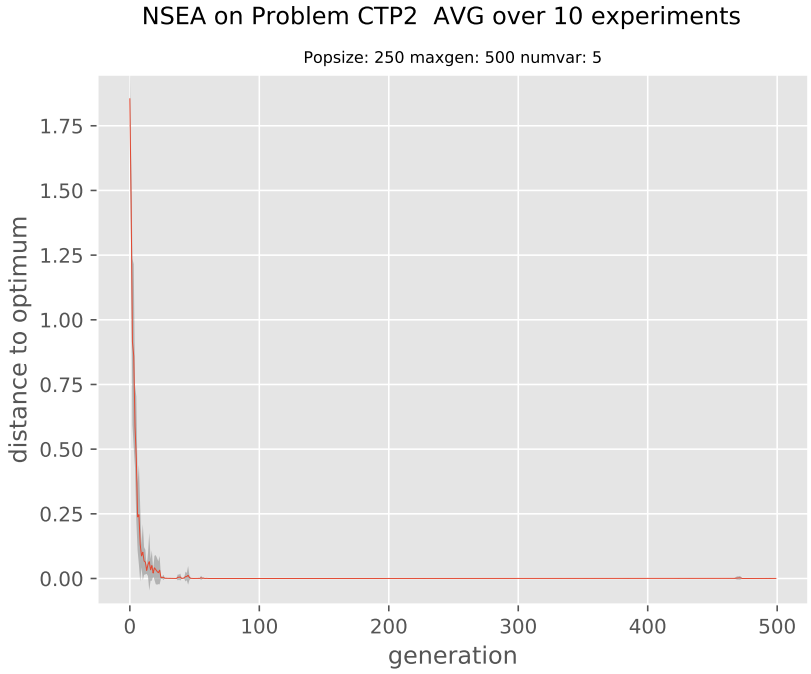

NSEA on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

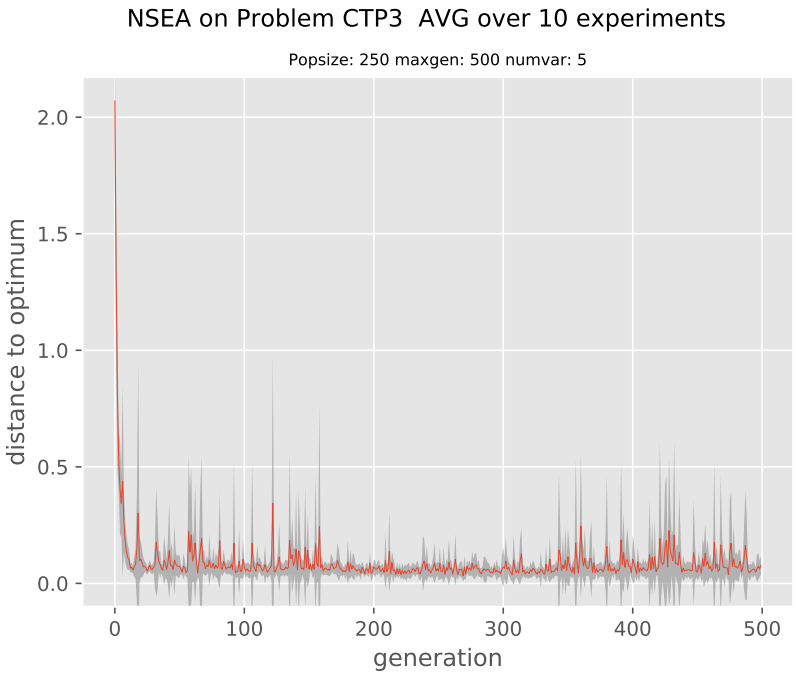

NSEA on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

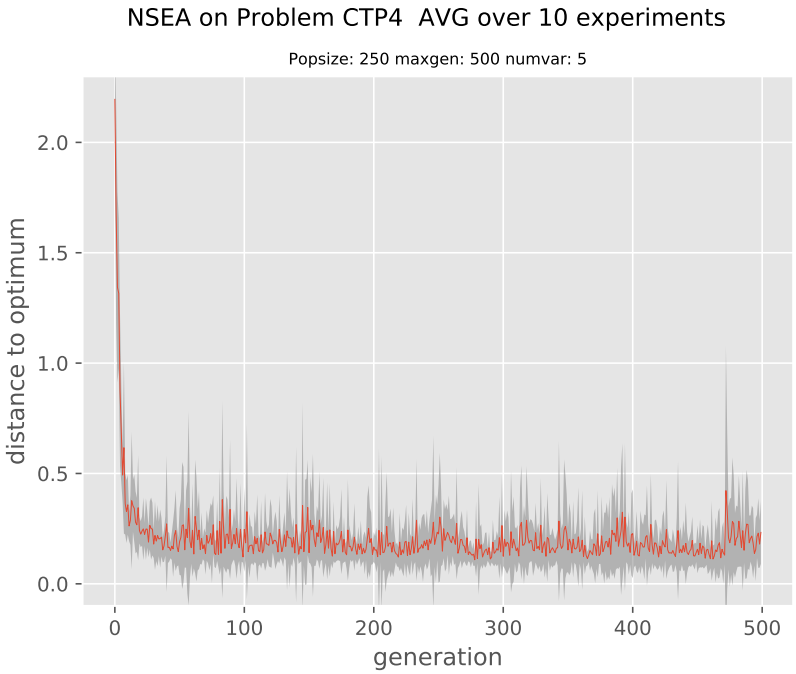

NSEA on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

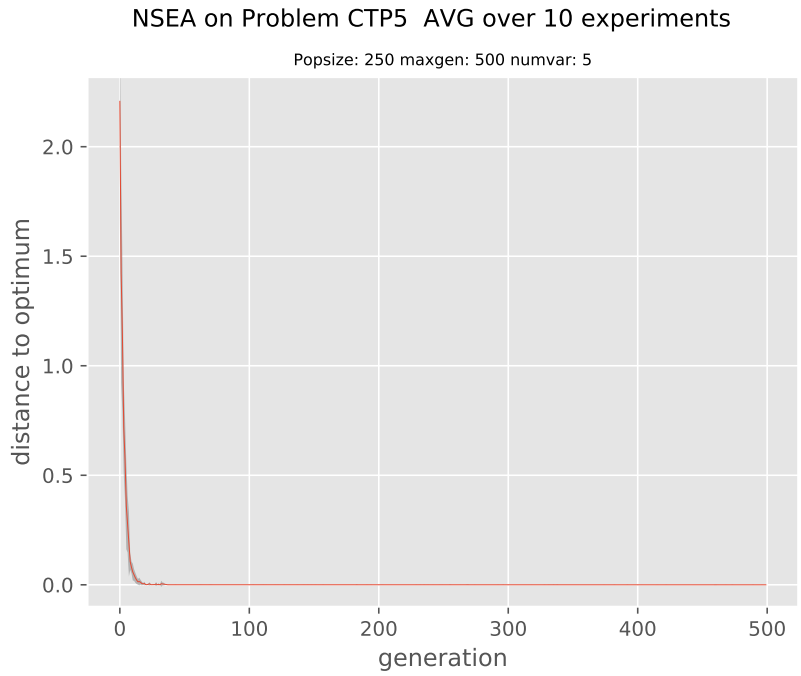
A.3.3 variables 5, generations 500, population 5

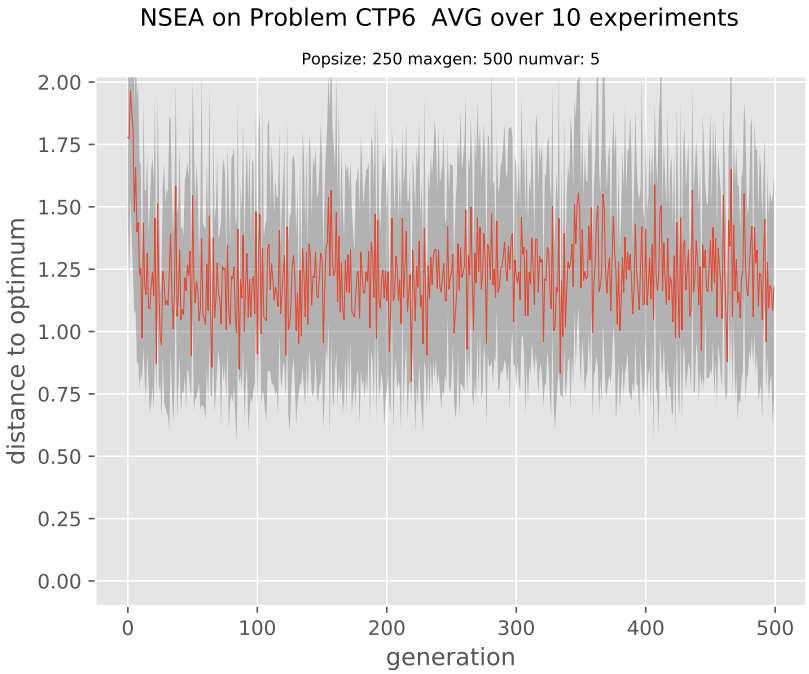


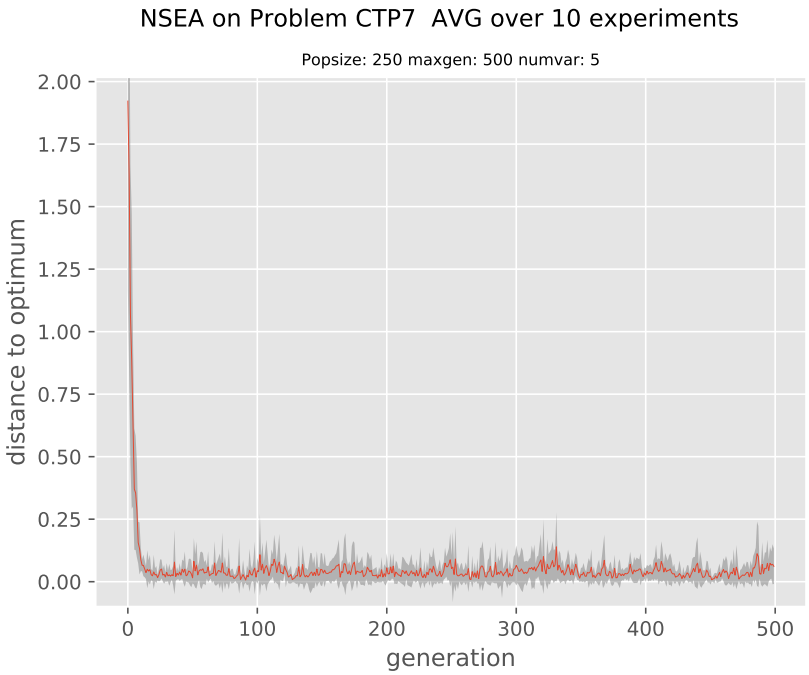



NSEA on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation


NSEA on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

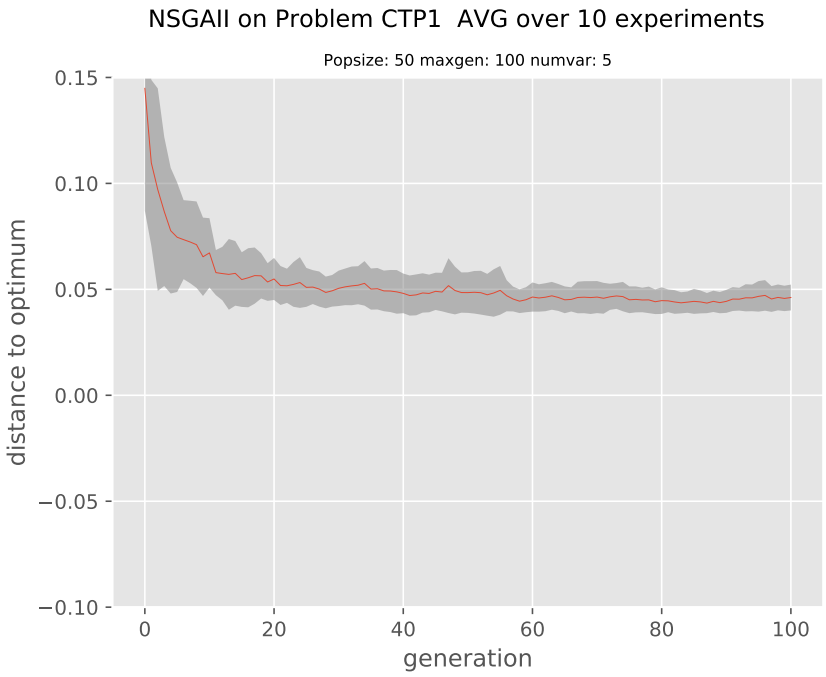

NSEA on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

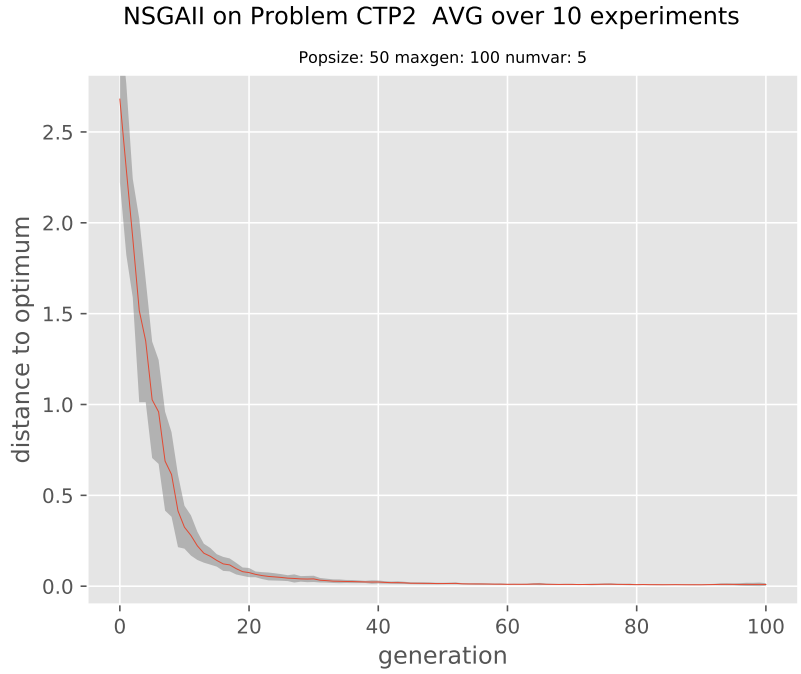

NSEA on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

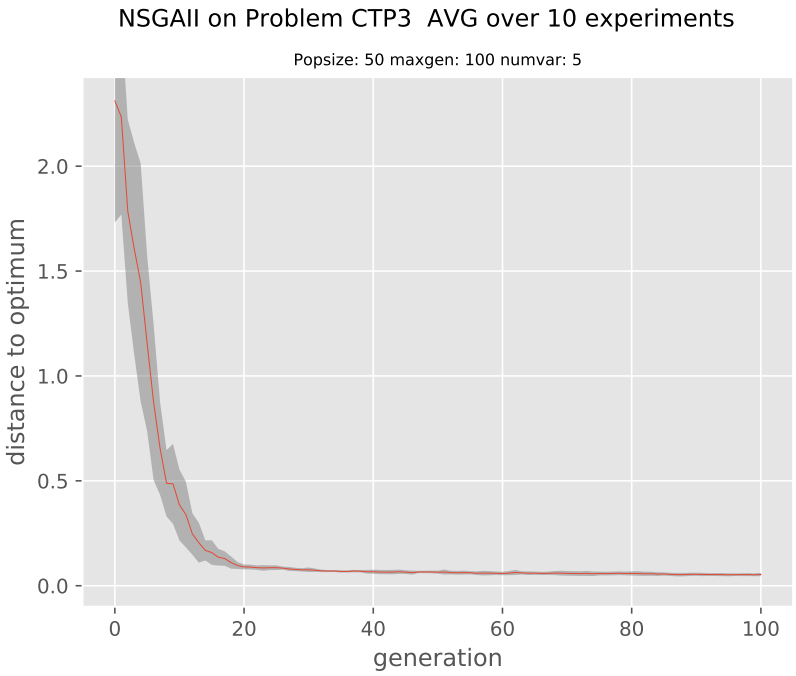

NSEA on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

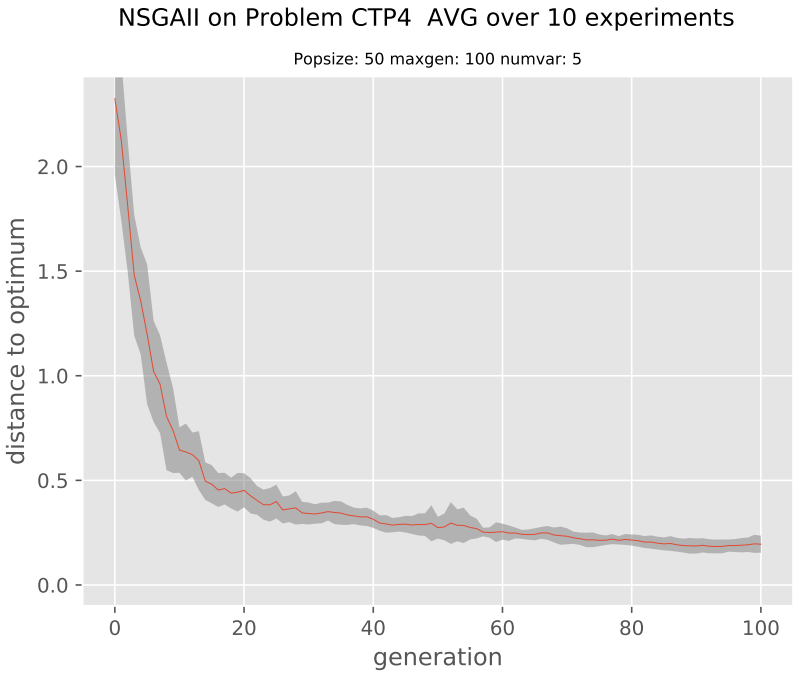
A.4 NSGA II Plots

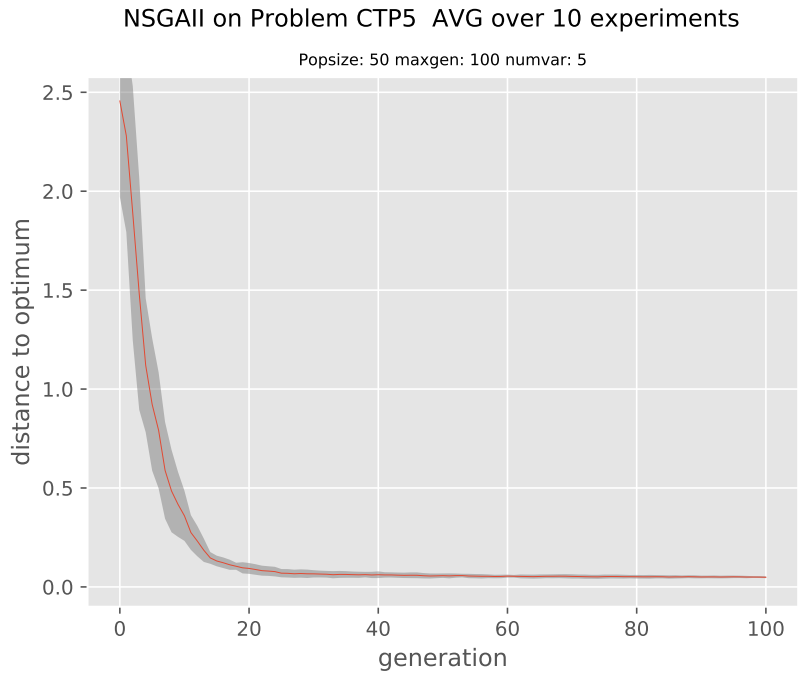
A.4.1 variables 5, generations 100, population 50

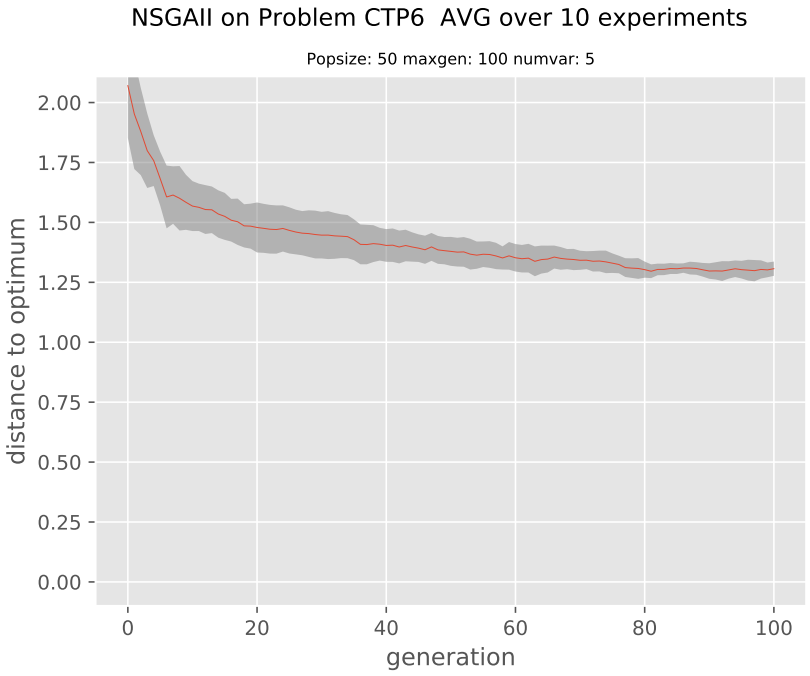


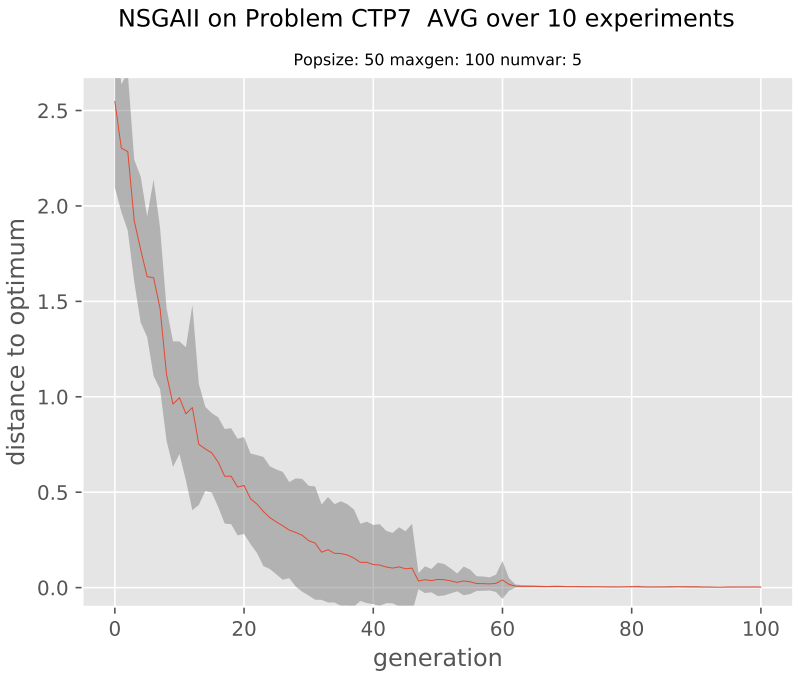



NSGAII on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
generation

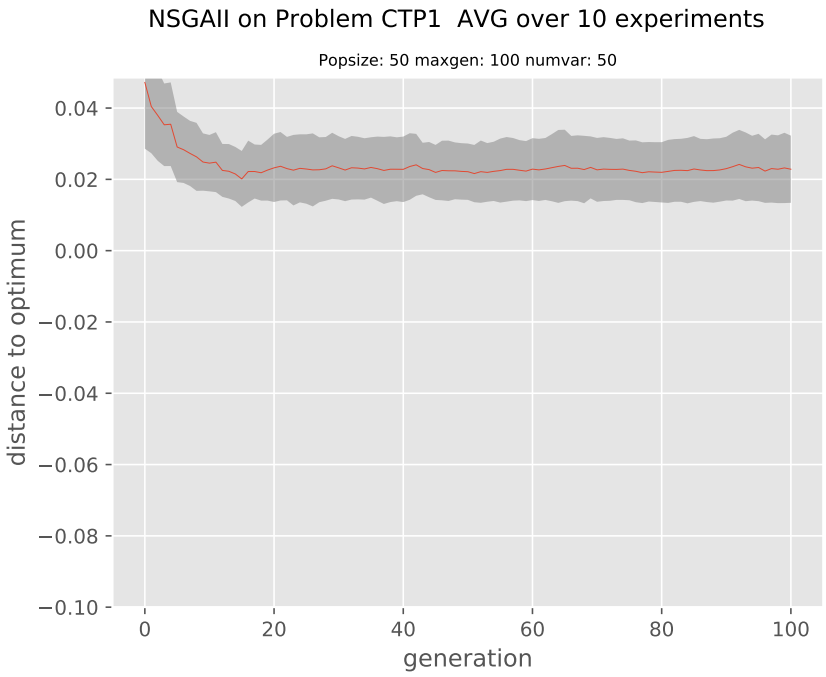

NSGAII on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
generation

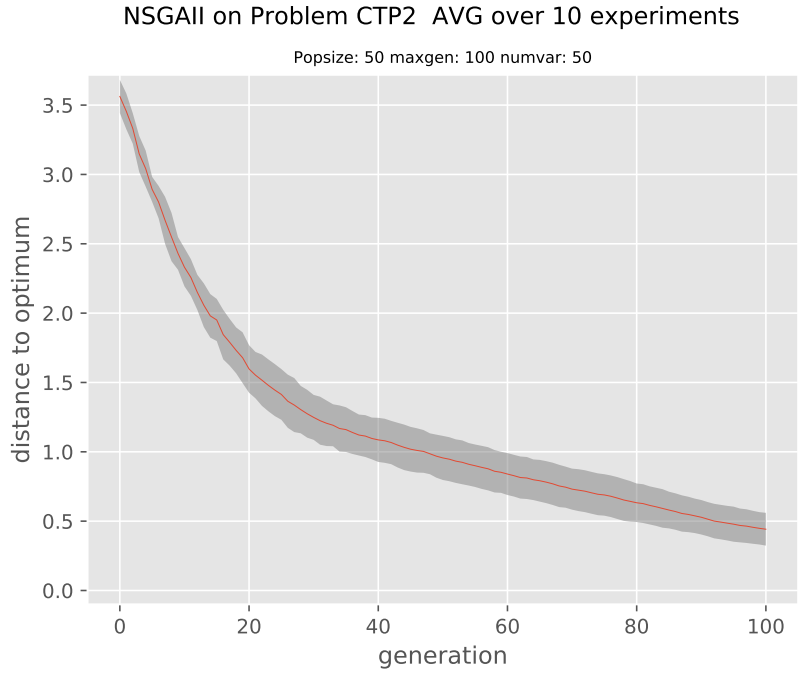

NSGAII on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

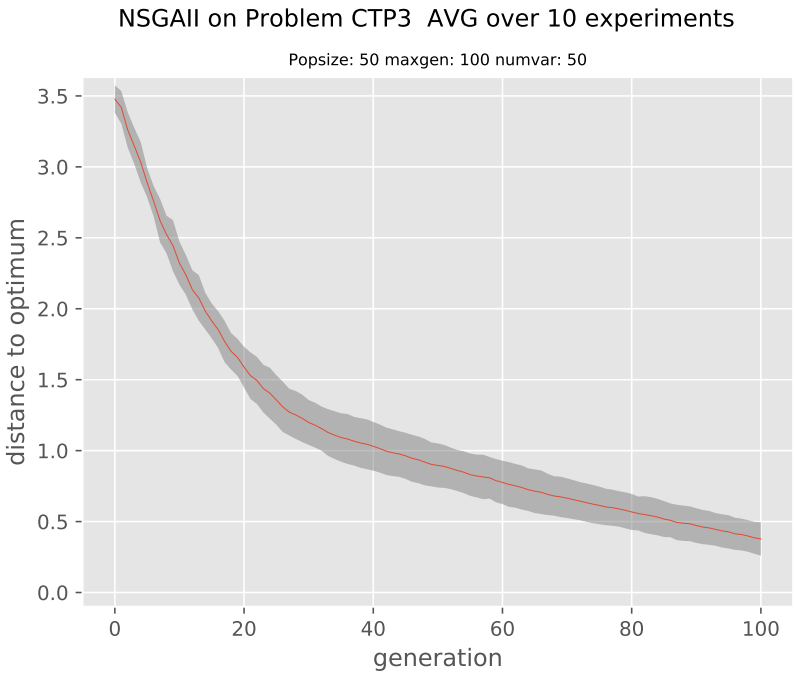

NSGAII on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
generation

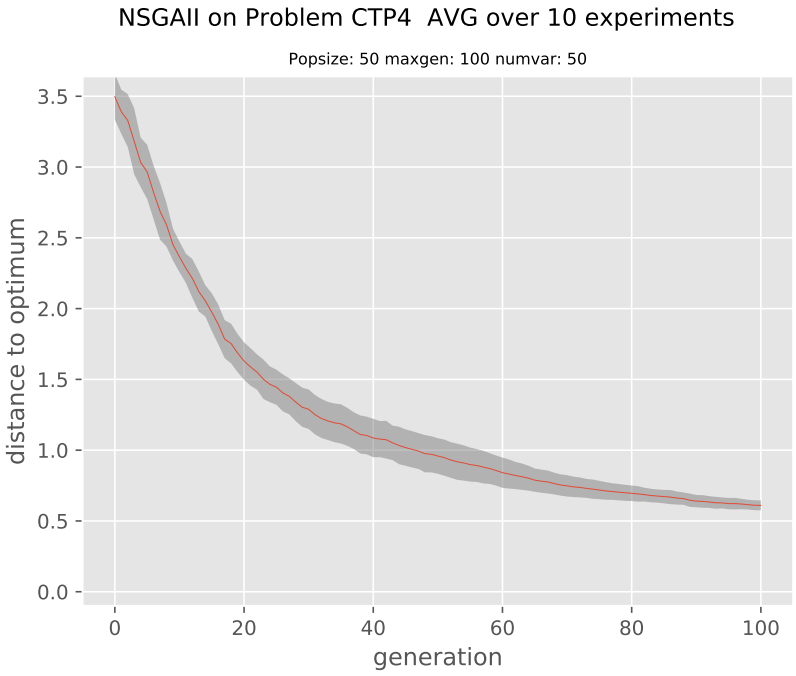

NSGAII on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

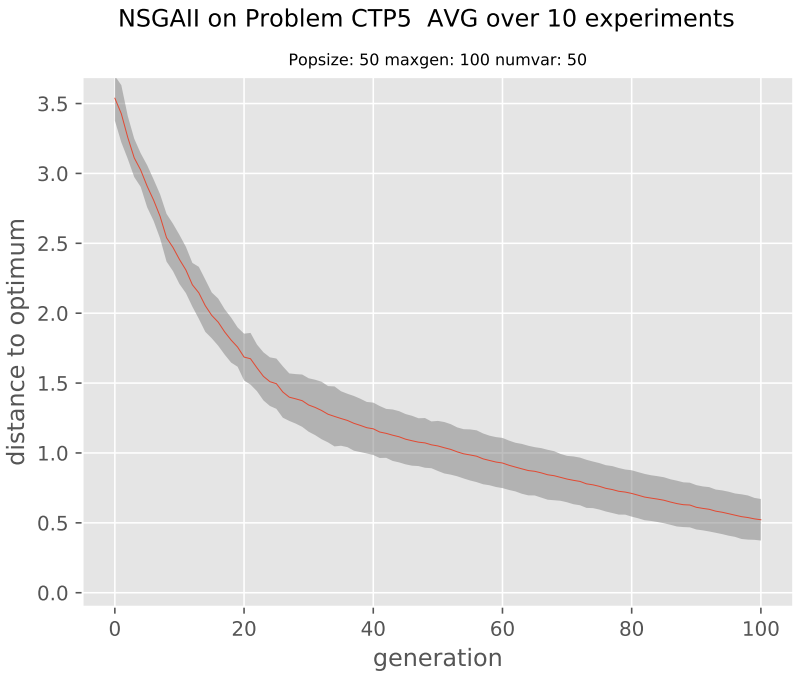
A.4.2 variables 50, generations 100, population 50

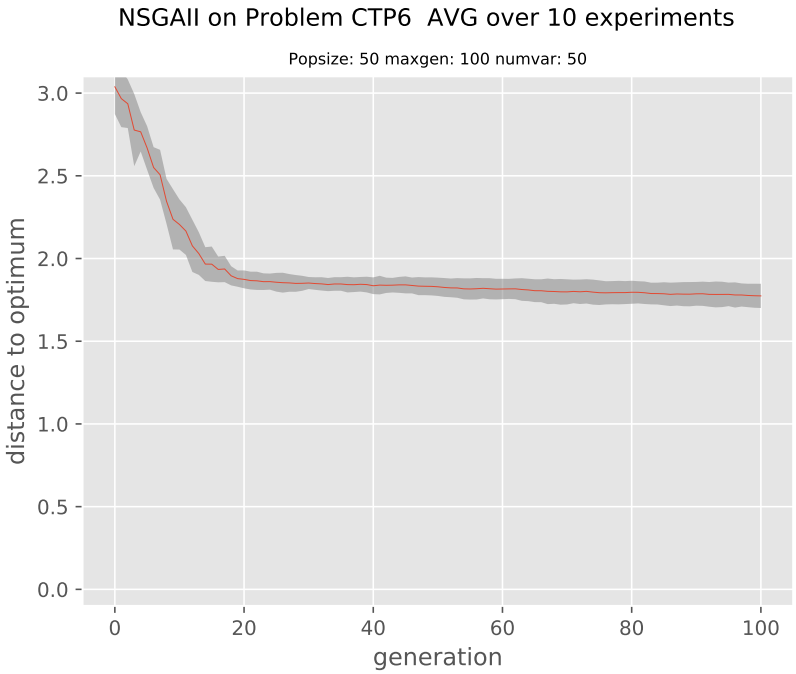


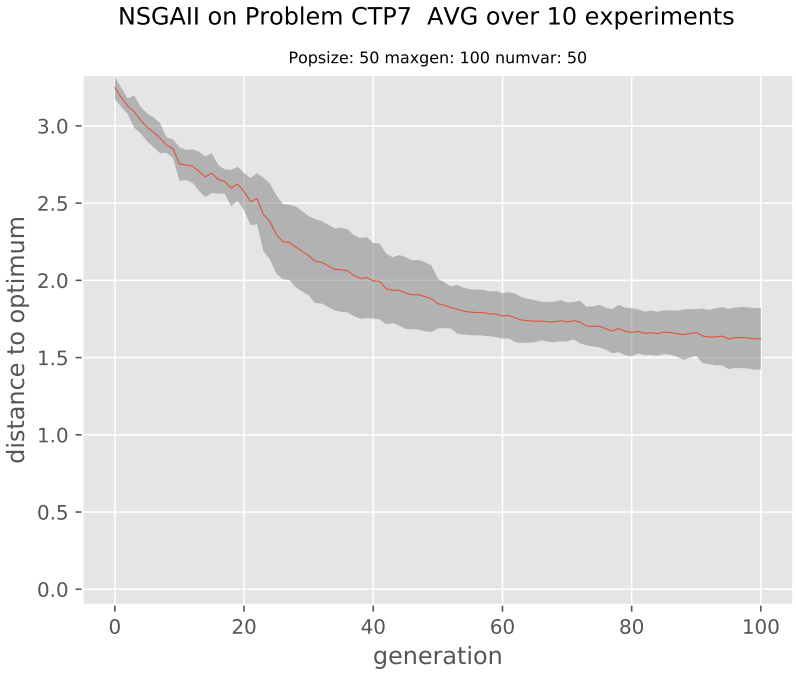



NSGAII on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

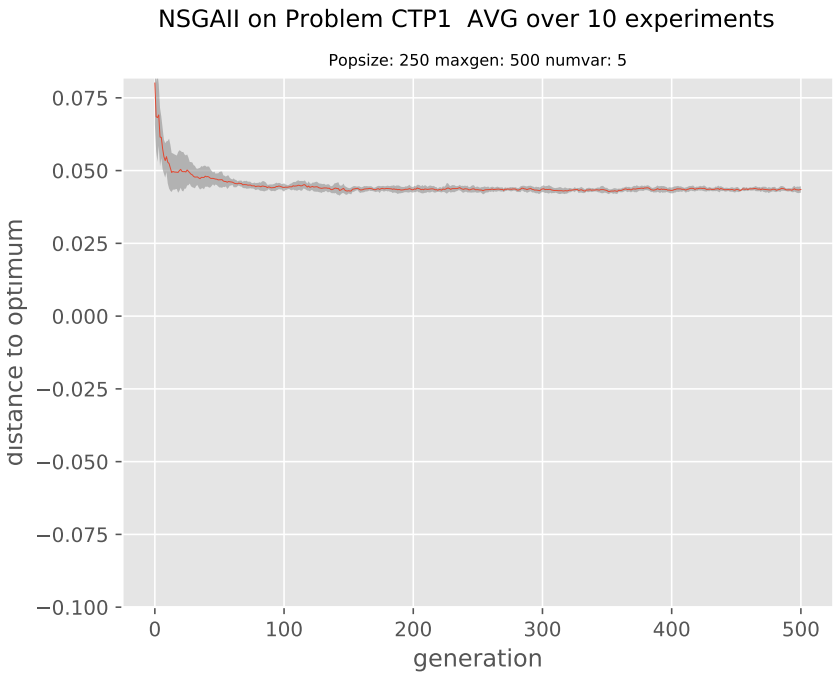

NSGAII on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

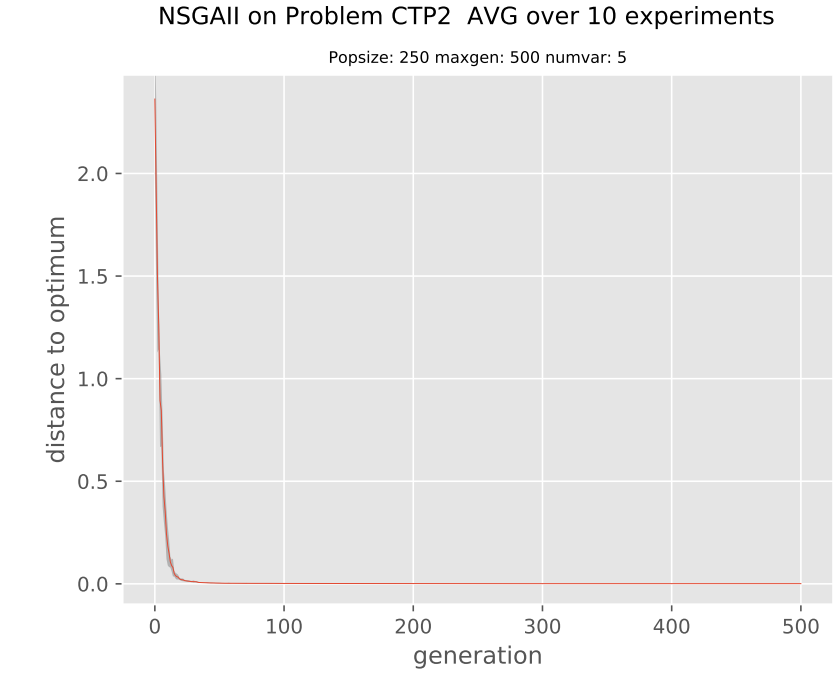

NSGAII on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

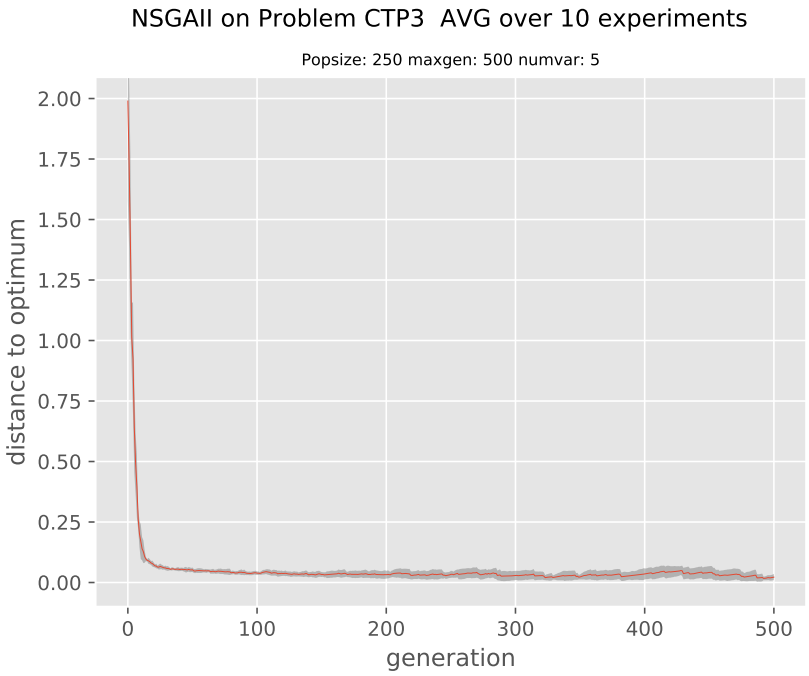

NSGAII on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

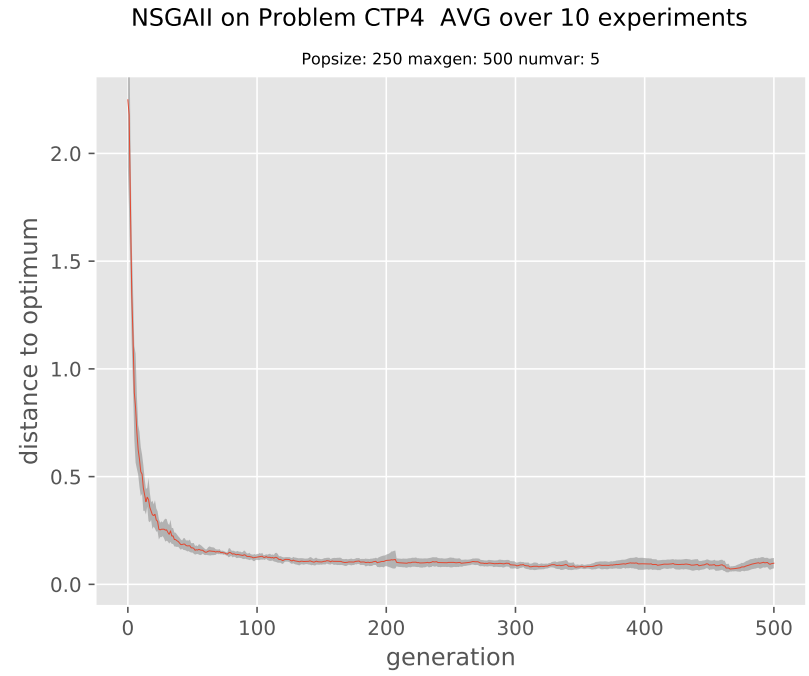

NSGAII on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

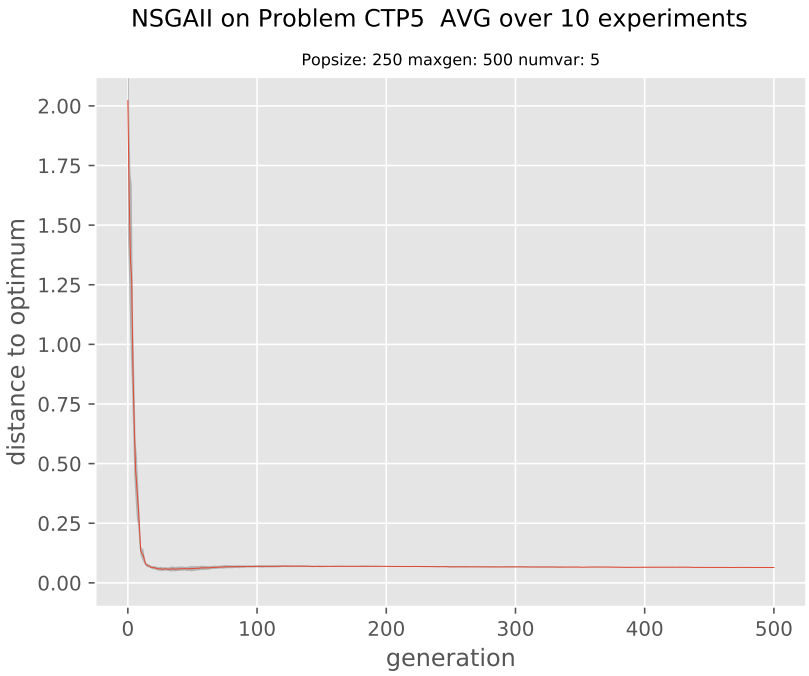
A.4.3 variables 5, generations 500, population 5

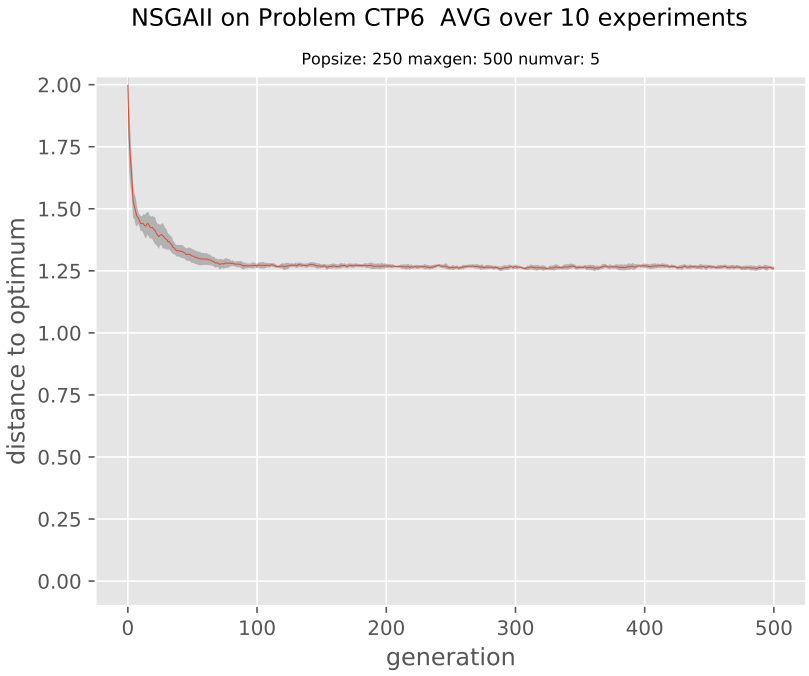


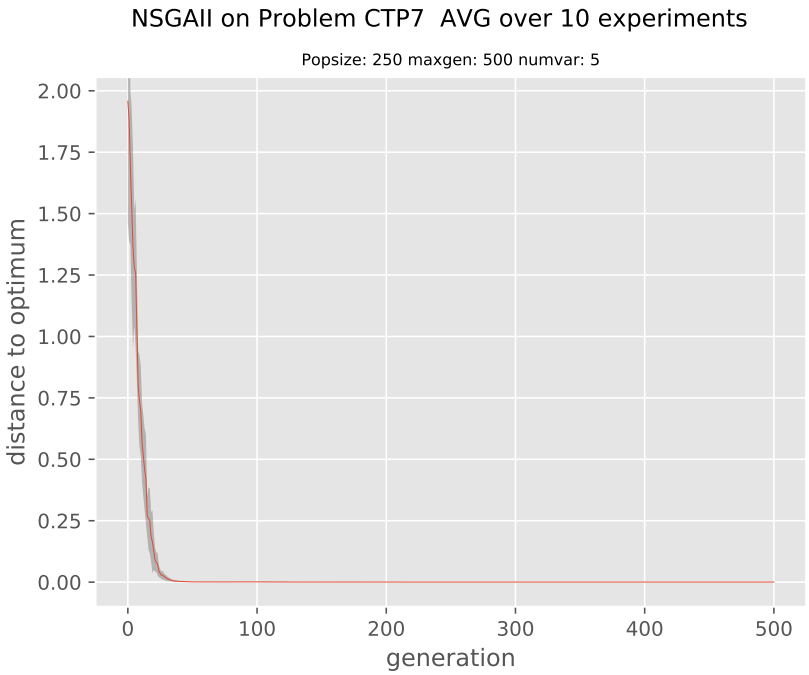



NSGAII on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation


NSGAII on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

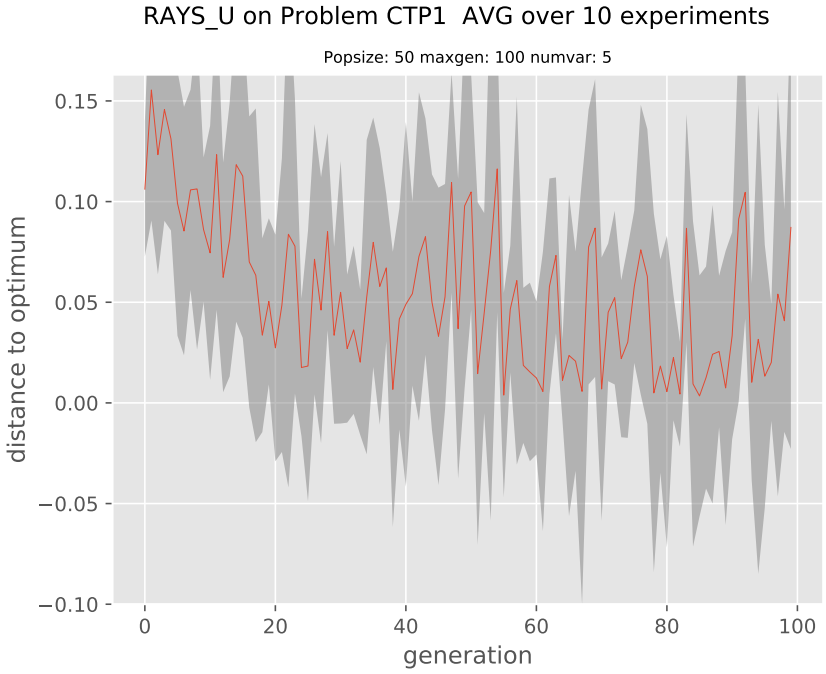

NSGAII on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

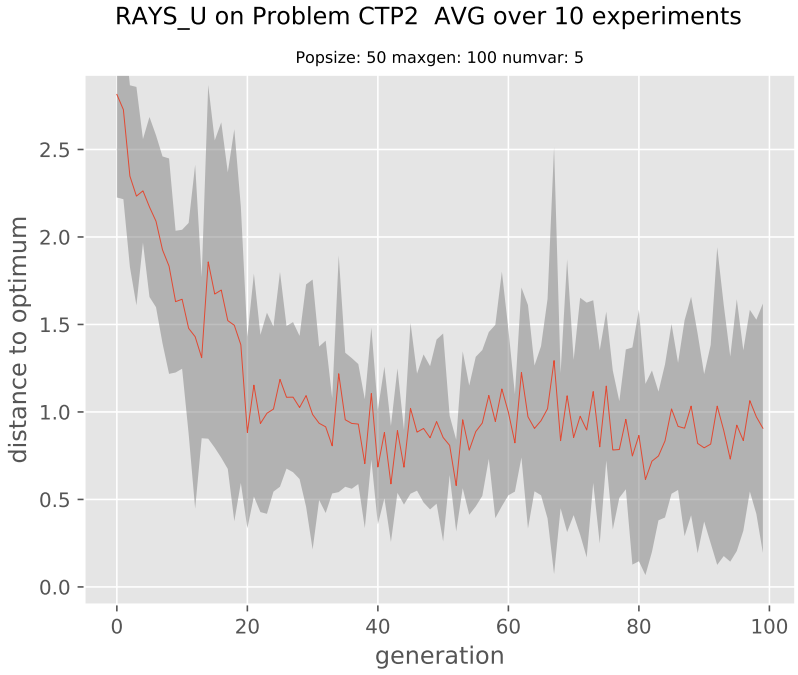

NSGAII on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

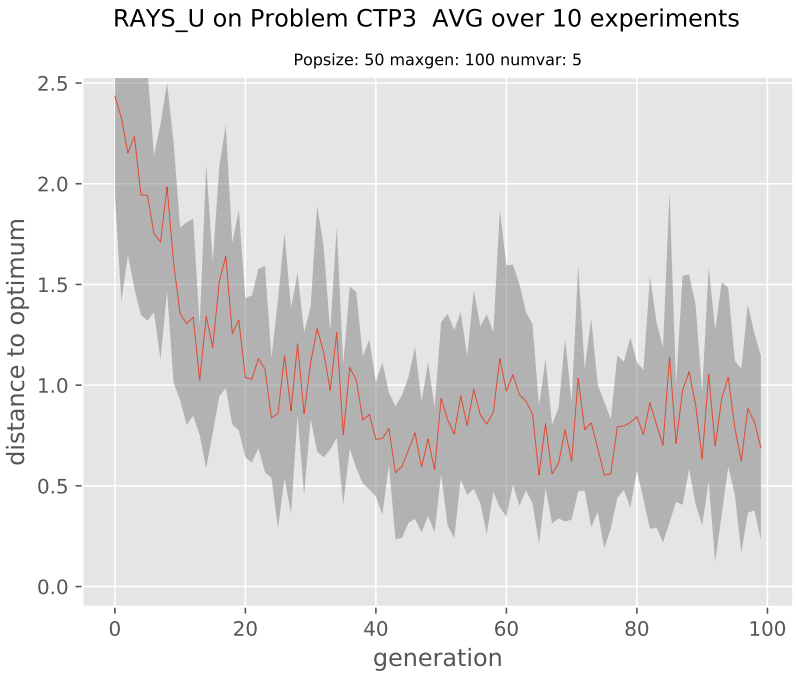

NSGAII on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

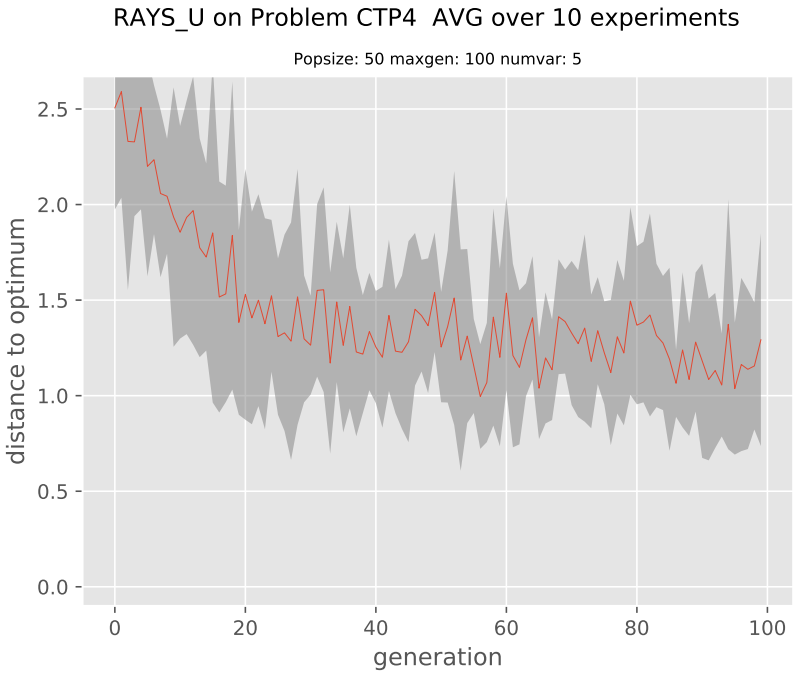
A.5 Ray's Plots

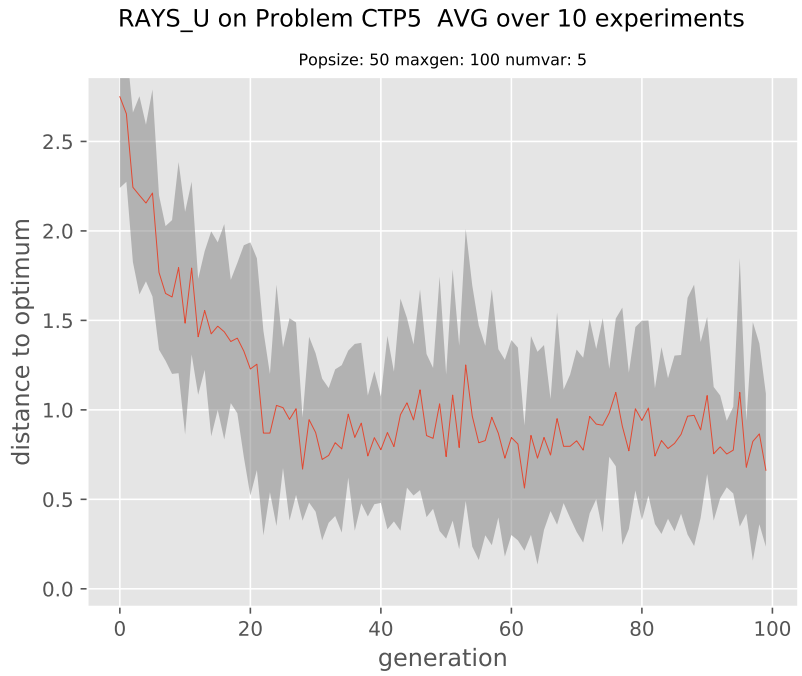
A.5.1 variables 5, generations 100, population 50

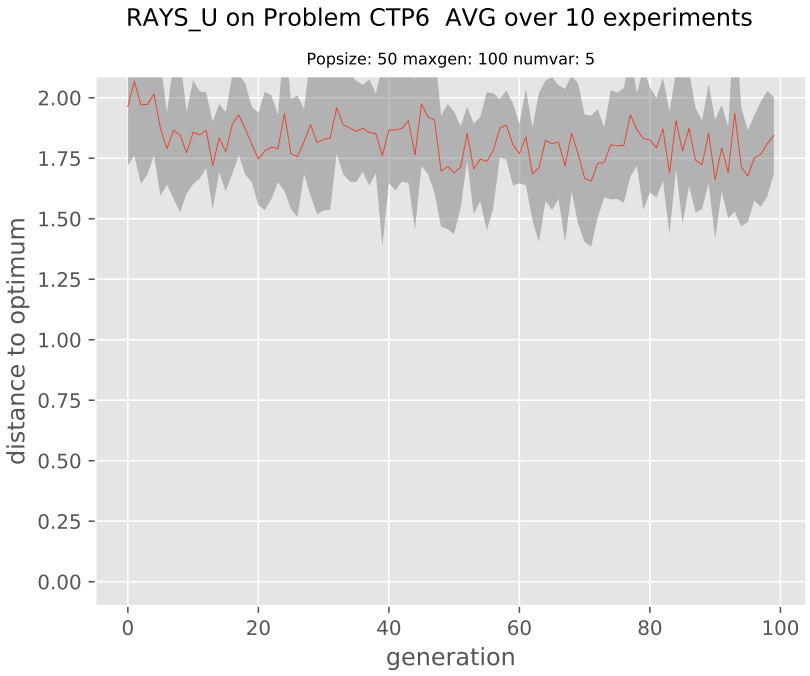


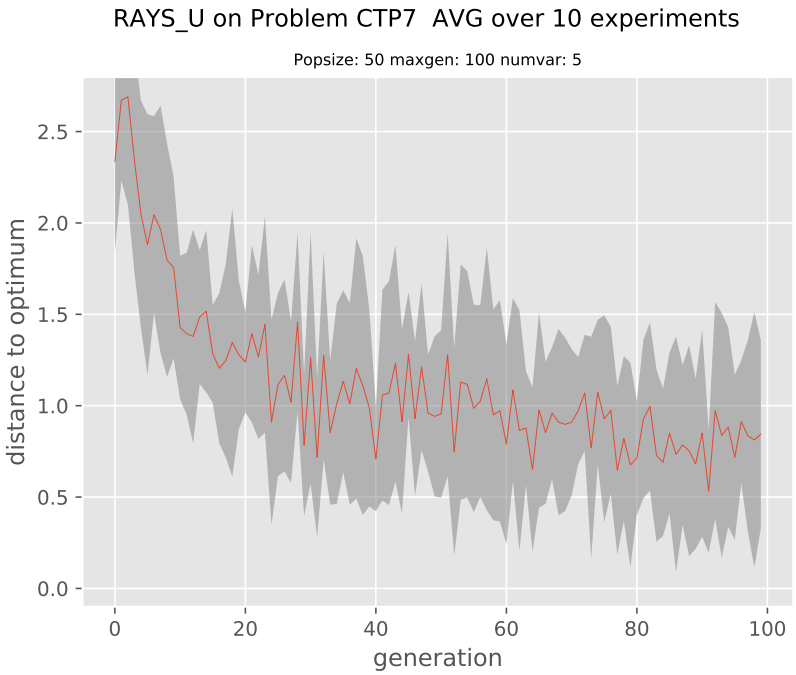


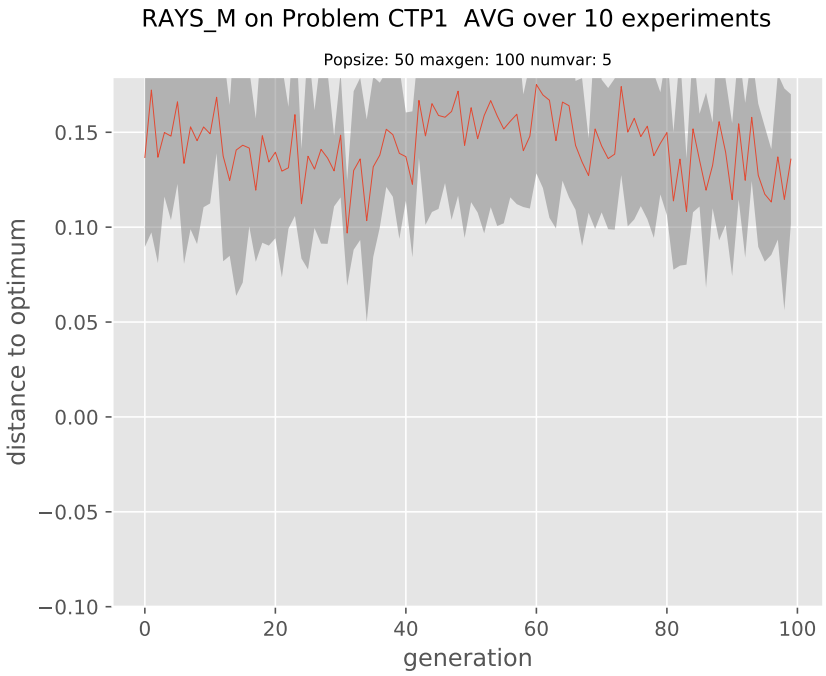

RAYS_U on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

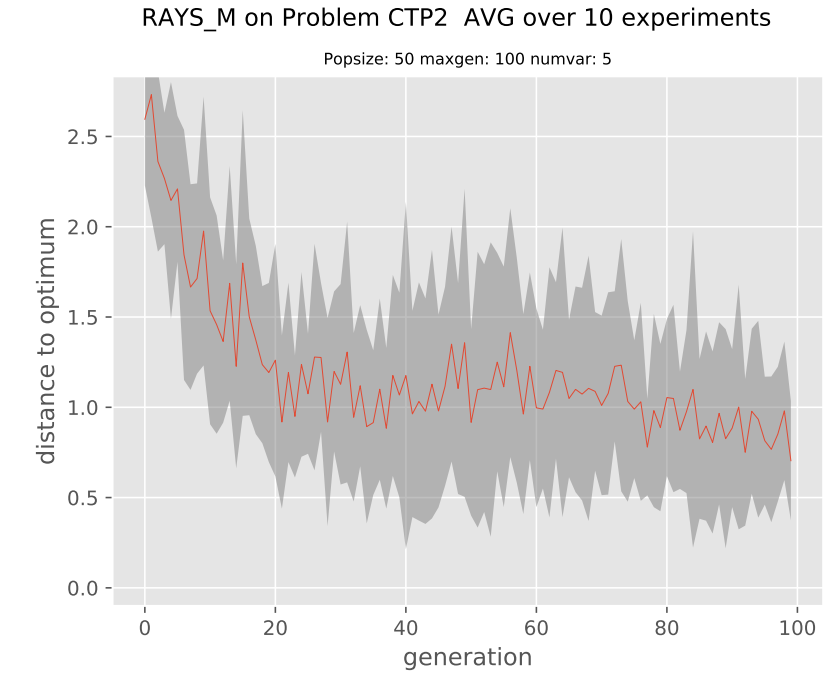

RAYS_U on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

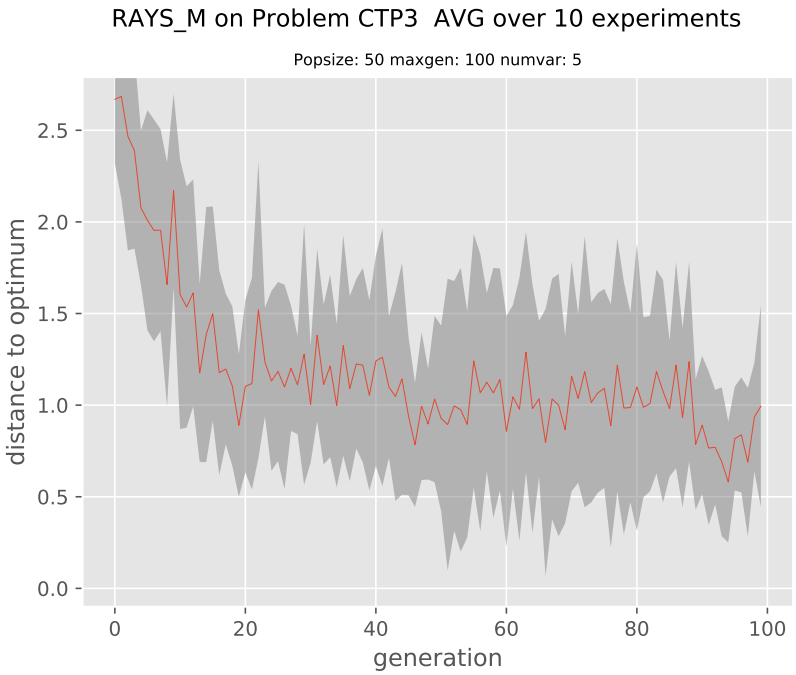

RAYS_U on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

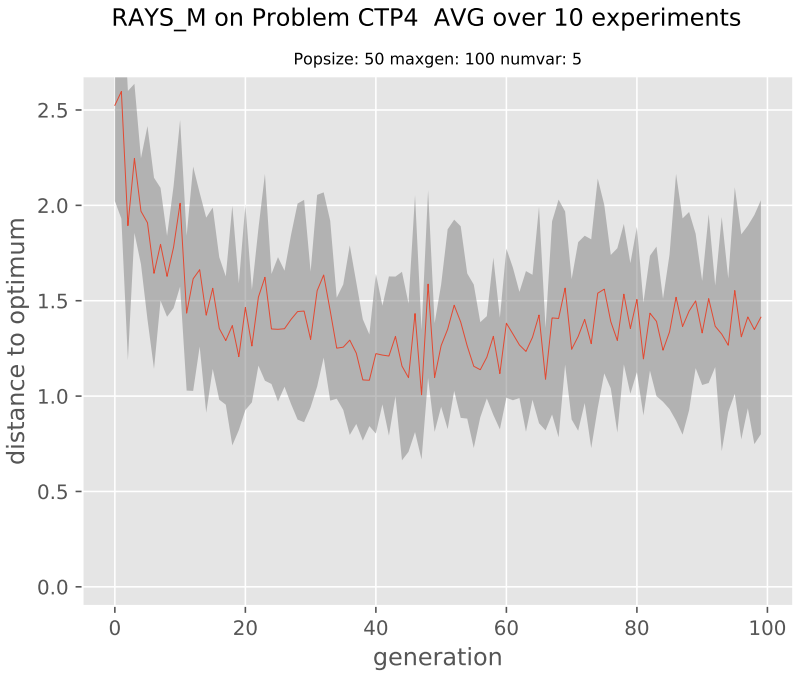

RAYS_U on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
generation

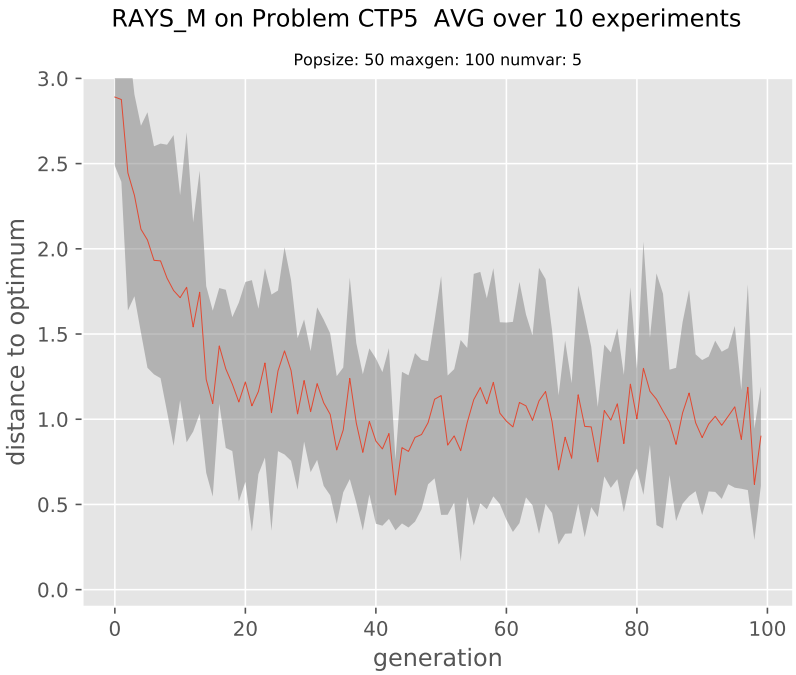

RAYS_U on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

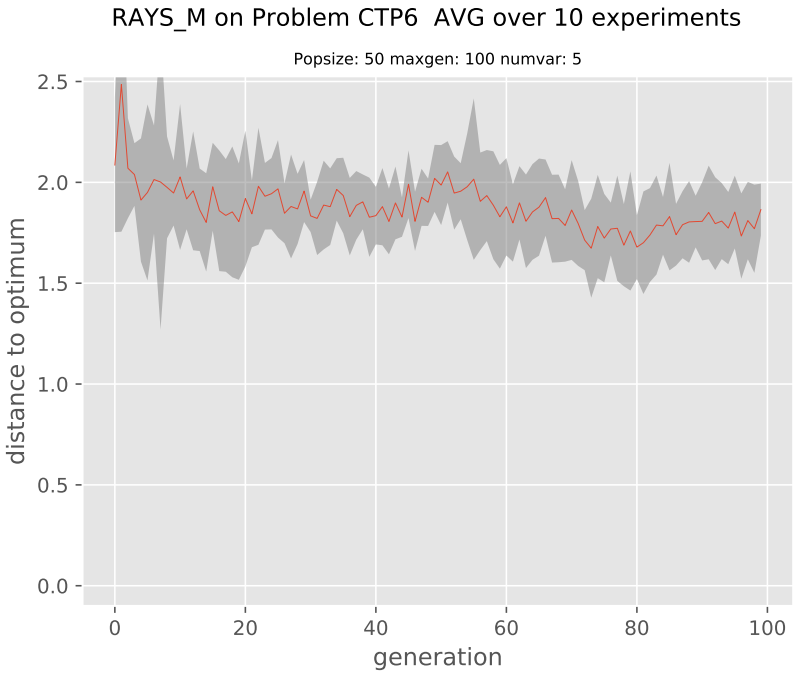

RAYS_M on Problem CTP1 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.15
0.10
0.05
0.00
−0.05
−0.10
0
20
40
60
80
100
generation

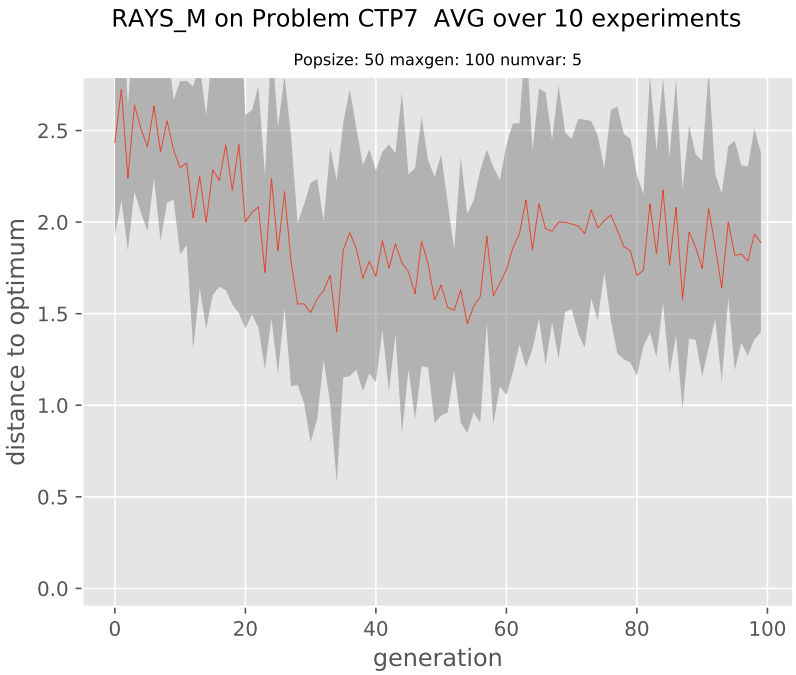

RAYS_M on Problem CTP2 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

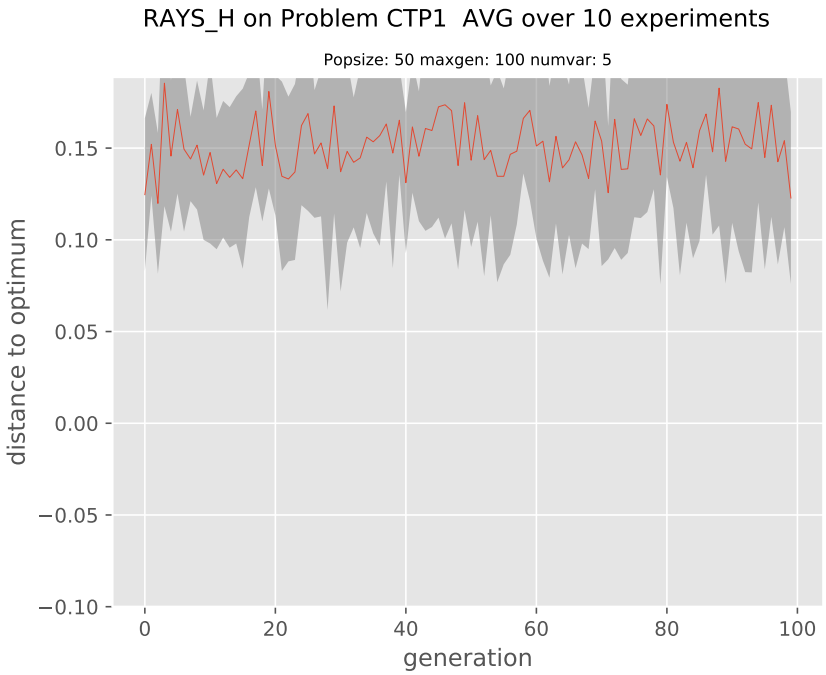

RAYS_M on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

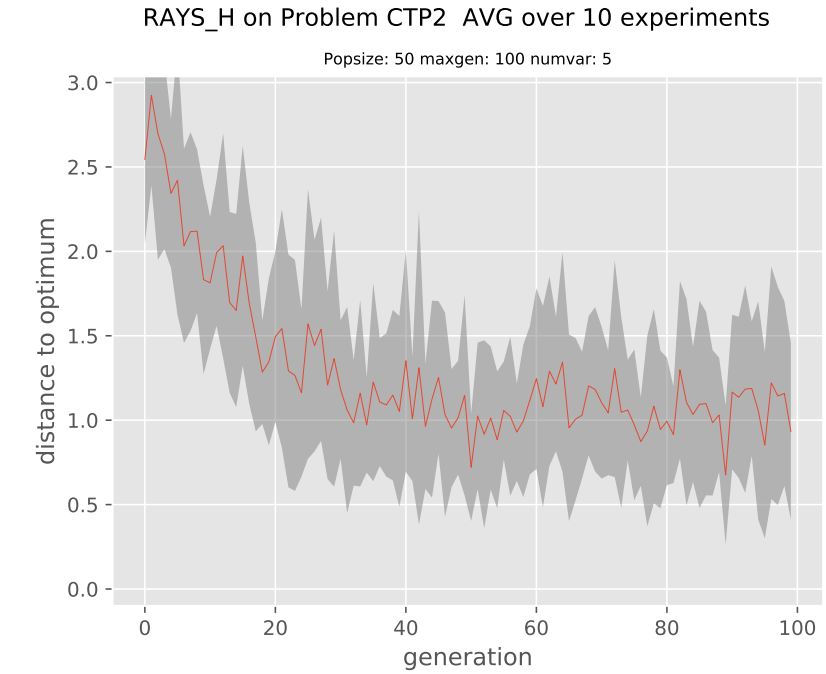

RAYS_M on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

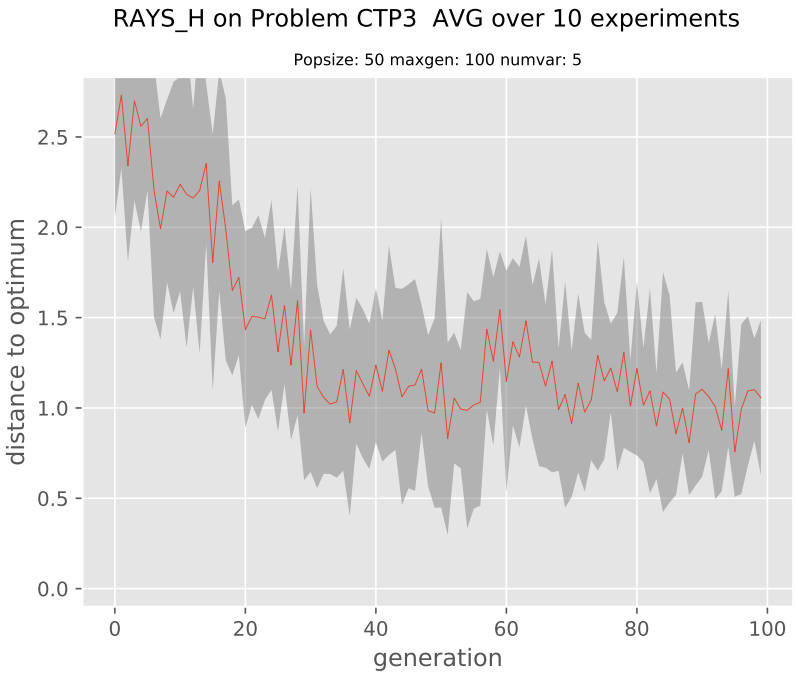

RAYS_M on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100

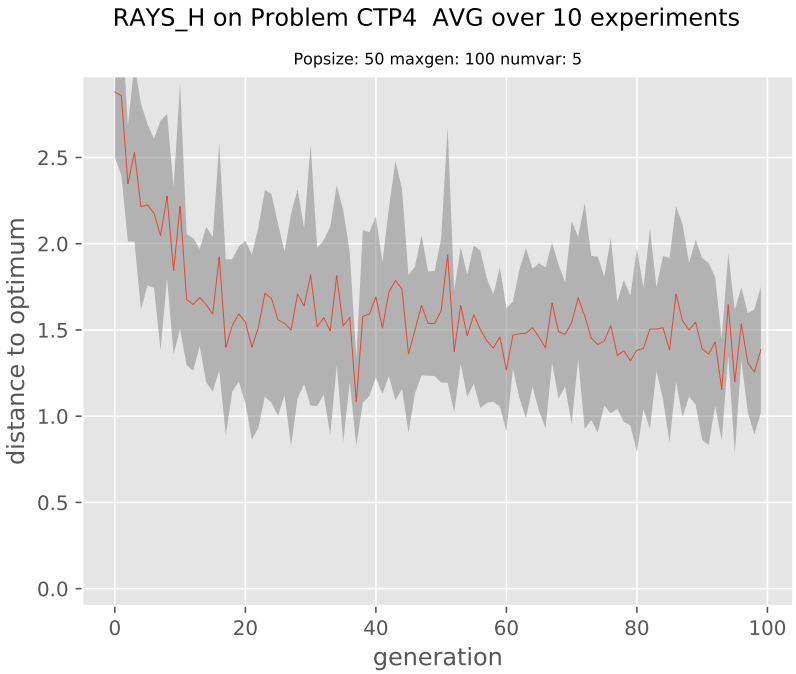

RAYS_M on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
generation
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100

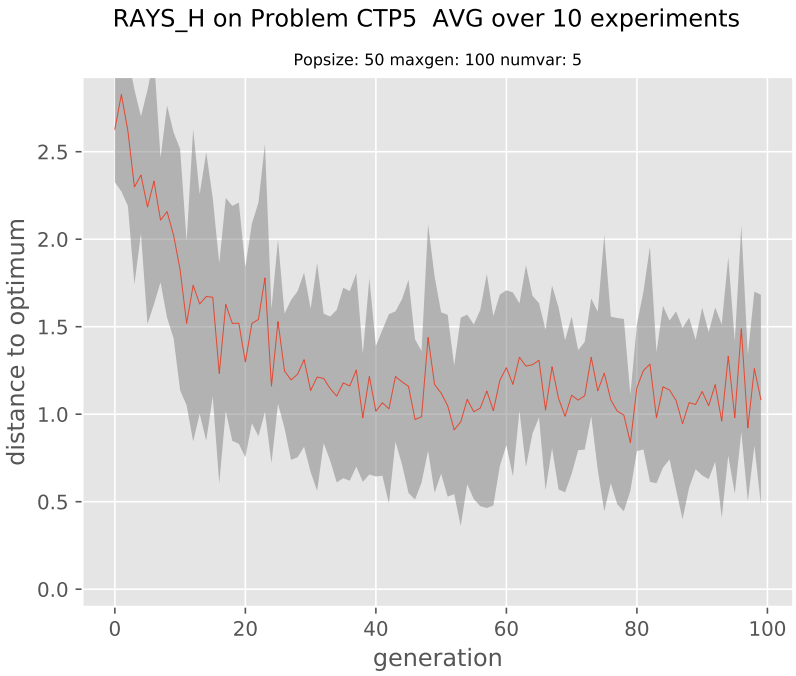

RAYS_M on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

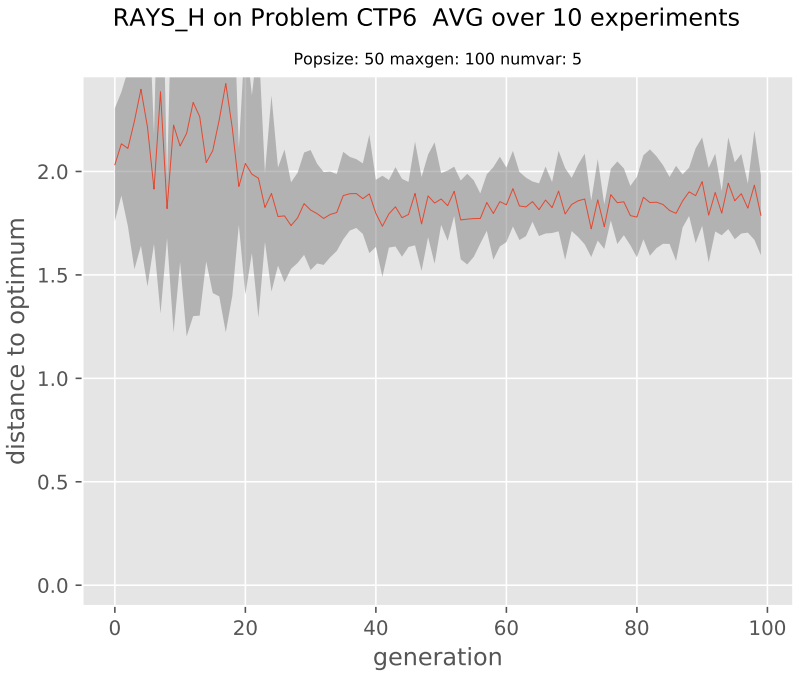

RAYS_H on Problem CTP1 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.15
0.10
0.05
0.00
−0.05
−0.10
0
20
40
60
80
100
generation

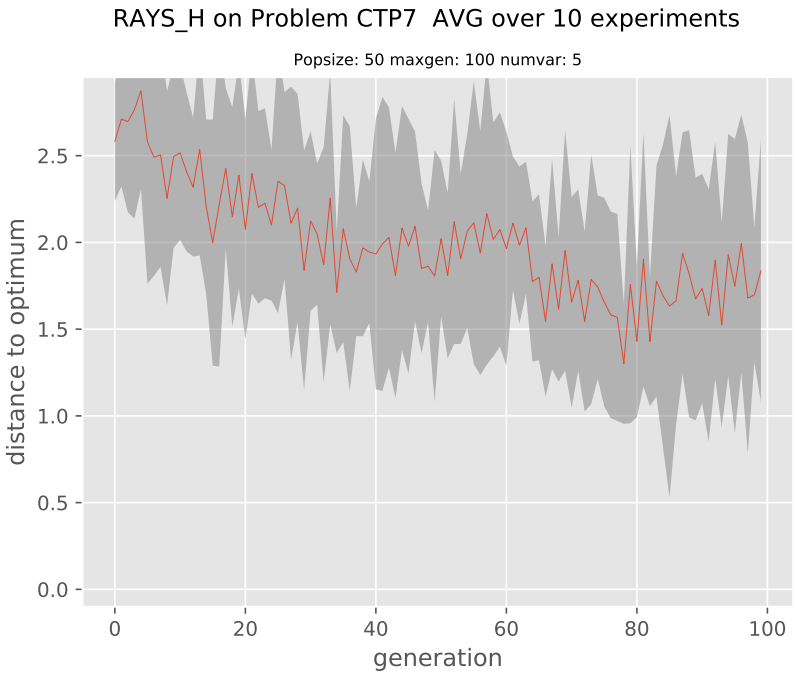

RAYS_H on Problem CTP2 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

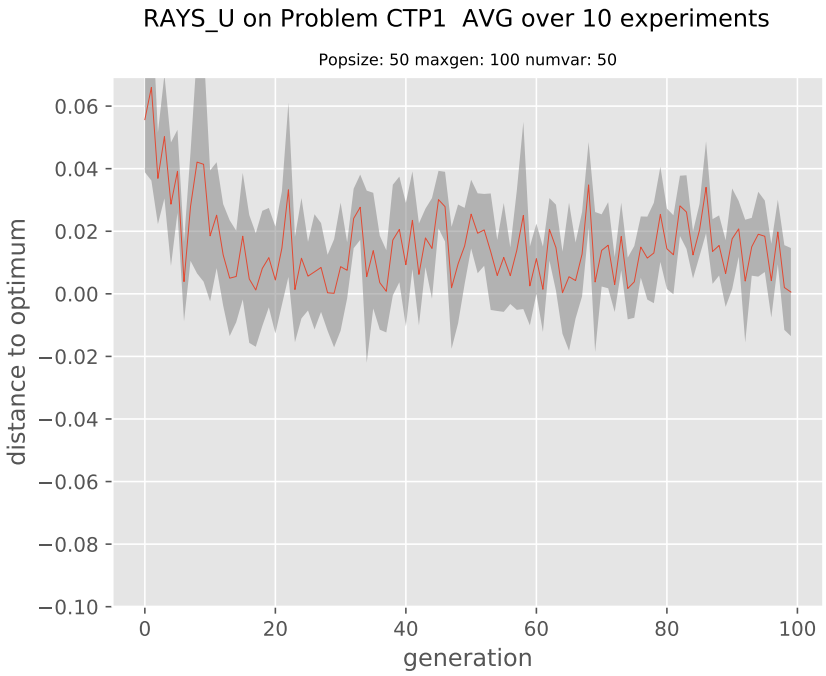

RAYS_H on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
generation


RAYS_H on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
generation

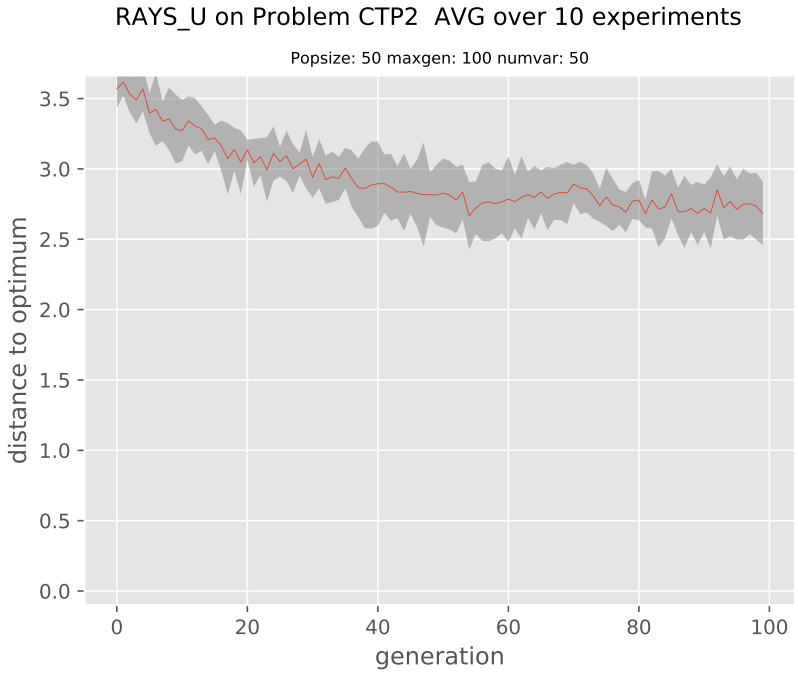

RAYS_H on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
generation

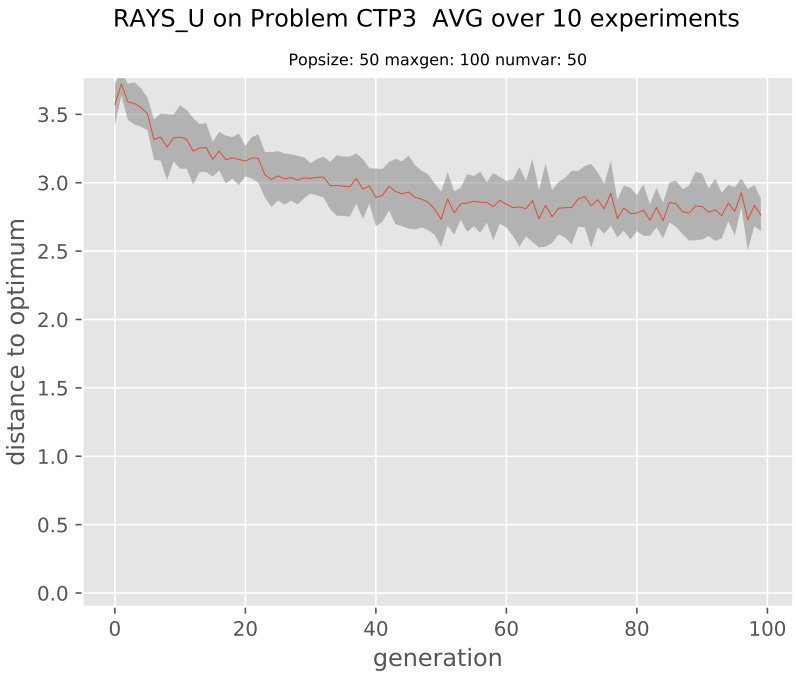

RAYS_H on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
generation

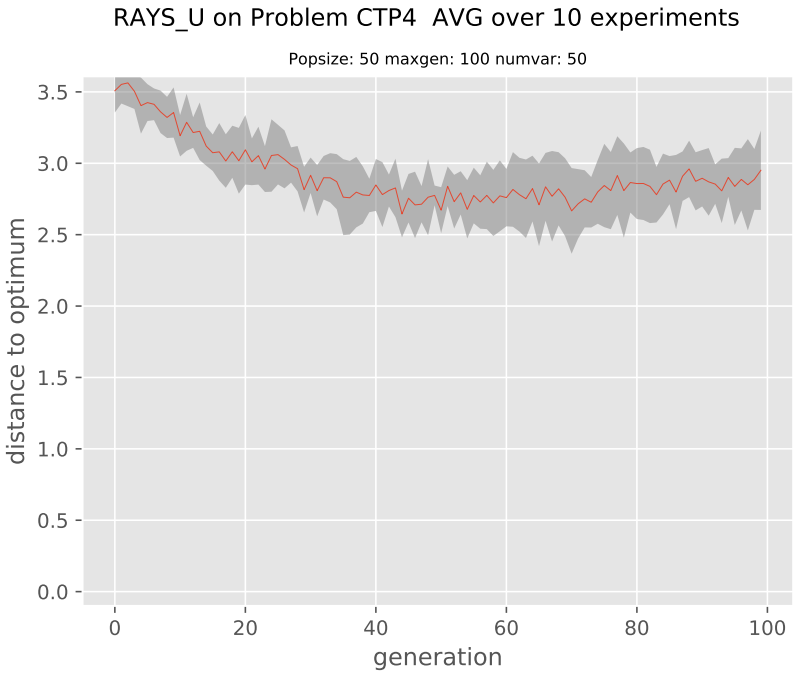

RAYS_H on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 5
distance to optimum
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

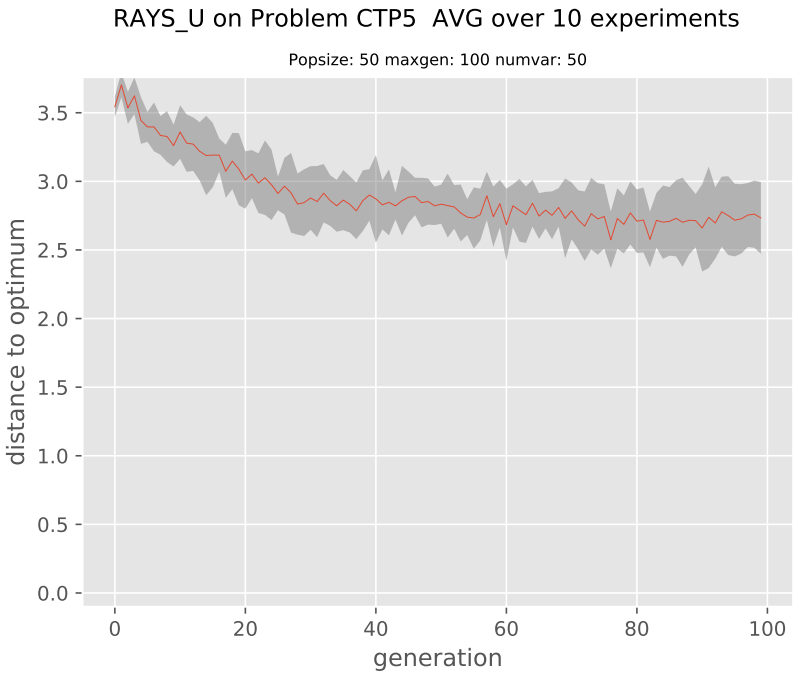


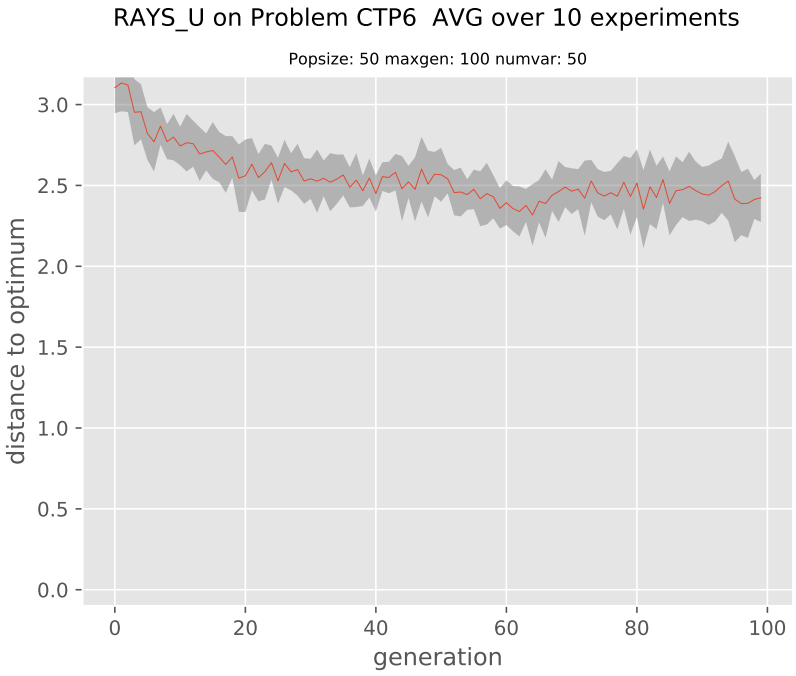
A.5.2 variables 50, generations 100, population 50

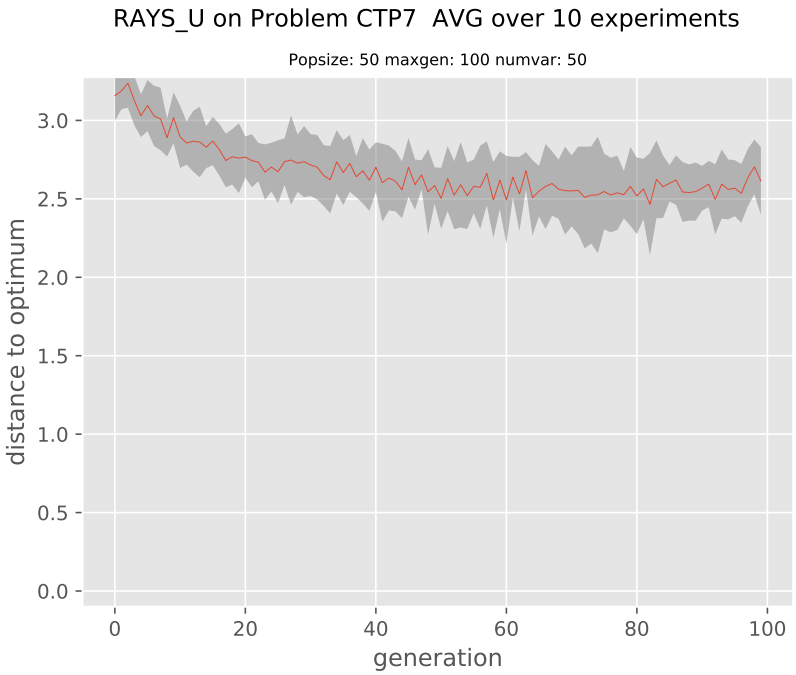


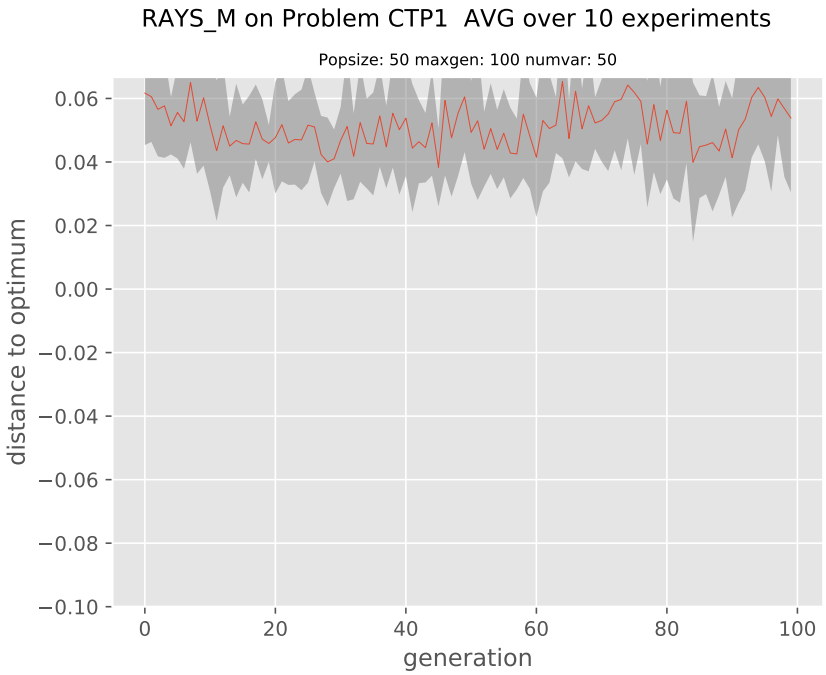

RAYS_U on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

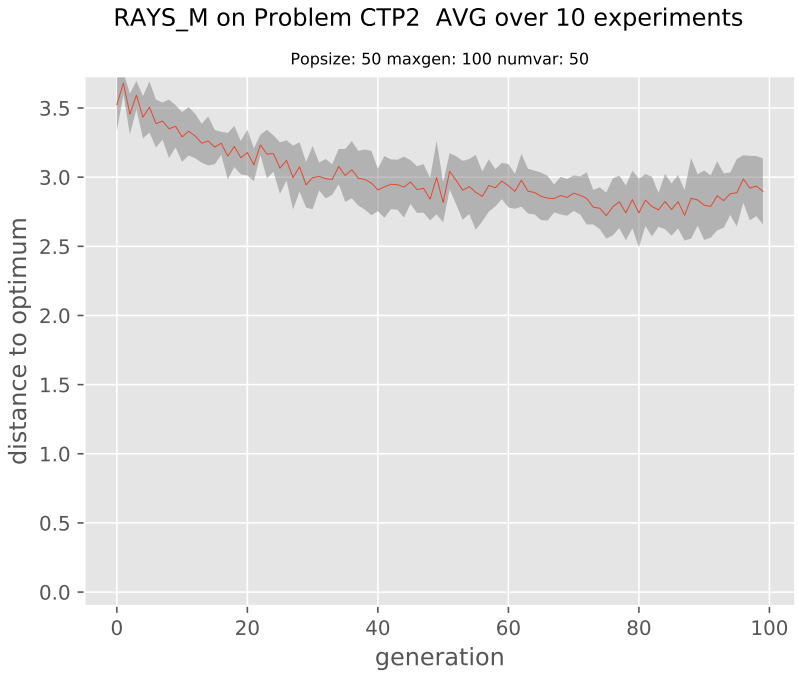

RAYS_U on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

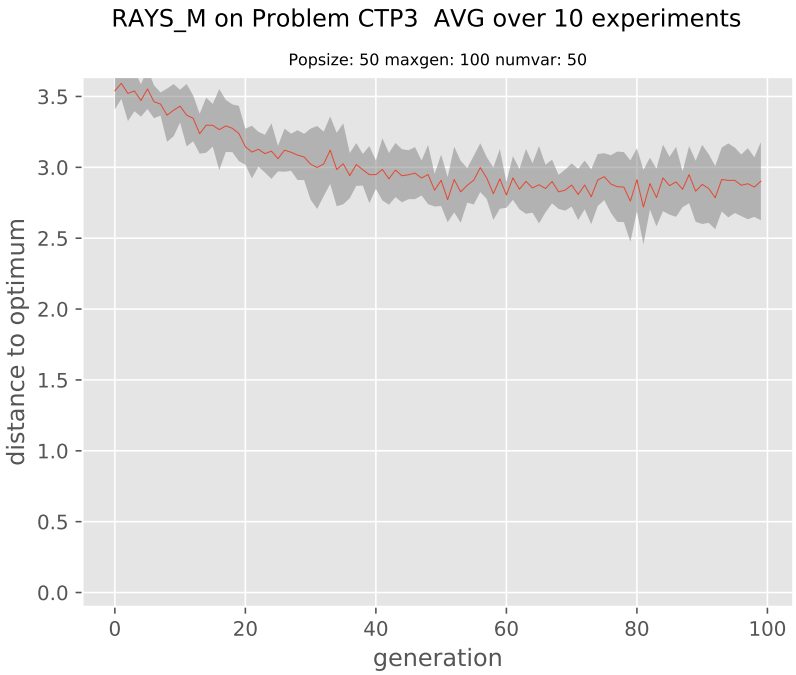

RAYS_U on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

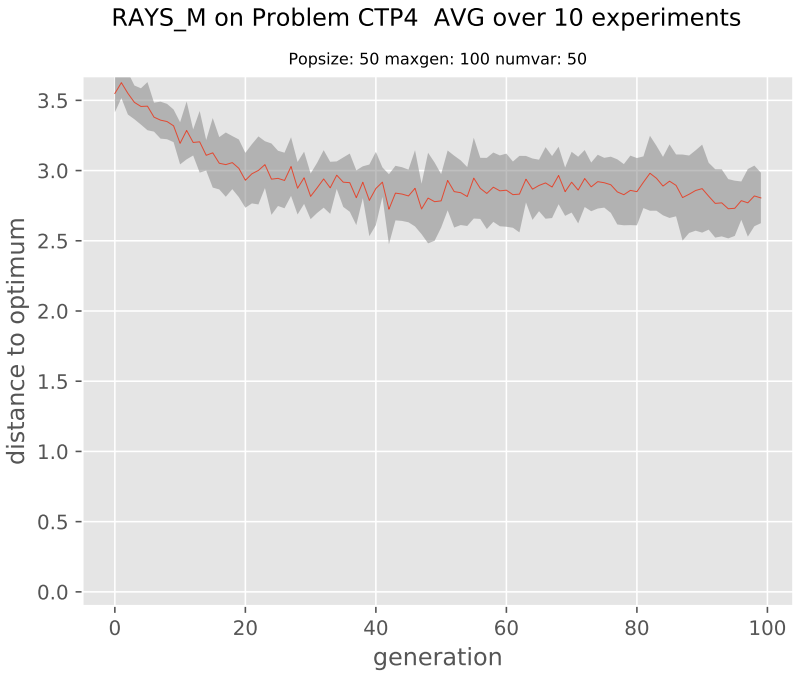

RAYS_U on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

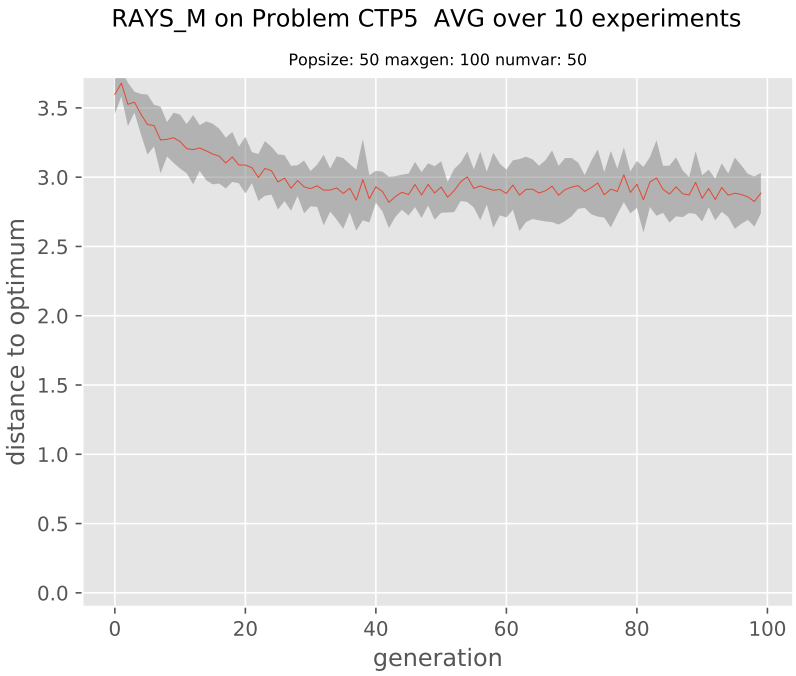

RAYS_U on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

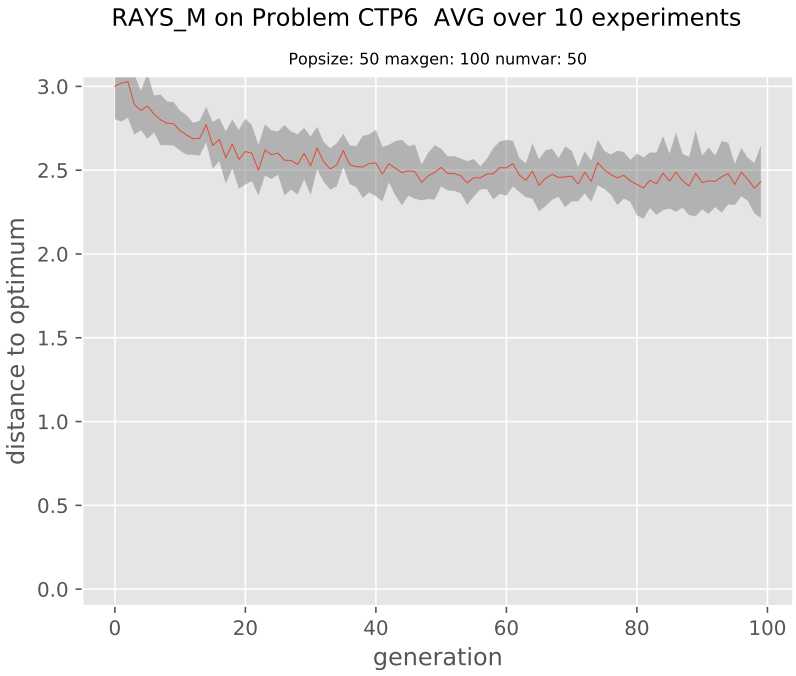

RAYS_M on Problem CTP1 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
0
20
40
60
80
100
generation

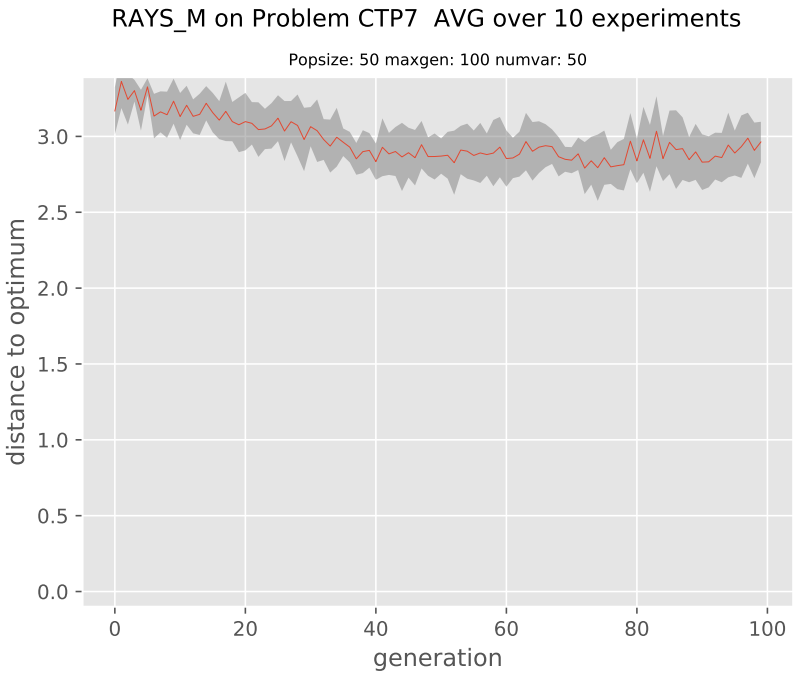

RAYS_M on Problem CTP2 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

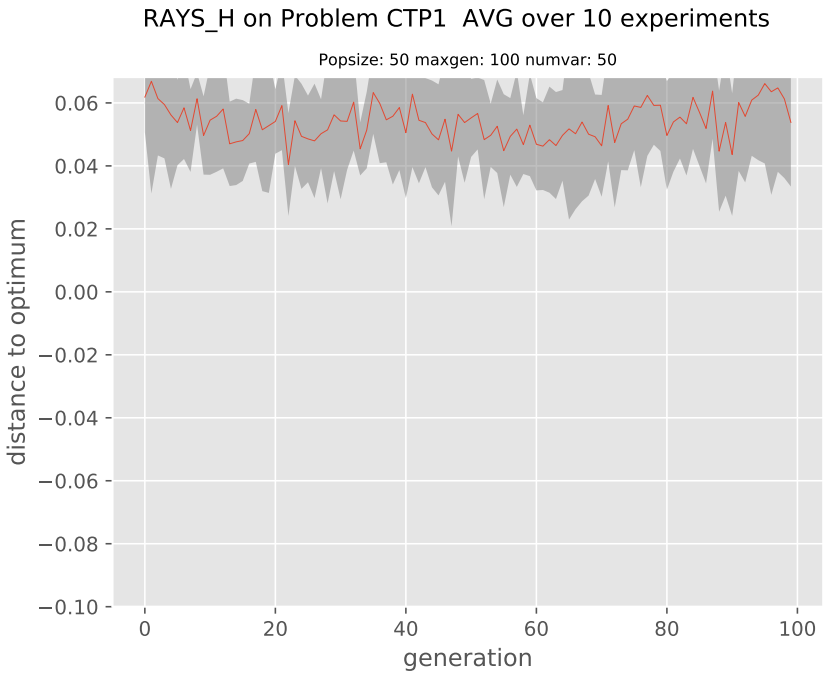

RAYS_M on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

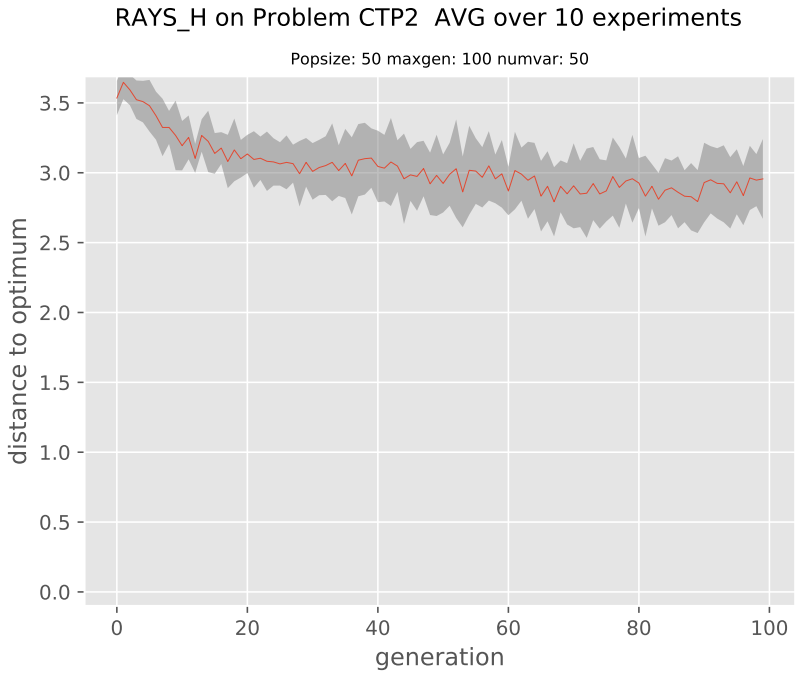

RAYS_M on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

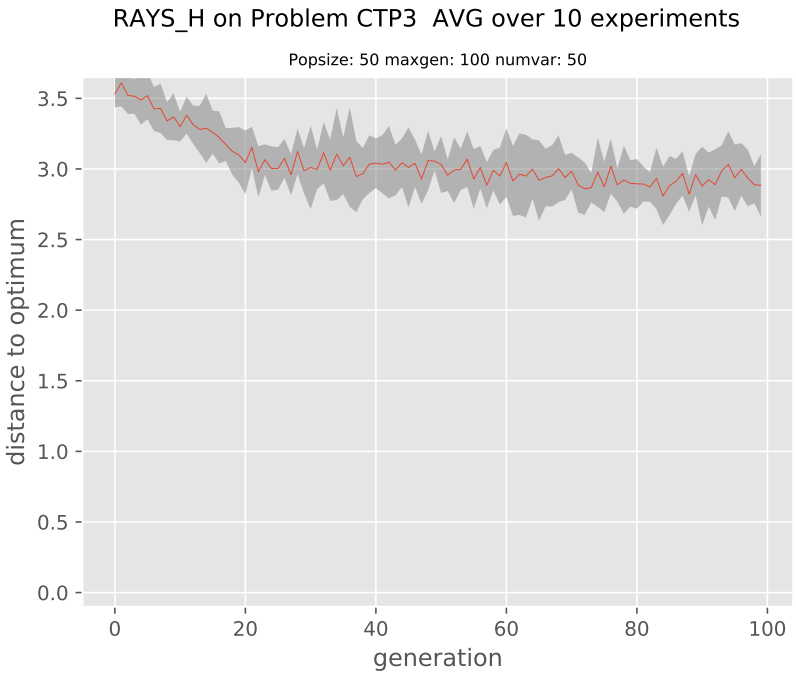

RAYS_M on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

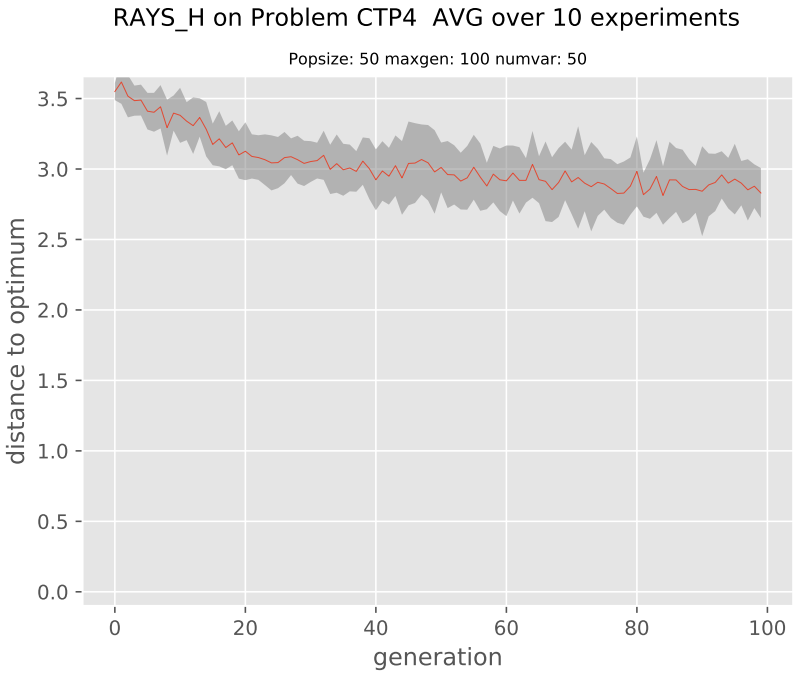

RAYS_M on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

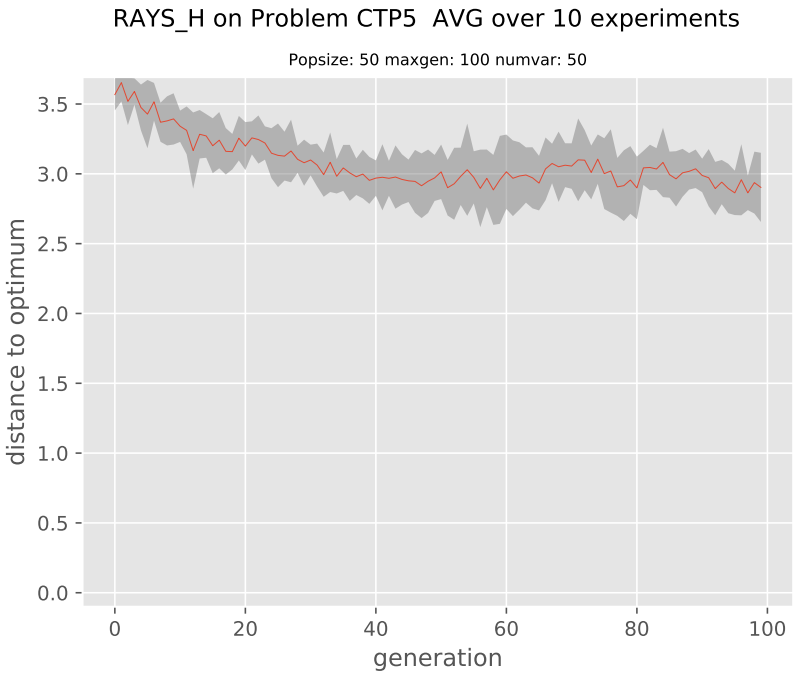

RAYS_M on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

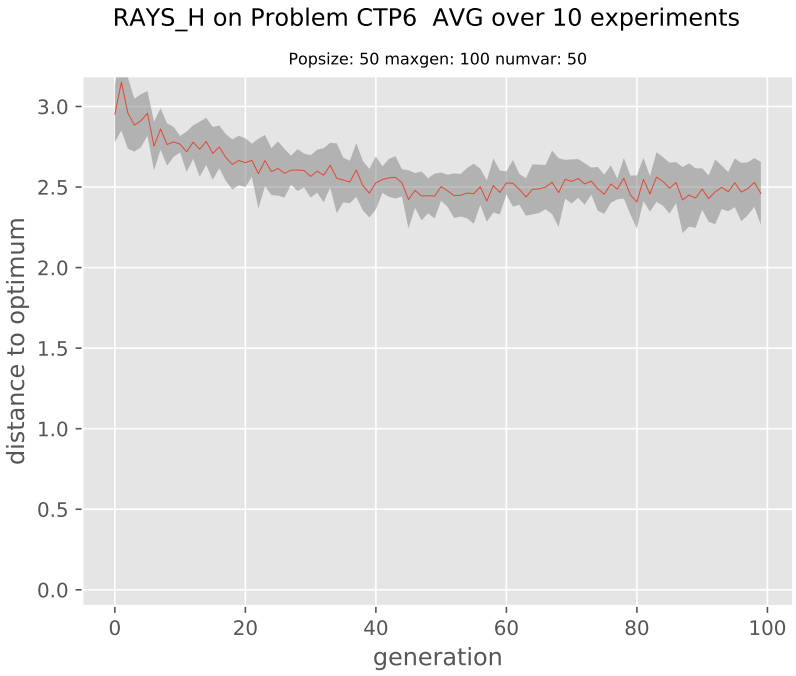

RAYS_H on Problem CTP1 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
0
20
40
60
80
100
generation

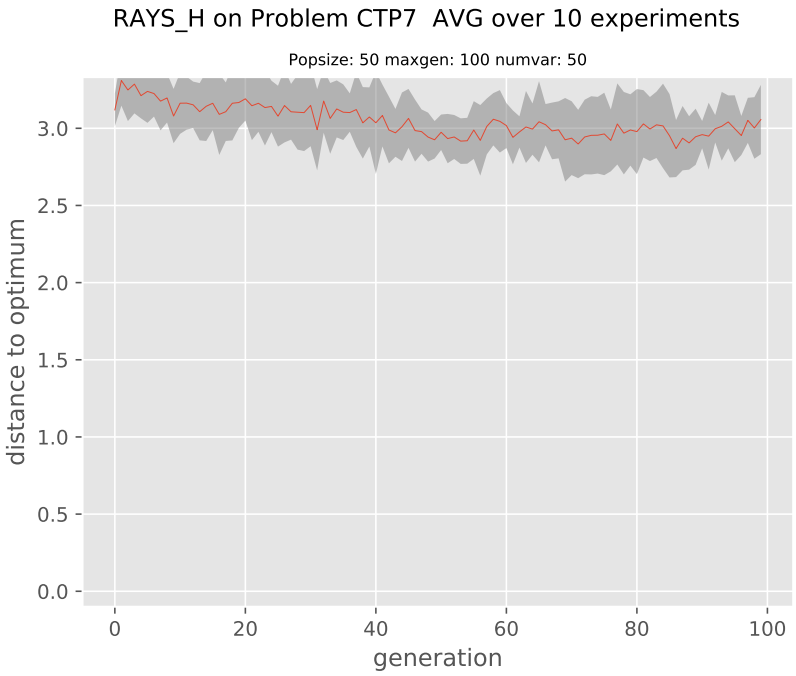

RAYS_H on Problem CTP2 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation


RAYS_H on Problem CTP3 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

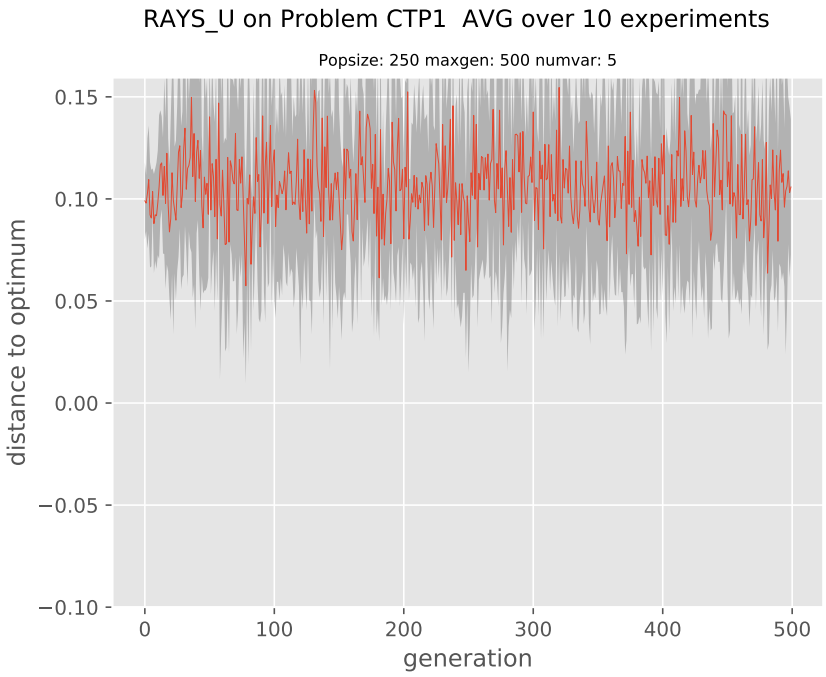

RAYS_H on Problem CTP4 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

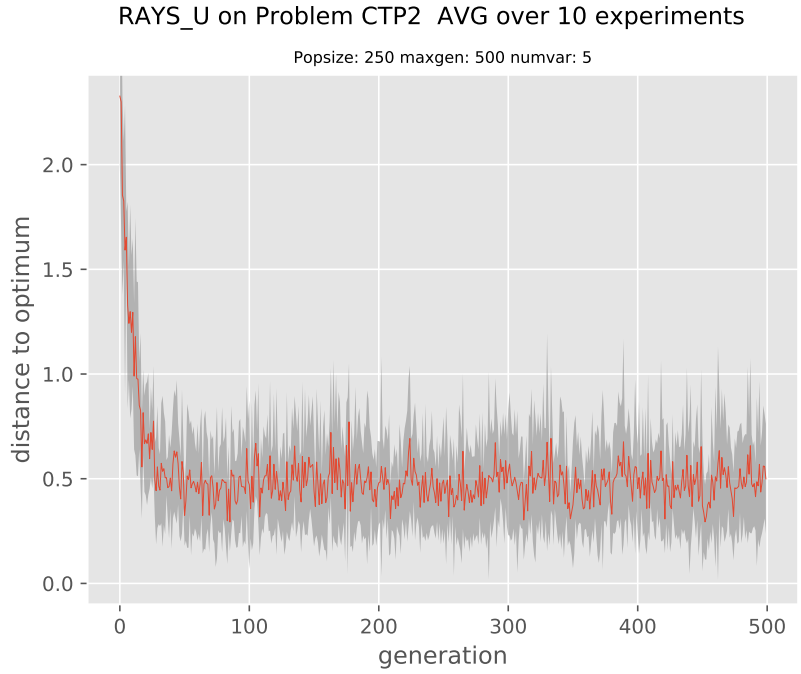

RAYS_H on Problem CTP5 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100

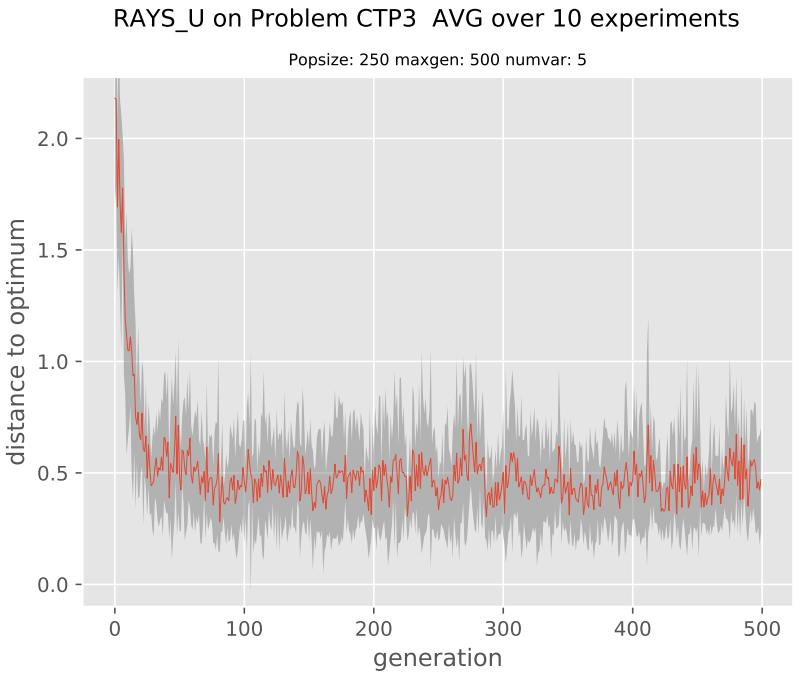

RAYS_H on Problem CTP6 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
generation
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100

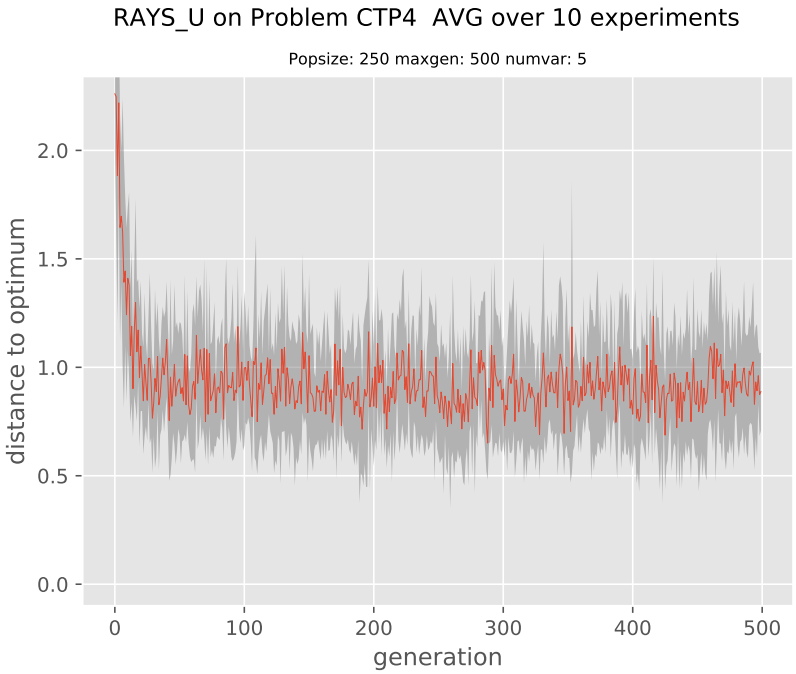

RAYS_H on Problem CTP7 AVG over 10 experiments
Popsize: 50 maxgen: 100 numvar: 50
distance to optimum
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
generation

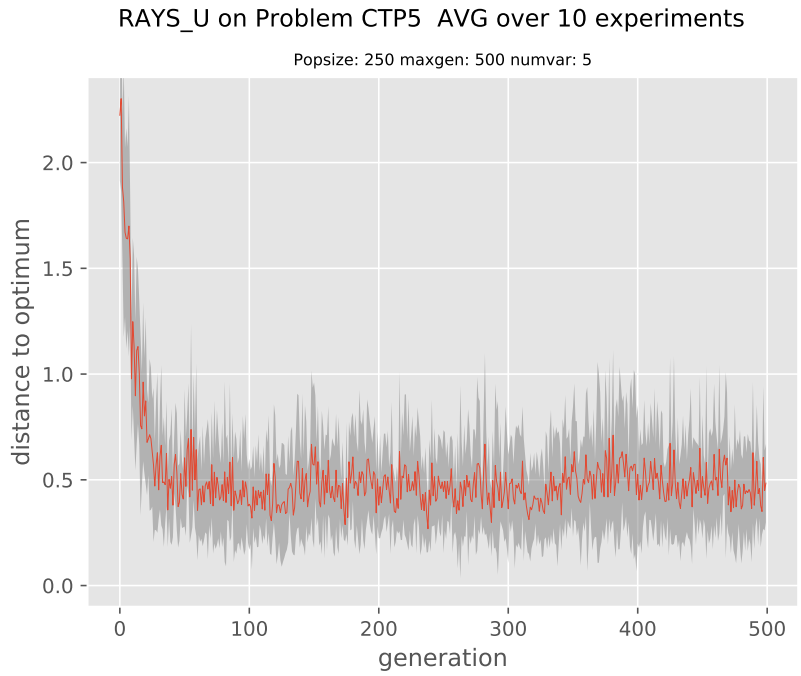
A.5.3 variables 5, generations 500, population 5

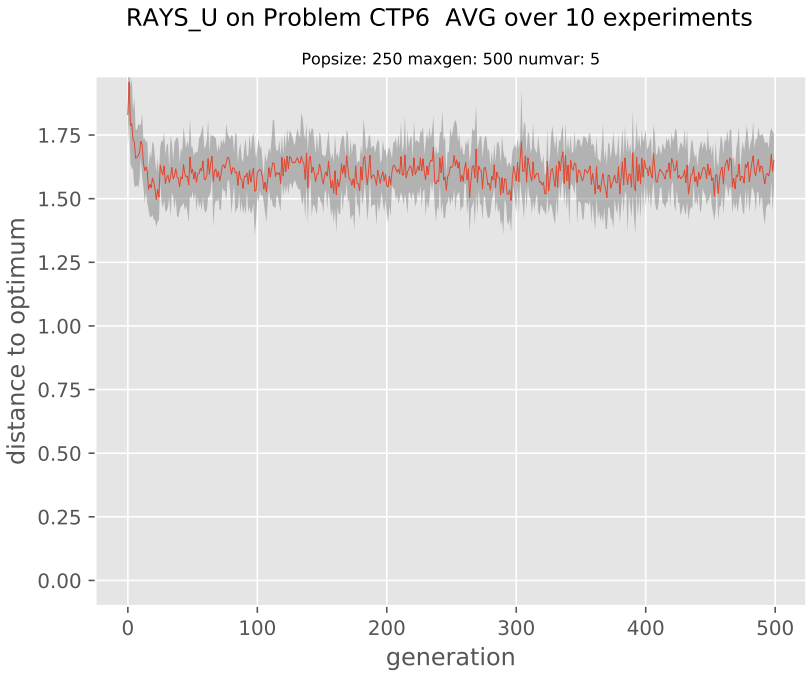


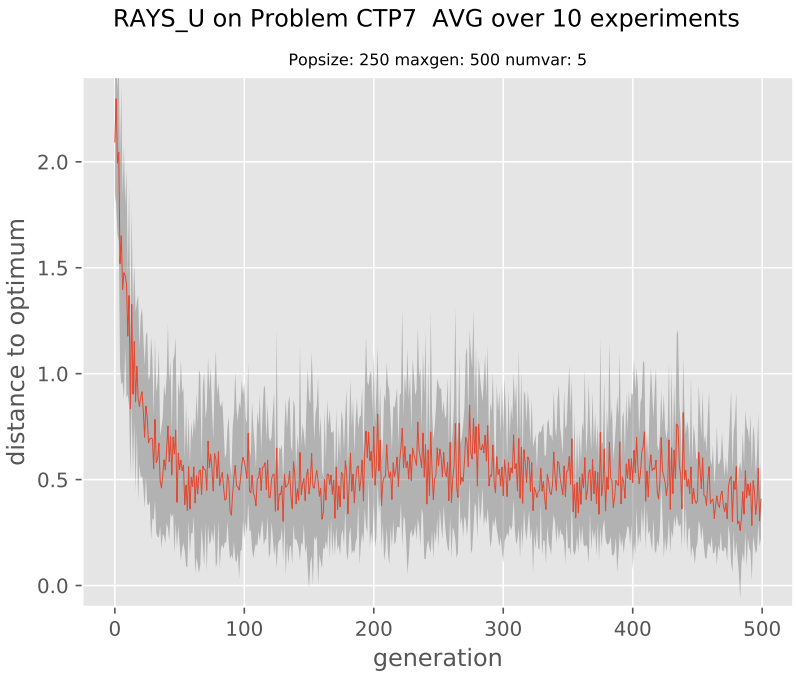


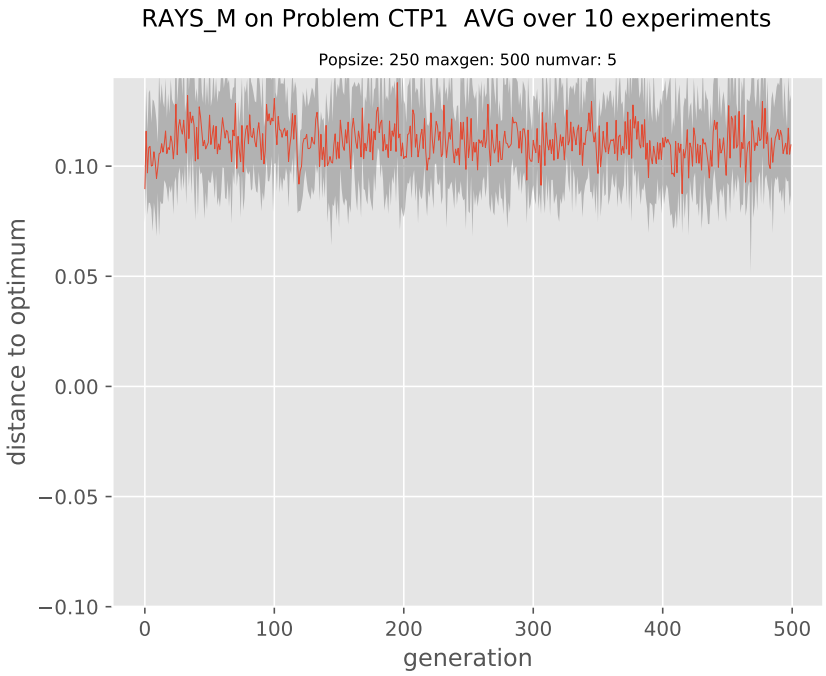

RAYS_U on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

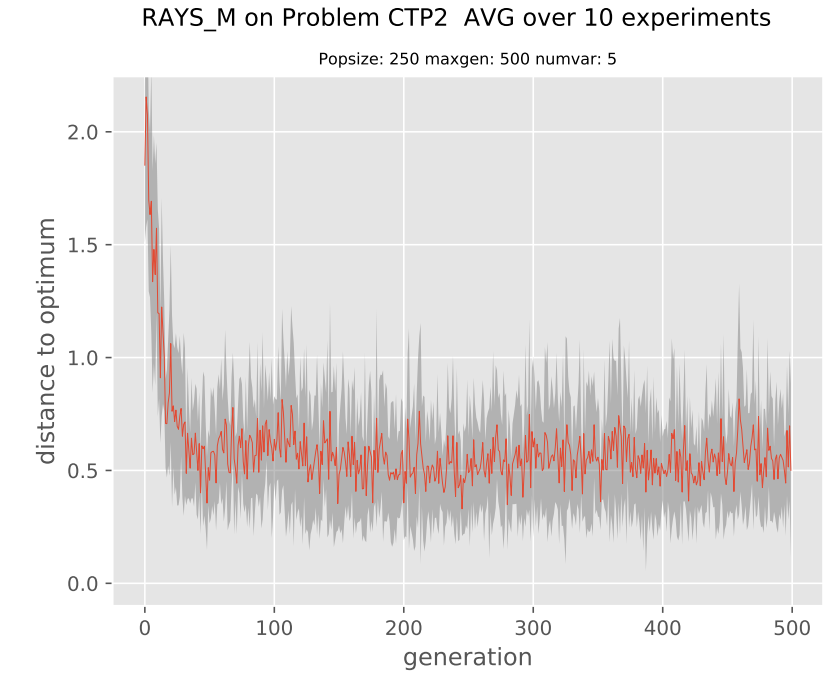

RAYS_U on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

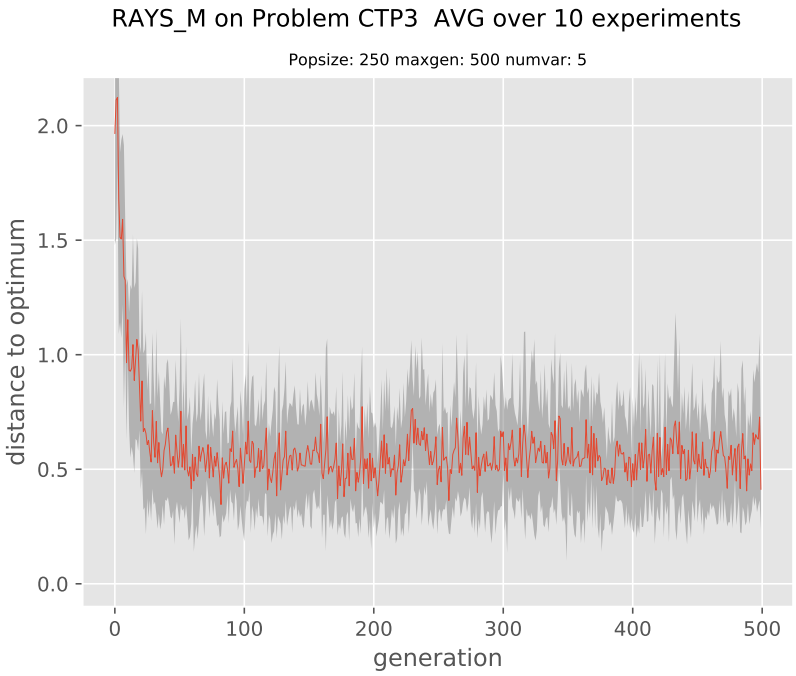

RAYS_U on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

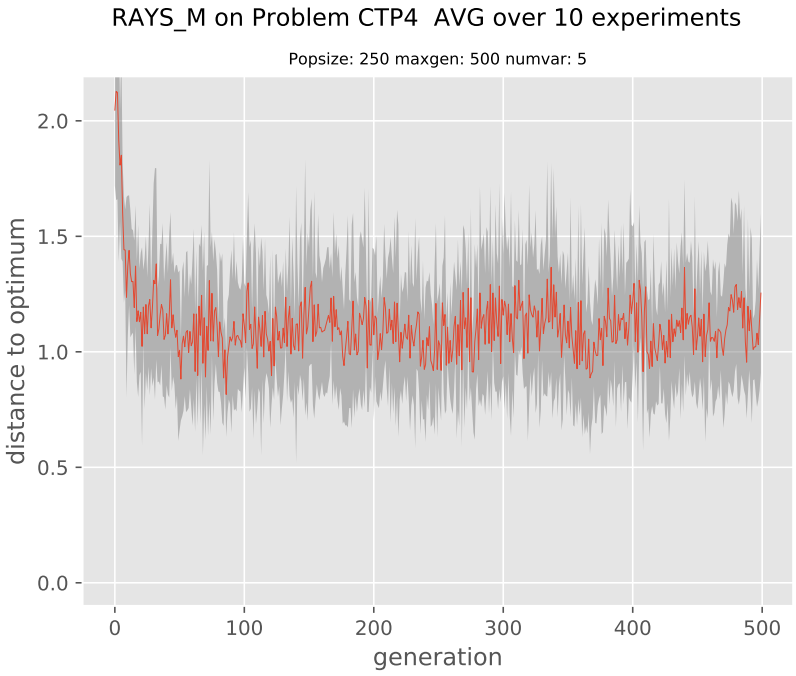

RAYS_U on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

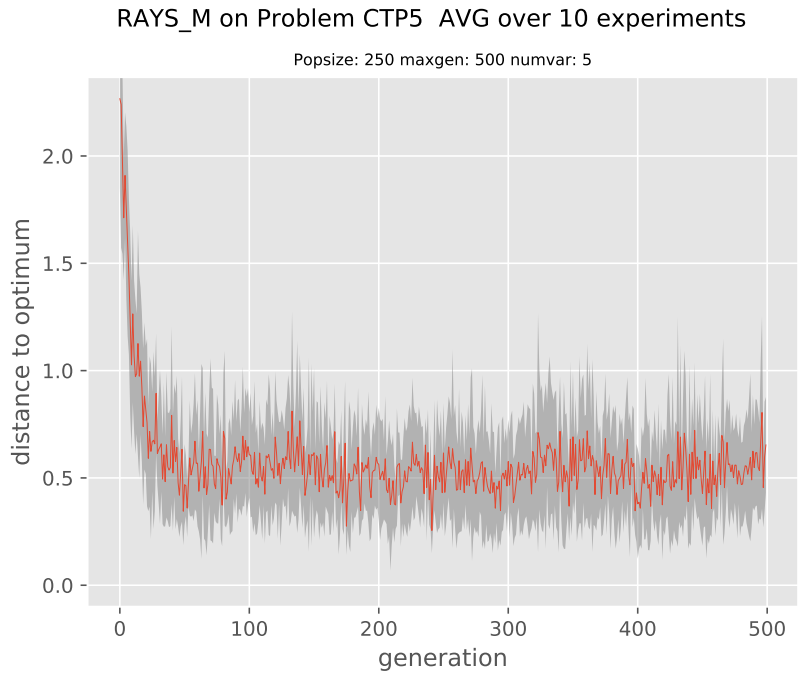

RAYS_U on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

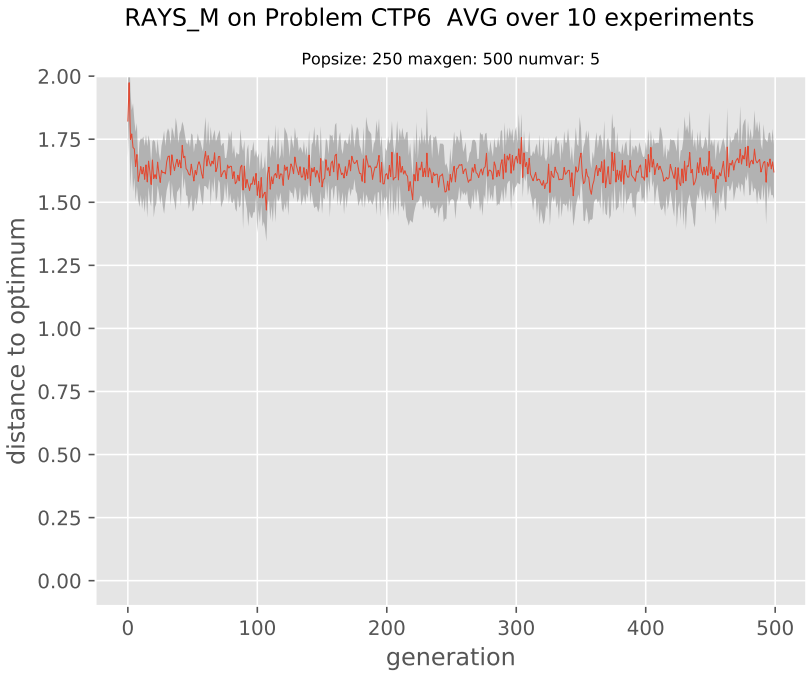

RAYS_M on Problem CTP1 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
0.10
0.05
0.00
−0.05
−0.10
0
100
200
300
400
500
generation

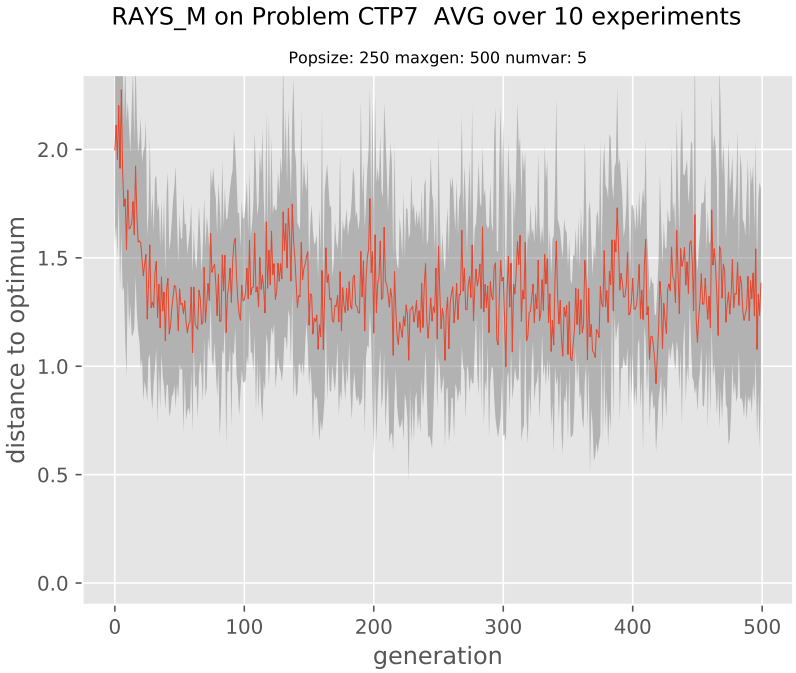

RAYS_M on Problem CTP2 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

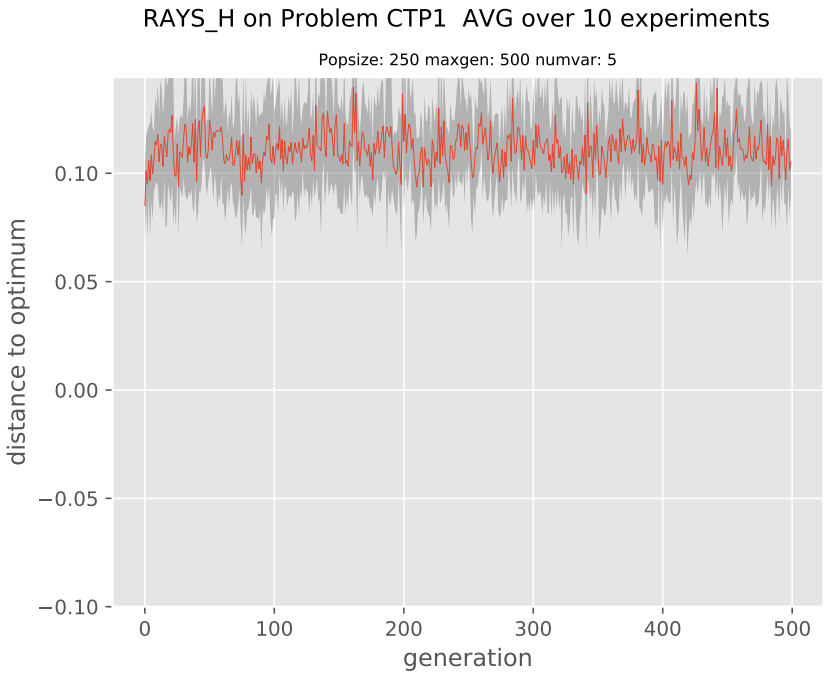

RAYS_M on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

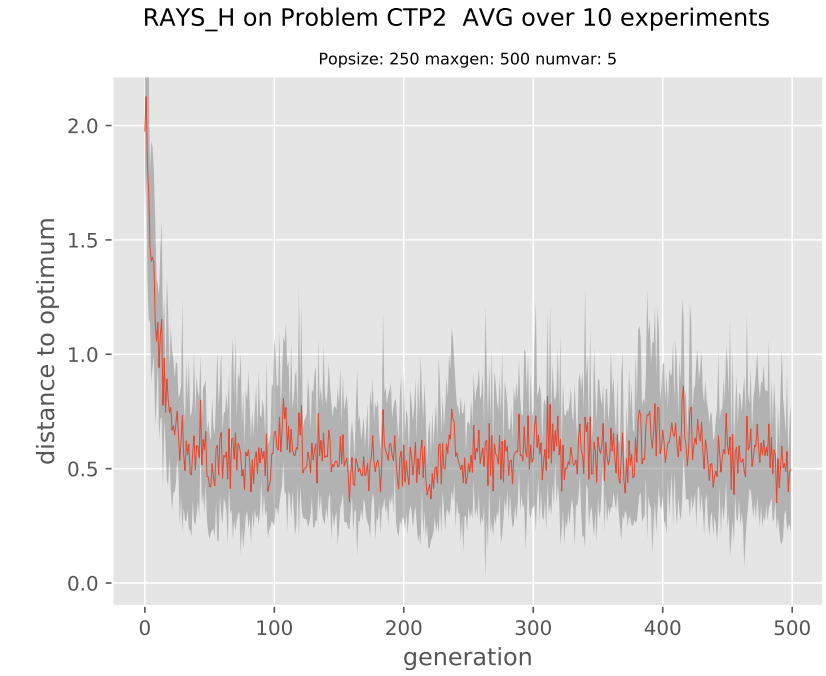

RAYS_M on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

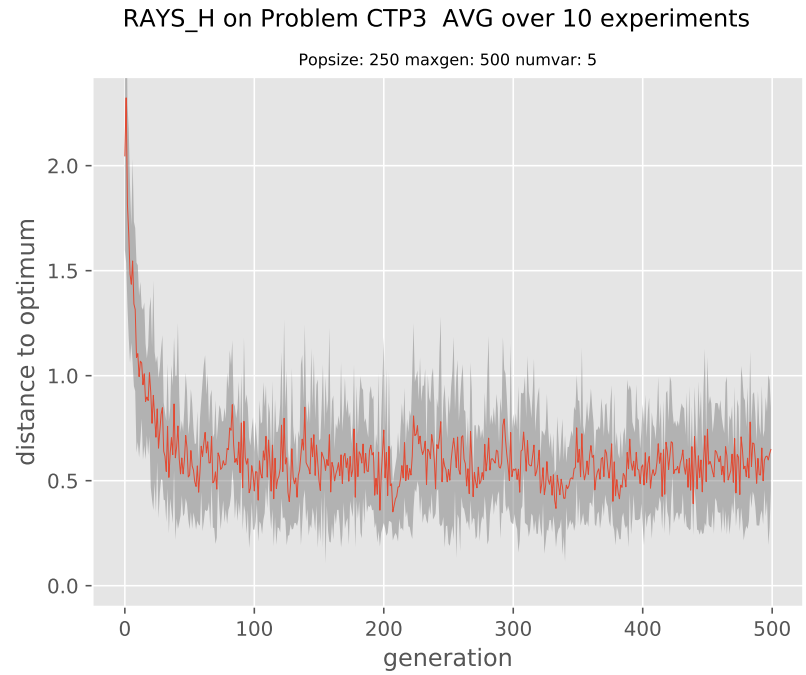

RAYS_M on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

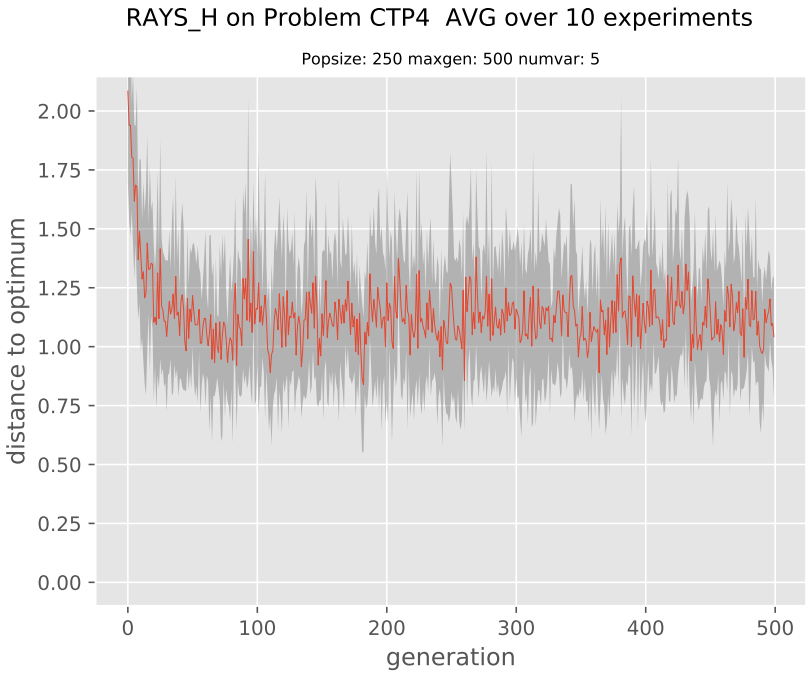

RAYS_M on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation

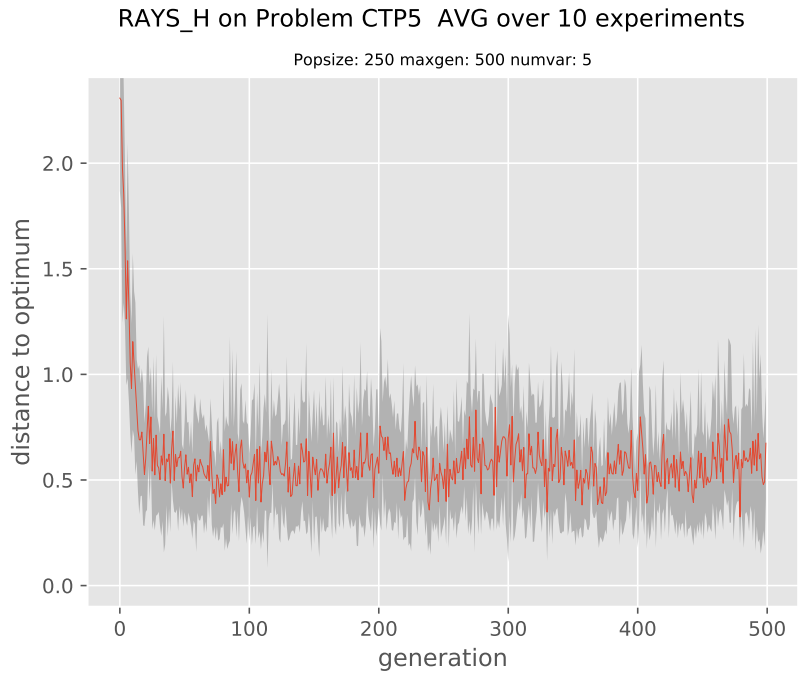

RAYS_M on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation

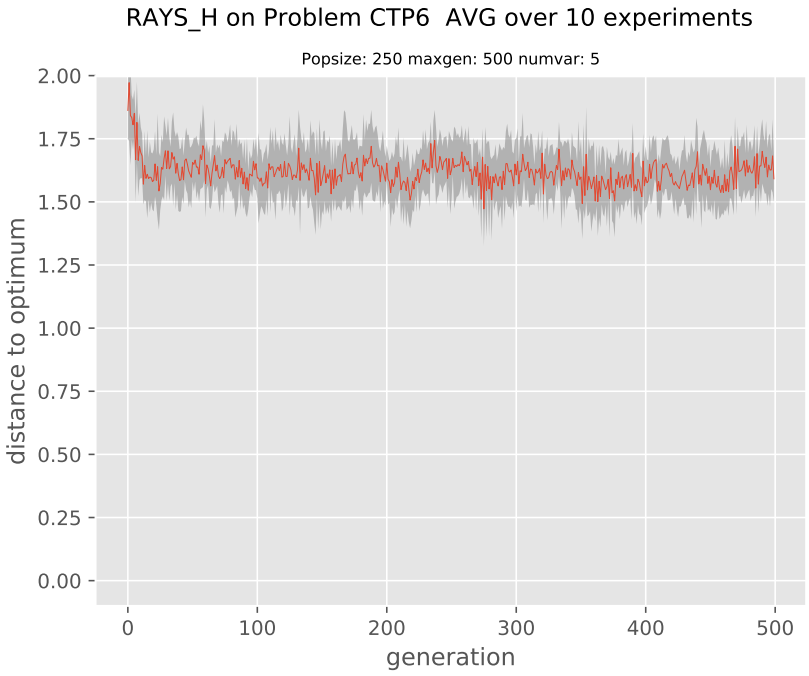

RAYS_H on Problem CTP1 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
0.10
0.05
0.00
−0.05
−0.10
0
100
200
300
400
500
generation

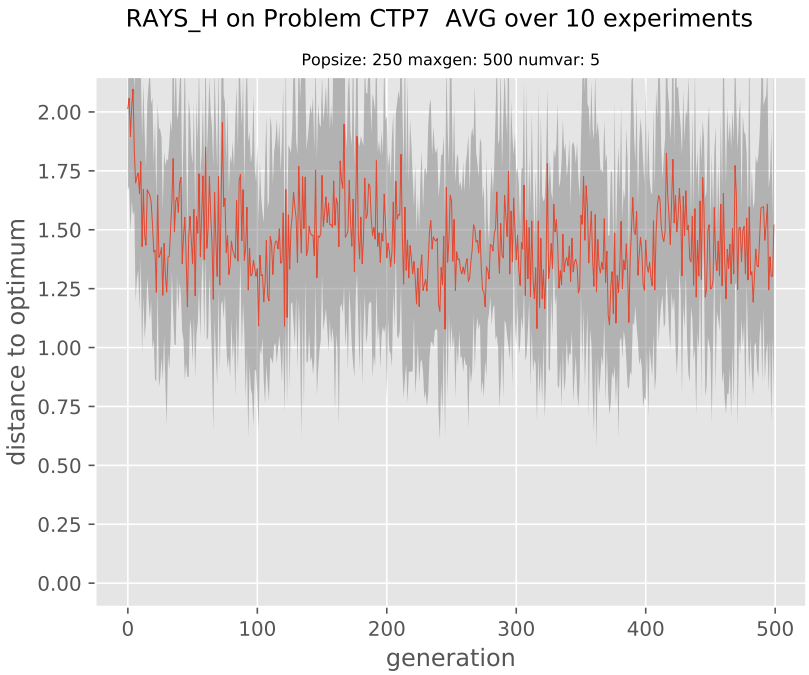

RAYS_H on Problem CTP2 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation


RAYS_H on Problem CTP3 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation


RAYS_H on Problem CTP4 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation


RAYS_H on Problem CTP5 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
generation


RAYS_H on Problem CTP6 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation


RAYS_H on Problem CTP7 AVG over 10 experiments
Popsize: 250 maxgen: 500 numvar: 5
distance to optimum
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500
generation