



Thomas Grandits, BSc

Optimization Algorithms for Satellite Communication Systems

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn., Wilfried Gappmair

Institut für Kommunikationsnetze und Satellitenkommunikation

Dipl.-Ing. Dr.techn., Johannes Ebert
Joanneum Research

Graz, September 2015

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

31.8.15

Date

Gwendit Homoy

Signature

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTÄTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 31.8.15.....

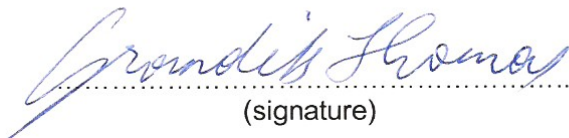

.....
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

31.8.15
.....
date


.....
(signature)

Diese Arbeit ist den vielen Menschen gewidmet die mich während meiner Studienzeit und speziell während dieser Masterarbeit unterstützt haben. Diesen möchte ich hiermit meinen Dank ausdrücken:

Meiner Familie und allen Verwandten, die mir in dieser anstrengenden Zeit mit Rat und Tat zur Seite gestanden haben

Meinen Freunden, welche mir Mut und Motivation verschafften

Allen Wegbegleitern, die mich bis hierher brachten

Weiters gilt mein Dank dem Institut für Kommunikationsnetze und Satellitenkommunikation der Technischen Universität Graz, ebenso der Abteilung Weltraumtechnik und Kommunikationstechnologie von Joanneum Research, wobei ich vor allem Johannes Ebert und Wilfried Gappmair wegen der sehr guten Betreuung erwähnen möchte, sowie Karin Plimon für die vorgeschlagenen Korrekturen

Abstract

Modern satellite communication systems have a vast set of parameters to choose from, like the hardware to use, objectives to maximize, like data-throughput and requirements like availability, or overall cost of the system. The aim of this master thesis was to provide a generic interface to find optimal parameter settings in satellite systems against objectives, while maintaining a set of requirements. In order to achieve this, numerical optimization methods were employed on several satellite communication problems and used to gain knowledge about the potential relationships between the system variables. To provide meaningful results, an existing simulation software, developed by Joanneum Research, was used and extended to simulate theoretical satellite communication systems, in which the generic interface was also embedded to provide the means to optimize any simulation. In a final step, the developed components were used in several experiments of increasing complexity to show the possibilities of optimization on a system level and to shed a new light on the design of multi-beam satellite systems.

Kurzfassung

Moderne Satellitenkommunikationssysteme können vielfältig gestaltet werden, beispielsweise durch die genutzte Hardware, während bestimmte Ziele bestmöglichst erfüllt werden soll, wie die Maximierung der Datendurchsatzrate, gleichzeitig aber Anforderungen ebenfalls berücksichtigt werden müssen (z.B. eine Mindestverfügbarkeit). Aufgabe dieser Masterarbeit war es, eine einfach zu bedienende Optimierungssoftware zu erstellen, um optimale Konfigurationen genannter Satellitenkommunikationssysteme, unter Berücksichtigung der Anforderungen, zu berechnen. Zu diesem Zwecke wurden Numerische Optimierungsverfahren auf die Probleme angewandt und das in diesem Zuge gewonnene Wissen genutzt, um die Beziehungen zwischen den Systemvariablen besser zu verstehen. Um die Sinnhaftigkeit der Ergebnisse zu gewährleisten, wurde ein bestehender, von Joanneum Research entwickelter Simulator genutzt und um die Möglichkeit zur Simulation theoretischer Satellitenkommunikationssysteme sowie um die Schnittstelle zur Optimierung erweitert. Alle entwickelten Komponenten wurden in einem finalen Schritt in mehreren Experimenten kombiniert genutzt, um die Möglichkeiten der Optimierung auf Systemebene zu zeigen und ein neues Licht auf die Chancen beim Design von Multi-Beam Satellitensystemen zu werfen.

Table of Contents

1	Introduction.....	3
1.1	Motivation and Background.....	3
1.2	Overview of the Thesis.....	4
2	Optimization Fundamentals.....	5
2.1	Algorithms.....	7
2.1.1	Downhill-simplex.....	7
2.1.2	Simulated Annealing (SA).....	8
2.1.3	Genetic Algorithms (GA) & Evolutionary Algorithms (EA).....	8
2.1.4	Particle Swarm Optimization (PSO).....	9
2.1.5	Artificial Bee Colony.....	9
2.1.6	Nondominated Sorting Genetic Algorithm II (NSGA-II).....	10
3	Related Work.....	11
3.1	Parameters in Optimization.....	13
3.1.1	Algorithm Portfolios.....	13
3.1.2	Self Adaptation.....	16
3.1.3	Meta Optimization.....	17
4	Design.....	19
4.1	Simulator.....	19
4.2	Optimizer.....	22
4.3	Link Budget.....	23
4.4	Multi-Beam Systems.....	29
4.5	Benchmark.....	35
4.6	Output & Result Evaluation.....	36
4.7	Planned Algorithms.....	37
5	Implementation.....	38
5.1	Current Structure.....	38
5.1.1	Simulation Core.....	38
5.1.2	Burst Container.....	38
5.1.3	Simulation Main Loop.....	38
5.1.4	Sections and Multi Threading.....	39

5.2 Restructuring.....	39
5.2.1 Simulation Core.....	40
5.2.2 Burst Container.....	40
5.2.3 Simulation Main Loop.....	40
5.2.4 New Modules.....	40
5.2.5 Matlab Tool.....	47
5.3 Final Class Diagrams.....	53
6 Evaluation.....	56
6.1 Benchmark.....	56
6.2 Simple Crossover Experiment.....	58
6.3 Advanced Interference Mitigation Experiment.....	64
6.4 System Cost Experiment.....	66
7 Conclusion and Future Work.....	73
Bibliography.....	76

List of Listings

Listing 4.1: Simple Simulation File Example.....	20
Listing 4.2: Result File of the Simulation shown in Listing 4.1.....	20
Listing 4.3: Simulation File with Optimization Tags.....	22
Listing 5.1: Exemplary instantiation of the Optimizer Module.....	41
Listing 5.2: Code to update the Velocity of each Particle, according to Equation (2.5).....	42
Listing 5.3: Exemplary Text Output File of an Optimization.....	44
Listing 5.4: Final Version of the Link Budget Modules.....	44
Listing 5.5: Interference Modules.....	45
Listing 5.6: Command to call the Script for Processing the Binary Files.....	49
Listing 5.7: Script to interpolate the Surface from the Function Evaluations.....	51
Listing 6.1: Output of the Optimization, showing the best Results.....	62

List of Figures

Figure 2.1: Visualization of a Pareto Frontier. The lines show the range of domination. From Wikipedia on Pareto-efficiency [28].....	6
Figure 2.2: Nelder Mead Optimization on the Example of a Half-Sphere.....	8
Figure 3.1: Reconstructed Far Field Radiation Pattern for the Pyramidal Horn. (a) Amplitude. (b) Phase, from [8].....	11
Figure 3.2: Cost Function Evolution over the Number of Function Evaluations.....	14
Figure 3.3: Cost Evolution Function, with Lines created through Linear Regression.....	14
Figure 3.4: Created Distribution through the Function 'ksdensity'.....	15
Figure 3.5: Evolution of the Probabilites in the SaDE algorithm. From [13].....	17
Figure 4.1: Preliminary Class Diagram of the additional Optimization Modules.....	23
Figure 4.2: Simplified View of a Satellite System.....	24
Figure 4.3: 4+12APSK Modulation Symbol Positions from [27].....	28
Figure 4.4: Simplified View of a Satellite System, with a few possible variables, constraints and an objective.....	29
Figure 4.5: Exemplary Generated Radiation Pattern [dBi], Cartesian representation.....	30
Figure 4.6: Closeup of Figure 4.5 on the Main Lobe with marked Crossover Angle of 3dB.....	31
Figure 4.7: Visualization of a Spot Beam.....	31
Figure 4.8: Hexagonal Spot Beam Pattern in use.....	33
Figure 4.9: Normalized C and I values for the scenario shown in figure 4.8, from [23].....	33
Figure 4.10: Planned Modules and Information Flow between them.....	35

Figure 5.1: Visualization of the Workflow of the Simulation with different Sections.....	39
Figure 5.2: Hexagon Pattern in use.....	46
Figure 5.3: Surface of Function Number Eight of the CEC 2013 challenge [24].....	50
Figure 5.4: Interpolated Surface Reconstruction. Blue Dots mark the sampled Points by the PSO algorithm.....	52
Figure 5.5: Performance Evaluation for every Iteration of every Particle.....	53
Figure 5.6: Class Diagram of the Optimization Classes.....	54
Figure 5.7: Class Diagram of the Link Budget and Interference Classes.....	55
Figure 6.1: Best, Worst and Average Performance of PSO on the Sphere Function.....	57
Figure 6.2: Best, Worst and Average Performance of Nelder-Mead on the Sphere Function.....	57
Figure 6.3: Crossover vs Avg Goodput on the whole Surface.....	63
Figure 6.4: Second Experiment, comparing different Frequency Reuse Settings.....	65
Figure 6.5: Modcods and their required linear SNR (Threshold).....	66
Figure 6.6: Point Cloud of the Third Experiment: Varying the EIRP and Crossover.....	71
Figure 6.7: Point Cloud of the Third Experiment with the Constraint 'Availability > 99%'.....	72

List of Tables

Table 3.1: Covariance Matrix of 5 Algorithms on a Test Set of 10 Functions, from [16].....	16
Table 5.1: Current Meta Data Structure.....	48
Table 6.1: Modcod Thresholds from [27].....	59
Table 6.2: Distribution of the Atmospheric Attenuation, Q-Band, Location Graz.....	60
Table 6.3: Distribution of the Atmospheric Attenuation, V-Band, Location Graz.....	60
Table 6.4 Best found Solution for the Cost Experiment without any Constraints.....	68
Table 6.5 Optimal Solution after adding the Availability Constraint to the former Case.....	69
Table 6.6 Multiple Optimal Cases found after adding the Bandwidth as a Variable.....	69

1 Introduction

1.1 Motivation and Background

Numerical Optimization is a research field, motivated by the need to approximate solutions or automate the process of solving complex, non-linear systems. The need can arise from the difficulty to analytically calculate the minima/maxima, or the missing knowledge about the cost function. As such, the optimization algorithm tries to use only little information from the function to optimize (usually only the first few derivatives), or none at all.

More complex systems do not even provide closed formulas, as is the case with the Simulation Software at hand. Luckily, adapting and optimizing the parameters of such simulations is inherently not different for most derivative-free optimizers, as long as we still retain a proper cost function to optimize.

The Simulation Software of Joanneum Research, at which the developed optimization algorithms are aimed, allows easy composition and rearrangement of simulation components. This is meant to provide a universal tool for creating satellite scenarios to simulate. This comes at the before mentioned cost when optimising: Instead of deriving equations to solve and tuning optimization parameters to the task, one needs to find a robust algorithm, suitable for almost all simulations, or an operator who intelligently chooses an algorithm from the available ones.

Still, the simulations may be more demanding to compute, leaving us with less function evaluations within the given time budget. When dealing with simulations of satellite systems, we could try to analyse the composed simulation with great detail, derive closed formulas and try to design efficient optimization for given simulation, but there is the need to freely change the simulation, its components and parameters, which might leave the optimization useless once again.

Alternatively, one could resort to choosing one optimization algorithm for all simulations as a trade-off between speed and quality, but it would probably only provide adequate performance in the average case, while also being difficult to find.

Yet better, we could provide multiple algorithms and let the user, or the program intelligently choose one. This master thesis tries to design and implement such an optimizer and integrate it into the existing simulation software, which we look into further detail in section 4.1. The optimizer will then be applied to solve practical problems in satellite communications, which should offer insight into the problem domain. The gathered data may then be used to reconstruct some parts of the cost function.

These optimization methods can and will be employed in the Q/V-Band of the Alphasat

project [1] to offer further insight into this new domain.

1.2 Overview of the Thesis

Section 2 will deal with the fundamental theorems and definitions of optimization, to lay the foundation of the thesis.

In section 3, we will analyze existing approaches to deal with complex optimization. We will present the various algorithms and different approaches, which have been developed and weigh their advantages against the disadvantages for our application. Some metrics and definitions from previous works will find their way in this thesis.

Section 4 will present the concept, developed after consulting existing literature and choosing an appropriate way to optimize simulations. We will start with the simple optimization of a link-budget, which will be presented in detail in section 4.3.

After that, section 5 will document the more concrete way of implementing the algorithms. Changes and deviations from the initially designed concept will also be documented there, as well as additional features which were implemented.

Section 6 will provide an in-depth review of the retrieved results, for both the optimization performance, as well as the calculated optimal solutions for the given satellite system scenarios.

Finally, in section 7, we will summarize the knowledge obtained, explain shortcomings in our work and give hints for future, possible works.

2 Optimization Fundamentals

When optimizing functions, we either try to maximize a given value, usually called fitness, or try to do minimize it, which is then usually declared as cost. The optimization of the function is pursued by the means of an iterative algorithm that tries to approach the global optimum in every iteration.

All free parameters open to variation are contained in the vector \mathbf{x} , which we usually refer to as feature vector. Other common names are variable vector, unknowns, or parameters. We call the whole space, described by \mathbf{x} , the feature space. Our task is to find a feature vector that minimizes the function, where we usually have to keep in mind some constraints, limiting the feature space. We use the notations and definitions from [2]:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad \text{subject to} \quad \begin{cases} c_i(\mathbf{x}) = 0 \\ c_j(\mathbf{x}) \geq 0 \end{cases} \quad (2.1)$$

f is called the objective function, while c_i and c_j are the equality and inequality constraint functions respectively. The space which satisfies all constraints c_i and c_j is called feasible region, or feasible space.

Many algorithms and papers deal with global optima, while avoiding local optima. A useful definition of these terms is also given in [2], which we will cite here:

“A point \mathbf{x}^ is a global minimizer if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$.”*

“A point \mathbf{x}^ is a local minimizer if there is a neighborhood N of \mathbf{x}^* such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for $\mathbf{x} \in N$ “(Note: N is not to be confused with the natural numbers)*

Optimization problems are not limited to real number problems as specified above, but it is a very common optimization problem. We can conclude that a global minimum is the special case of a local minimum, where the neighborhood extends over the whole feature space. Determining if the current local minimum is a global minimum is of course not an easy task and has extensively been researched in past works.

If we want to maximize an objective function, we can also apply the same theorems and algorithms as for minimization since:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \max_{\mathbf{x}} -f(\mathbf{x}) \quad (2.2)$$

Until now, we only spoke of the so-called single objective optimization, where only one objective function exists. If we extend this theory on multiple objective functions, we have to add a few definitions. Now $F(\mathbf{x}) = \{ f_1(\mathbf{x}), \dots, f_n(\mathbf{x}) \}$ is the objective vector. Unfortunately, in most cases we cannot find a \mathbf{x} , such that we minimize all elements of the vector $F(\mathbf{x})$, but we can use the definition of dominating points to rank individual solutions.

$$\begin{aligned}
 F_1(\mathbf{x}) = \{ f_{1,1}(\mathbf{x}), \dots, f_{1,n}(\mathbf{x}) \}, \quad F_2(\mathbf{x}) = \{ f_{2,1}(\mathbf{x}), \dots, f_{2,n}(\mathbf{x}) \} \\
 \forall i: f_{1,i}(\mathbf{x}) \leq f_{2,i}(\mathbf{x}) \wedge \exists i: f_{1,i}(\mathbf{x}) < f_{2,i}
 \end{aligned}
 \tag{2.3}$$

Practically speaking, equation (2.3) says that all cost functions of $F_1(\mathbf{x})$ are at least as good as the cost functions in $F_2(\mathbf{x})$ while there is also at least one cost function, which is better. If equation (2.3) holds, solution $F_1(\mathbf{x})$ dominates $F_2(\mathbf{x})$. Dominated solutions are inefficient, since they are strictly worse than the solutions by which they are dominated. The set containing all non-dominated solutions, is called the Pareto Frontier. An example on how such a Pareto Frontier could look like, is given in figure 2.1, as the red line. The Pareto Frontier is approximated by the available non-dominated solutions, marked as blue dots.

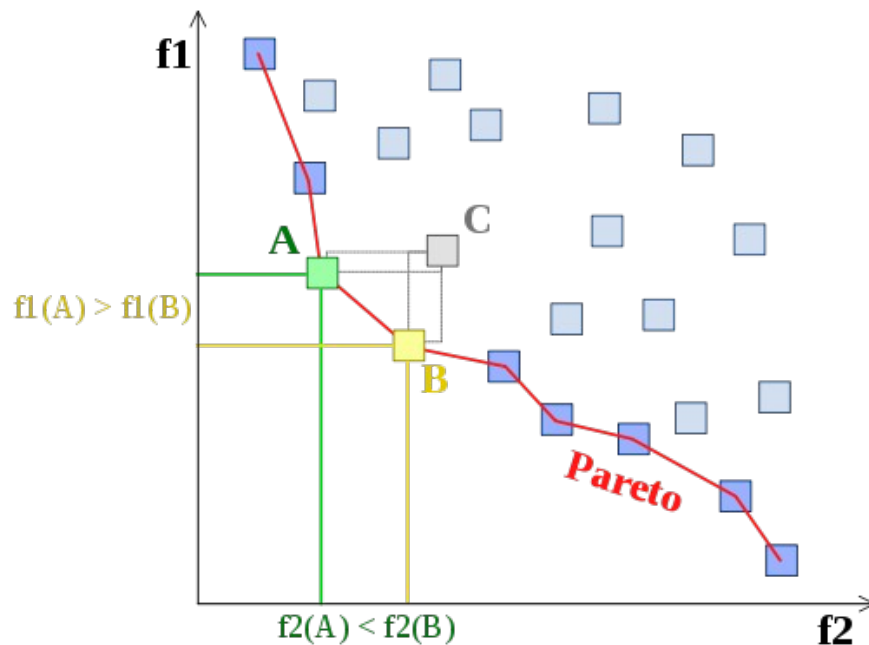


Figure 2.1: Visualization of a Pareto Frontier. The lines show the range of domination. From Wikipedia on Pareto-efficiency [28]

The roughest distinction one could possibly make for numerical optimization methods, is between deterministic and heuristic/stochastic algorithms. The first class of algorithms relies

on traditional ways, mainly calculus (derivatives and gradients), to find the optimum. The behaviour is deterministic, i.e. the algorithm will always behave the same way, given the same situation in an iteration. Their decision on how to modify the feature vector is usually based on either the finite differences between multiple function evaluations, derivatives of the function, or a combination of the mentioned.

On the other hand, algorithms belonging to the latter class use a type of randomness, usually tolerating minor deteriorations in the quality of the function, in order to explore the search space. Characterizing the multitude of existing algorithms is not a straightforward task, since the algorithms usually employ multiple aspects. Algorithms of both classes will be discussed in section 2.1.

While the fundamental ideas behind the algorithms are usually easily described, the algorithms itself introduce their parameters as new subjects to study and optimization. Heuristic algorithms are especially sensitive to these parameters.

This way, the optimization is a recurring problem. In section 3, many works will be presented which deal with searching optimal parameters for problems of certain problem domains. These will greatly vary between the domains, but we will also see that optimal parameters shift even if we change the problem within domain, or only its characteristics.

Heuristic algorithms usually have a robust method to explore the feature space, but at the cost of additionally needed function evaluations. This can pose problems for time-critical, online optimizations, or computationally expensive function evaluations.

If we consider a time budget, which limits the given time and in further consequence the number of function evaluations, we may prioritize earlier convergence into a local minimum to slower, possible convergence towards the global minimum outside the time budget.

Since the time to converge is highly dependent on the parameters, they should be chosen according to the given time budget. Section 3 will therefore provide references to works, where the time budget is taken into consideration when choosing parameters and algorithms.

2.1 Algorithms

This chapter contains a short description of algorithms which are used throughout this thesis, or well-known optimization algorithms, on which new algorithms are usually based. They serve as a first foundation of better understanding Numerical Optimization.

2.1.1 Downhill-simplex

Downhill-simplex, or Nelder-Mead method [3] uses a general simplex with $n+1$ vertices, where n is the number of dimensions of the problem.

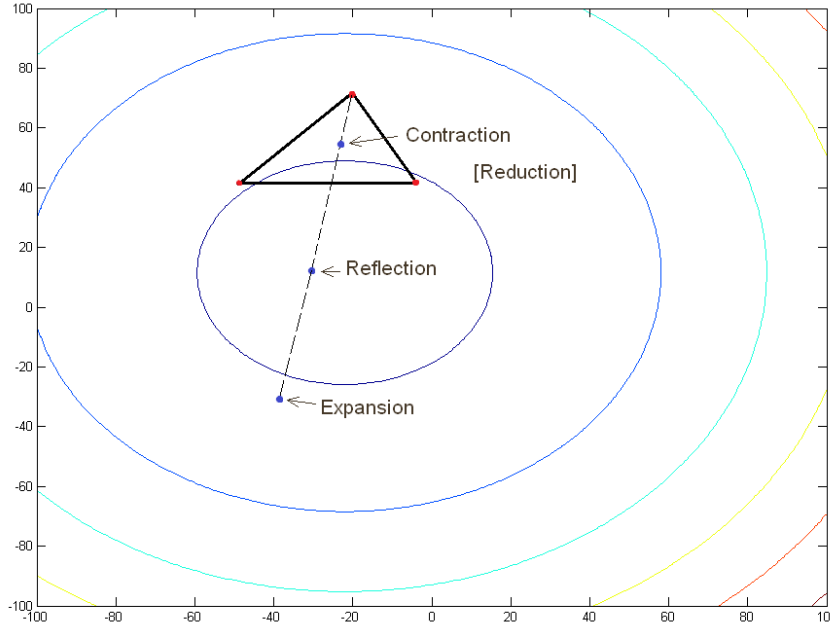


Figure 2.2: Nelder Mead Optimization on the Example of a Half-Sphere

For the two-dimensional case, figure 2.2 illustrates a simple example how the optimization works: The simplex is defined by the three red points. The upper point is the worst point, and thus we calculate the centroid of the two better points, which lies in the middle of the line between the lower two points. The worst point will be reflected, expanded and contracted along the centroid of the two better points. If all of these new points, marked in blue, offer no improvement, all current points of the simplex are moved towards the best point (reduction).

2.1.2 Simulated Annealing (SA)

Simulated Annealing is a thermodynamically inspired algorithm. The algorithm was described in [4] as follows: From a point \mathbf{x} a new point \mathbf{y} is randomly created. The new point is immediately accepted if $f(\mathbf{y}) \leq f(\mathbf{x})$. Otherwise, it is accepted with the probability according to the Boltzmann distribution, given in equation (2.4). The parameter T is called the temperature and is a decreasing series, according to a cooling-down schedule.

$$p = \exp\left(\frac{f(\mathbf{x}) - f(\mathbf{y})}{T}\right) \quad (2.4)$$

2.1.3 Genetic Algorithms (GA) & Evolutionary Algorithms (EA)

Differentiating between GA and EA is not easy and the terms are used interchangeably in the literature. As Mitchell already said it in [5]:

“It turns out that there is no rigorous definition of "genetic algorithm" accepted by all in the evolutionary–computation community that differentiates GAs from other evolutionary computation methods. However, it can be said that most methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring”

The same usually holds for evolutionary algorithms, but evolutionary algorithms usually code the problem as real number problems, while genetic algorithms try to find bit string encodings of the problems.

2.1.4 Particle Swarm Optimization (PSO)

PSO belongs to the class of so-called Swarm Intelligence algorithms. What differentiates them from EAs is that the created population in the beginning is used throughout the whole optimization process. In PSO, we give every member of the population (particle) a starting velocity, and calculate the velocity of the next iteration according to equation (2.5), presented in [6]. This equation shows, how a particle is drawn towards the best positions, while being restrained from sudden velocity changes by inertia. The factors w, φ_p, φ_g are inertia, cognitive and social factors respectively, r_1 and r_2 are uniformly distributed random numbers drawn from the interval $[0, 1]$ and $\mathbf{p}_{\text{best}}, \mathbf{g}_{\text{best}}$ are the personal and global best known positions respectively.

$$\mathbf{v}' = w \cdot \mathbf{v} + \varphi_p r_1 \cdot (\mathbf{p}_{\text{best}} - \mathbf{x}) + \varphi_g r_2 \cdot (\mathbf{g}_{\text{best}} - \mathbf{x}) \quad (2.5)$$

2.1.5 Artificial Bee Colony

This is another algorithm, inspired by nature, belonging to the class of Swarm Intelligence algorithms. The hive consists of three different types of bees: Scout, onlooker and employed bees. Scouts look for new food sources, which they report back to the waiting onlookers, which will in turn become employed bees and gather that food source. In optimization, the position of a food source is a parameter setting, while the quality of the food source corresponds to the function evaluation at that point. Employed bees now make minor modifications to their position to test the surrounding area of their current food source. If the modification offered an improvement, the employed bee will report the better position back to the onlooker bees in the hive. The scout population is maintained throughout the whole procedure to explore the search space. Since onlookers have many food sources to choose from, they randomly choose one of the presented positions, where the probability is

proportional to the quality. The parameters for this algorithm are simple: Population size, number of onlookers, number of scouts and number of employed bees.

2.1.6 Nondominated Sorting Genetic Algorithm II (NSGA-II)

This is one of the most popular multi objective optimization algorithms, which tries to find the Pareto frontier. First shown in [7], it is a speed- and performance-wise improvement over the first NSGA algorithm. It uses the characteristics of an EA algorithm, but changes the selection: All non-dominated solutions are found, and get rank i , after which they are removed and the procedure starts over for rank $i+1$. Within the ranks, a crowding metric is used, to distinguish between solutions in crowded areas and non-crowded areas. Solutions in non-crowded areas are preferred during selection, to better explore and find the whole Pareto frontier.

3 Related Work

There is a vast range of topics in engineering and satellite communications, which use numerical optimization to solve difficult problems. In [8], the algorithms Simulated Annealing, Genetic Algorithm, Downhill-simplex and Particle Swarm Optimization were compared in their ability to reconstruct the far-field radiation pattern of antennas, from near-field measurements. Multiple variations of PSO were also tested: One can decide, if the best fitness value is immediately shared, or only after all individuals have all evolved one iteration (asynchronous and synchronous respectively), and if all individuals share the global best fitness value, or only within clusters of particles (global and local). The comparison showed that all algorithms were capable of mostly reconstructing the patterns, like can be seen in figure 3.1 but PSO with global asynchronous updates exceeded the performance of all other mentioned algorithms by converging faster.

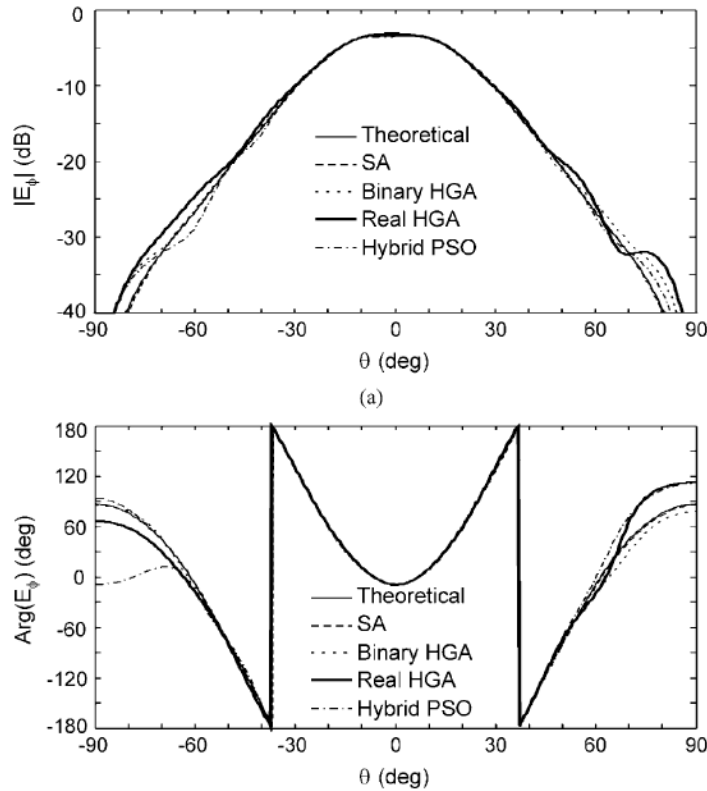


Figure 3.1: Reconstructed Far Field Radiation Pattern for the Pyramidal Horn. (a) Amplitude. (b) Phase, from [8]

PSO can also be used to solve combinatorial problems, like resource scheduling, demonstrated in [9]. The authors combined the genetic operators selection and crossover, with PSO in order to achieve a resource scheduling optimization of a satellite system with constraints. The

system was restricted by few minor constraints and the available power and memory. It consisted of six payloads p_1, \dots, p_6 , each of which can be turned on, or off (represented by 0 or 1). So the position \mathbf{x} of a particle is defined as $\mathbf{x}=[p_1, \dots, p_6]$. PSO was able to find an optimal schedule, respecting the constraints.

Since there is such a wide range of algorithms, there is the need to compare all of them through metrics and challenges in order to rank them and identify 'superior' algorithms to 'inferior' ones. However, when talking about ranking algorithms, there is a fundamental theory developed by Wolpert and Macready, which we always have to keep in mind when trying to compare optimization algorithms. In their paper [10], they developed a number of theorems, for which they concluded that all optimization algorithms are equally good, if we would consider all existing real number problems, and give them the same a-priori probability, or in their words:

“[...]for any algorithm, any elevated performance over one class of problems is offset by performance over another class.”

This leaves us with the conclusion that a general algorithm, which is better than all its competitors for every problem, cannot exist. As demotivating as this sounds, if we take into account that on one hand, we do not need to solve every possible problem, and on the other hand that we are fine with average performance in the worst case, the problem is less prevalent.

One way to achieve a good benchmark of algorithms, is to define a set of problems and functions, which employ very different aspects and compare the final minima obtained in multiple runs. A popular example is the annual CEC congress on evolutionary computing, which regularly releases challenges on different optimization topics [2005 and 2008]. The exact definitions, problems and graphs for the 2D problems in the year 2005 on real valued problems can be seen in [11]. The dimensionality of the problems can be adapted to much higher dimensional cases, to more realistically reflect practical problems.

Numerical Optimization algorithms need some parameters to be set. Optimal parameter settings have been subject to study for many investigations. Works like [12] try to provide a good overview for the operator, which algorithms and parameters to choose depending on the problem at hand, but this approach requires an operator with enough knowledge of the problem domain. Alternatively adaptive algorithms exist, like the one presented in [13], which uses differential evolution and automatically adapts the parameters to the given situation for a good overall performance. A more traditional way is to take a few problem examples from the problem domain and fine-tune the parameters to it. One example for this way is shown in [14], where the symbols of an APSK modulation were optimized on the polar coordinate system, to decrease the symbol error rate for a low Signal-to-Noise-Ratio. The different test scenarios were: 16-APSK and 32-APSK with no, single, or double symmetry for both cases.

Multiple crossover and selection parameters of the genetic algorithms were combined, which resulted in sixty different variations, where all of them were tried. The optimal crossover and selection heuristics, which lead to the best fitness, or converged faster, changed between nearly every modulation scenario. The margin was not exceedingly large, but this example demonstrates that even small changes within the problem domain can change the optimal parameters for the optimization methods.

3.1 Parameters in Optimization

When looking at the presented related work in practice so far, we can see that optimization can effectively deal with complex problems, but introduces another hurdle at the same time: The parameters necessary for the algorithms. Their number and significance greatly vary from algorithm to algorithm, but a wrong set of parameters can undermine the algorithms efficiency, or even completely shut it down. This section presents a few papers, where the authors tried to overcome this problem in different ways.

3.1.1 Algorithm Portfolios

Algorithm Portfolios try to deal with this problem, by combining multiple algorithms, or a single algorithm with multiple parameter settings, into one single portfolio.

One could argue that by composing the portfolio, we introduce another, even more complex parameter, namely all algorithms in the portfolio and their parameters, but an automated approach to solve this dilemma will be presented later in this section.

The hard part is now the decision, when to choose which algorithm for the next iteration, since the number of function evaluations is very limited. In [15], Yuen et al. propose a method for choosing the best algorithm from a set of algorithms, through predicting their future fitness. The computational time is then invested in the best predicted algorithm, since it most likely leads to a better fitness. The prediction and selection is done every iteration. In figure 3.2, we can see an exemplary cost curve, where the x-axis represents the number of function evaluations, while the y-axis shows the best fitness until that point.

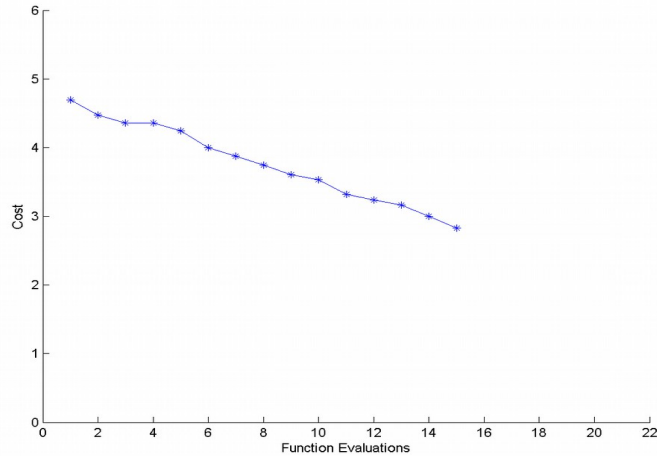


Figure 3.2: Cost Function Evolution over the Number of Function Evaluations

They now want to predict the fitness of each algorithm, at the nearest common point in future, which is defined as the longest number of times an algorithm has run, plus one. They define the set $C_i = \{c_1, \dots, c_i\}$, where c_i is the cost after i function evaluations. We now create a line using linear regression for each of these sets C_2, \dots, C_n , for each algorithm. As visualized in figure 3.3, we now intersect each line with the nearest common point in future. Using these fitnesses, we create a distribution from which we sample a predicted fitness.

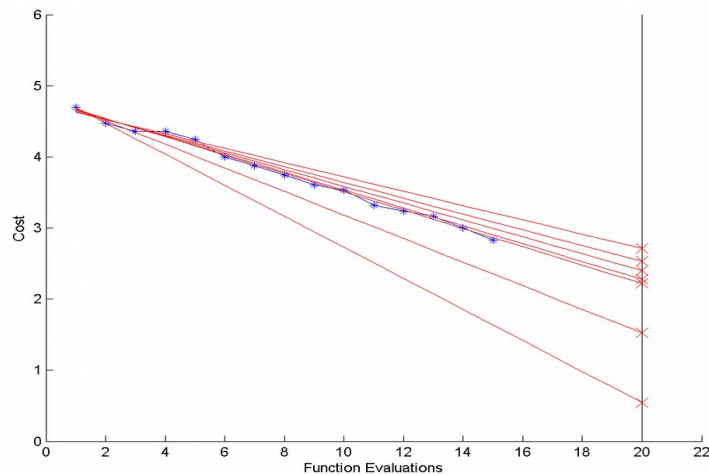


Figure 3.3: Cost Evolution Function, with Lines created through Linear Regression

In their paper, creating the distribution with the Matlab function `'ksdensity'` offered the best prediction, but they were able to produce nearly similar performance using a histogram based

distribution. The distribution generated from the points of figure 3.3, can be seen in figure 3.4. The red crosses mark the points of intersection.

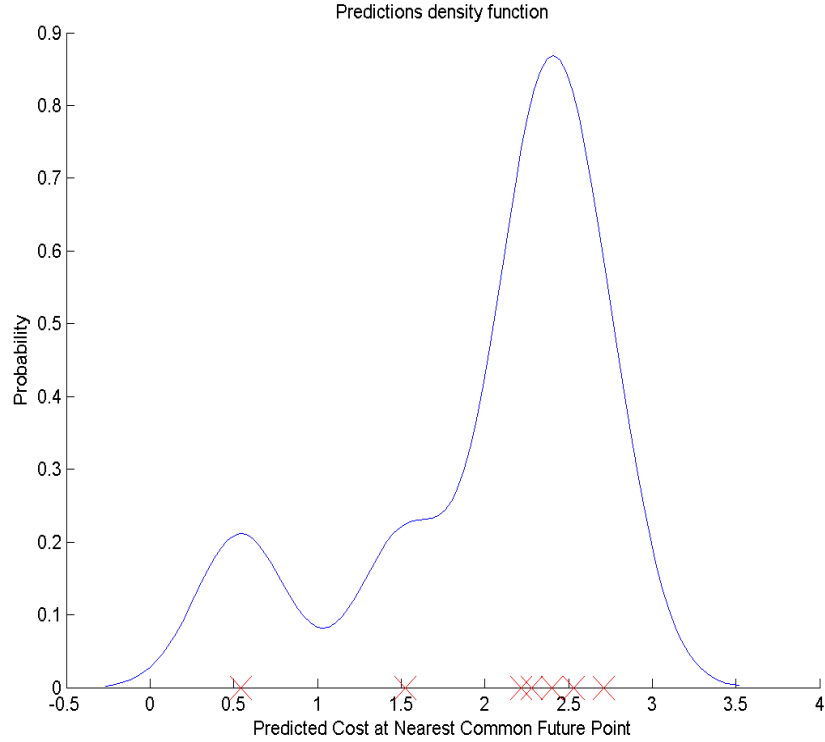


Figure 3.4: Created Distribution through the Function 'ksdensity'

At the start, we need to run each algorithm a few times until the fitness is decreasing, or otherwise the algorithm would never be chosen. An advantage of this prediction is that it does not need any control parameters.

Another question which still remains, is how to compose the portfolio. Which algorithms, and how many should we choose? We want a diverse set, applicable on a broad range of problems, but keeping the portfolio under a certain size, to reduce the overhead cost. In [16], an algorithm is introduced which (almost) automates the composition of such portfolios. We define a set of algorithms, from which the portfolio can be composed. All algorithms are used on a set of benchmark functions (they used the CEC 2005 functions [11]) and they are ranked according to their performance on each function. We now build a covariance-matrix between the rank of all algorithms and compose a portfolio of the two algorithms with the lowest covariance. This portfolio acts as a new algorithm for the next iteration of the algorithm. We stop, if the newly composed portfolio offers no improvement over the old one. The presented covariance matrix from [16] can be seen in table 3.1.

Algorithm	ABC	CMA-ES	CoDE	PSO	SaDE
ABC	1.96	-1.20	-0.49	-0.49	0.22
CMA-ES	-1.20	3.21	-0.70	-0.37	-0.94
CoDE	-0.49	-0.70	1.34	-0.32	0.17
PSO	-0.49	-0.37	-0.32	1.34	-0.17
SaDE	0.22	-0.94	0.17	-0.17	0.72

Table 3.1: Covariance Matrix of 5 Algorithms on a Test Set of 10 Functions, from [16]

Since we combine algorithms with the lowest covariance, we ensure that the portfolio offers a good performance on a diverse set of functions and the stopping criterion prevents us from bloating the portfolio with unnecessary algorithms which offer no gain in performance.

One such example of an algorithm portfolio is also shown in [17], where four multi-objective algorithms are composed into a portfolio to surpass the performance of one. Additionally, individuals of all populations migrate between the algorithms and less successful algorithms also create less offspring. This migration technique is also a common form of using multiple optimization algorithms, but all algorithms are usually from the same type, since it is easier to migrate their parameters.

3.1.2 Self Adaptation

Another approach is, to use the same algorithm, but vary the parameters accordingly to the current state of the search. One such example can be found in [13], where a self adaptive algorithm is developed, using Differential Evolution at the core. Differential Evolution is an algorithm presented in [18], by Storn and Price, which uses mutation, crossover and selection operations for a population of real number problems.

The problem with Differential Evolution is that the performance is very sensitive to parameters like mutation and crossover rate. In [13], Qin et al. regard the parameter setting for Differential Evolution, but their conclusion can basically be applied to almost any parameter-sensitive optimization algorithm:

“In DE literatures, various conflicting conclusions have been drawn with regard to the rules for manually choosing the strategy and control parameters, which undesirably confuse scientist and engineers who are about to utilize DE to solve scientific and engineering problems. In fact, most of these conclusions lack sufficient justifications as their validity is possibly restricted to the problems, strategies, and parameter values considered in the investigations.”

The self adaptive algorithm (named SaDE by Qin et al.), tries to solve this problem, by adaptively choosing these parameters. In their paper, they randomly draw the sensitive

parameters several times within a certain interval. The algorithm is now randomly run with one of the drawn parameters, where the probability to choose the parameter is dependent on the success, or failure rate of it so far. One such evolution of the probabilities can be seen in figure 3.5, where the Griewank function of the CEC was used as a benchmark.

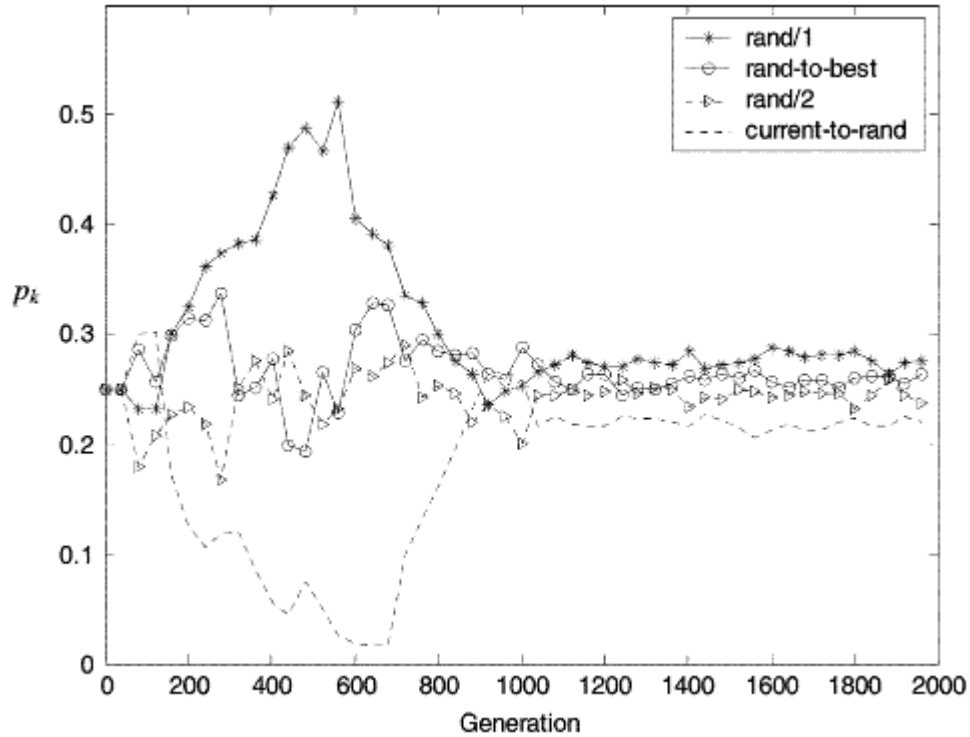


Figure 3.5: Evolution of the Probabilites in the SaDE algorithm. From [13]

In contrast to the portfolios, the current population is kept, even when changing the parameters.

3.1.3 Meta Optimization

Both portfolios and self adaptation aim at the same target, with similar means: Some parameters are varied through measuring and analyzing their success rate on the fitness function. If we look at it in a simplified way, we use one optimizer to configure another optimizer. This procedure is commonly known as meta optimization, or hyper heuristics. One possible way to find the optimal parameters, is to try different variations on a benchmark set of problems. Many works like [19] and [20] show that the optimal parameter setting for optimization is crucial and that the overlying optimizer for the parameters is more robust against wrong parameter settings. Advanced approaches like SaDE and portfolios are heavily researched since they can better deal with problems, not similar to the benchmark problems

on which the parameters were tuned on.

4 Design

This section will present the planned design and consider possible problems which could occur. Section 4.1 offers a short introduction into the currently available simulator. In section 4.2, we will show the preliminary design of the optimizer and continue with its practical application for the link budget calculation in section 4.3. Lastly, section 4.5 contains information on the planned test bed for testing and evaluating future optimization algorithms.

4.1 Simulator

The Simulator is composed of modules, where the data to process are passed from the top module, through all modules until the bottom module. There is no restriction on the ordering and processing of modules to offer the maximum amount of flexibility. In listing 4.1, an example simulation file is shown. The indented lines below a module are parameters from the user to configure the modules, where 'x1' marks a parameter which is varied in a fixed way: In this example, the Signal-to-Noise-Ratio (SNR) for the Additive White Gaussian Noise (AWGN) module is increased from 5dB to 20dB, in 1 dB increments for each simulation.

In listing 4.2, the result file is shown for the above simulation. The first column shows the varied x1 parameter, namely the SNR, while the second column shows the calculated corresponding Symbol Error Rate (SER). Obviously, the SER decreases, as the SNR increases. The used modulation is QPSK.

```

MODULE SimulationControl
    burstLen          100
    loopsPerPoint     100000
    codeRate          1 //mapping 1=1;12=1/2;23=2/3;34=3/4;45=4/5;78=7/8;89=8/9
    modulation        2 //1=BPSK; 2=QPSK; 3=8-PSK; 4=16-APSK; 5=16-PSK; 6=32-APSK

//Sender-----
MODULE DataGeneratorBytes
    mode              0 //0:random; 1:zero burst; 2:dirac

MODULE ByteToBit

TESTPOINT            3

MODULE MapperBitToSymbol

//Channel-----
MODULE AWGN
x1 snr                5 20 1 //begin end stepsize QPSK

//Receiver-----
MODULE MapperSymbolToHardBit

TESTPOINT            4

MODULE BitToByte

//TESTPOINT          2

// Analyzer-----
MODULE Analyzer_SER
    tp1               3
    tp2               4
//End of Simulation file ***

```

Listing 4.1: Simple Simulation File Example

snr	SER
5	0.0735048
6	0.0452853
7	0.024838
8	0.0118664
9	0.004765
10	0.00155138
11	0.0003795
12	6.86e-005
13	7.875e-006
14	4.25e-007
15	2.5e-008
16	0
17	0
18	0
19	0
20	0

Listing 4.2: Result File of the Simulation shown in Listing 4.1

Naturally, there is the need to leave the general structure of the simulation file untouched, while providing easy means to mark parameters for the optimizer. Since a recompilation is undesirable when changing the optimization configuration, the optimization parameters are marked in the simulation file. These so-called 'tags' retain the relatively simple, original structure of x1 and x2. According to this syntax, the tags were implemented as follows:

Variables:

Parameters that can be varied by the optimizer in order to achieve the best possible performance.

v [Parameter name] [min] [max] [step size]

Constraints:

If a parameter is marked as a constraint, the optimizer will look at its value at the end of each evaluation. The parameter's value needs to be smaller, higher, or exactly equal to the given value, or the solution will receive the worst possible performance, which is positive or negative infinity, depending if the objective needs to be maximized or minimized.

c< [Parameter name] [initial value] [max]

c> [Parameter name] [initial value] [min]

c= [Parameter name] [initial value] [required value]

Objective(s):

The objectives report the performance of the current simulation setting. The optimizer will look at the value obtained and adapt its strategy to improve the performance. o+ maximizes the parameter, while o- minimizes it. The employed underlying technique, to achieve both maximization and minimization applying only minimization algorithms, will be similar to equation (2.2).

o+ [Parameter name] [initial value]

o- [Parameter name] [initial value]

Each spacing can consist of an arbitrary collection of whitespace characters.

In the beginning, only one parameter may be marked as an objective. If time allows it, later releases will switch to multi objective optimization algorithms if multiple parameters are marked as an objective. For multi objective optimization, a suitable algorithm for finding the Pareto frontier, which has already been discussed in section 2, will be applied.

```

//Atmosphere Models: 0: V-Band, location Graz, 1: Q-Band, location Graz,
//                  2: V-Band, location Tito, 3: Q-Band, location Tito
//                  ...
MODULE LinkBudget
  //Uplink
  tx_antenna_diameter      3          //[m]
  tx_antenna_gain          60.84      //[dBi]
  tx_antenna_feed_loss     0          //[dB]
  tx_distance_to_satellite 38175      //[km]
  tx_tracking_depoining_loss 0.3      //[dB]
  tx_power                  200       //[W]
  v ul_frequency            46.5 50 0.1 //[GHz]
  ul_bandwidth              50        //[MHz]
  v ul_atmosphere_model     0 3 1
  //Satellite
  sat_g_over_t              4.4       //[dB/K]
  sat_max_eirp              39        //[dBW]
  sat_obo                   1.1       //[dB]
  //Downlink
  rx_antenna_diameter      3          //[m]
  rx_antenna_gain          59.2       //[dBi]
  rx_antenna_feed_loss     0.96       //[dB]
  rx_distance_to_satellite 38175      //[km]
  rx_tracking_depoining_loss 0.2       //[dB]
  rx_lna_transmission_loss 0.52       //[dB]
  rx_lna_noise_temperature 205        //[K]
  rx_mean_sky_temperature  270        //[K]
  rx_ground_temperature    30         //[K]
  dl_frequency              37.85     //[GHz]
  dl_bandwidth              50        //[MHz]
  dl_atmosphere_model      1
  o+ mean_received_power    -120      //[dBW]
  c< tx_antenna_efficiency  60        //[%]
END

```

Listing 4.3: Simulation File with Optimization Tags

A simulation file with a few of these tags added, for the link budget example, which will be introduced in section 4.3, can be seen in listing 4.3. The tagged parameters are highlighted in a red rectangle.

4.2 Optimizer

There are two main requirements for developing the optimizer: On the one hand, its integration into the simulator has to be as easy as possible for the end-user and secondly, it needs to be extensible in the sense that new optimization algorithms can quickly be integrated.

The first requirement will mainly be ensured by an easy extension for the simulation files, which was already presented in section 4.1, but also by making the optimizer a module itself.

By making the optimizer a module, we can set its parameters like any other module during start-up and without recompilation by modifying the simulation file. Since the parameters of the algorithm heavily depend on the number of variables and other function specific properties, parameterless algorithms are preferred over very parameter-dependent algorithms.

In figure 4.1, the planned class hierarchy is shown. The base classes should provide enough of the main functionality for optimization (bound values to step size, translate between feature points and module parameters,...) to keep the actual algorithms as short as possible. Additionally, we can benefit from this structure by decoupling the simulator-specific module interfaces, from the actual optimization.

In a first step, a stand-alone benchmark is planned to ensure functionality and quality of the algorithms, before integrating them into the simulator. The planned benchmark is described in more detail in section 4.5.

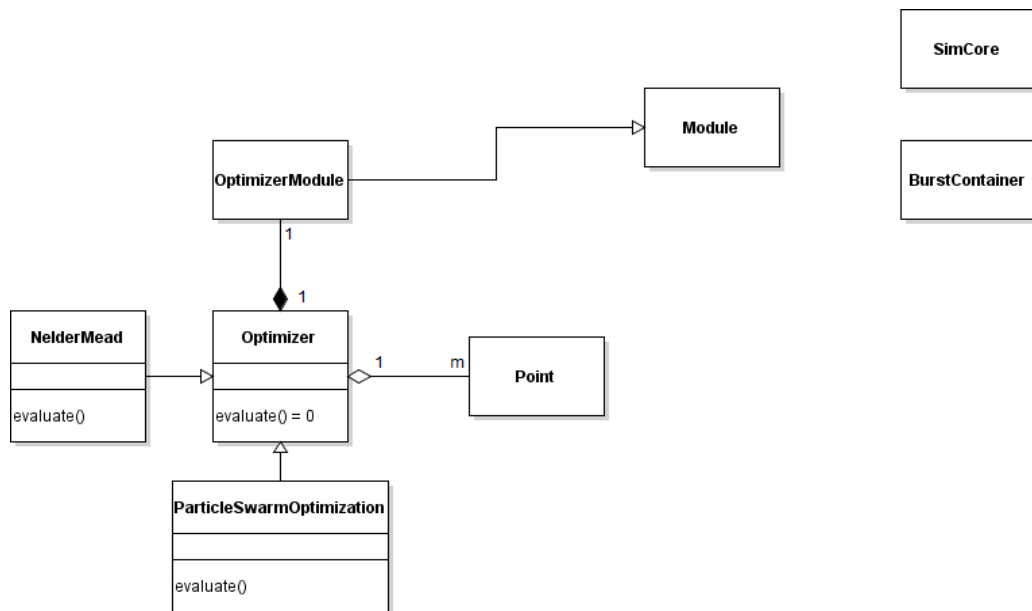


Figure 4.1: Preliminary Class Diagram of the additional Optimization Modules

4.3 Link Budget

A satellite system, when regarded in a very simplified way, can be depicted as shown in figure 4.2:

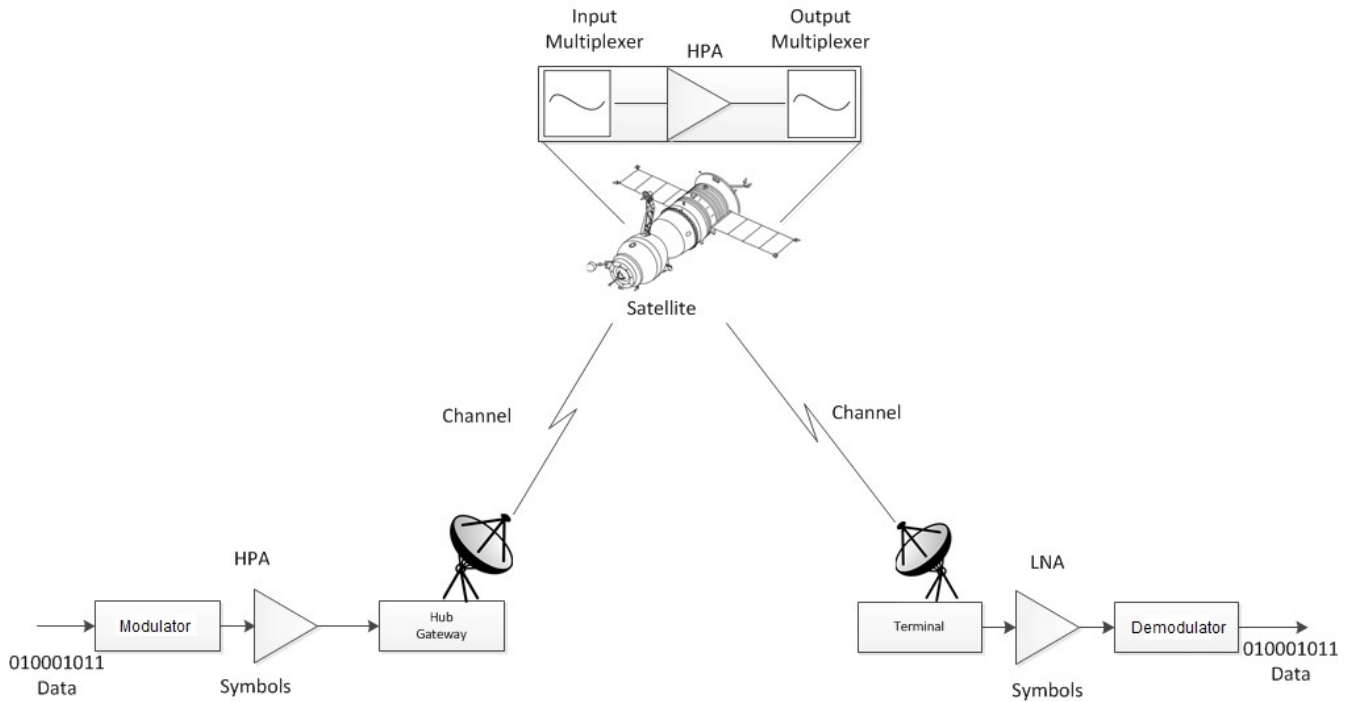


Figure 4.2: Simplified View of a Satellite System

We have a data stream, which is modulated on the carrier signal, amplified by the HPA and transmitted over the gateway towards the satellite, usually termed as 'Uplink'. In this figure, the satellite is considered to be a transparent repeater, which simply receives the signals, amplifies them and sends them to a terminal, which might be the transmitting gateway again (Loopback mode). In contrast, we call the second part of the transmission 'Downlink'.

Since naturally not all satellite systems are similar with respect to their performance, we have to firstly define a few parameters, constants and quality criteria, which influence the quality of the links:

General:

Wavelength: λ [m]

Distance to satellite: R [m]

Free Space Loss:

$$L_{FS} = \left(\frac{4\pi R}{\lambda} \right)^2 \quad (4.1)$$

Boltzmann's constant: $k \left[\frac{J}{K} \right]$

Aperture efficiency: ϵ

Antenna diameter: d [m]

Antenna gain:

$$G = \epsilon \left(\frac{\pi \cdot d}{\lambda} \right)^2 \quad (4.2)$$

Uplink:

Transmit power: P_T [W]

Transmit antenna gain: G_T

Antenna feed losses: L_{fr}

Antenna depointing loss: L_d

G/T of satellite: $G/T_{SAT} \left[\frac{1}{K} \right]$

EIRP towards satellite:

$$EIRP_{GS} = \frac{P_T G_T}{L_d L_{fr}} \text{ [W]} \quad (4.3)$$

Power flux density at satellite without atmosphere:

$$\Phi_{Sat} = \frac{EIRP_{GS}}{4 \pi R^2} \left[\frac{W}{m^2} \right] \quad (4.4)$$

Atmospheric attenuation: L_A

Uplink C/No:

$$\left(\frac{C}{N_0} \right)_U = \frac{EIRP_{GS} \cdot G/T_{SAT}}{k \cdot L_{FS} \cdot L_A} \text{ [Hz]} \quad (4.5)$$

Effective transponder noise bandwidth: B_{SAT}

Uplink C/N:

$$\left(\frac{C}{N} \right)_U = \left(\frac{C}{N_0} \right)_U / B_{SAT} \quad (4.6)$$

Power robbing:

$$L_{rob} = \frac{C+N}{C} = 1 + 1/\left(\frac{C}{N}\right) \quad (4.7)$$

Downlink:

Receive antenna gain: G_R

Antenna feed losses: L_{fr}

Antenna tracking loss: L_d

Maximum satellite EIRP: $EIRP_{SAT}$ [W]

Satellite transponder output backoff: OBO [W]

Power flux density at ground station without atmosphere:

$$\Phi_{GS} = \frac{EIRP_{Sat} - OBO}{4\pi R^2} \left[\frac{W}{m^2} \right] \quad (4.8)$$

Atmospheric attenuation: A_{SKY}

Mean sky temperature: T_m [K]

Sky noise temperature¹:

$$T_{SKY} = T_m(1 - A_{SKY}) + \frac{2.7}{A_{SKY}} \text{ [K]} \quad (4.9)$$

Ground noise collected by receiver antenna: T_{GR} [K]

Antenna noise temperature:

$$T_A = T_{SKY} + T_{GR} \text{ [K]} \quad (4.10)$$

LNA noise temperature: T_{LNA} [K]

Downconverter noise temperature: T_{DC} [K]

Reference temperature: $T_0 = 290\text{K}$

Receiver temperature²:

1 2.7K is cosmic background noise

2 The gain of the LNA is typically so large, that the noise contribution of the down converter is negligible.

$$T_R = T_{LNA} + \frac{T_{DC}}{G_{LNA}} \approx T_{LNA} \text{ [K]} \quad (4.11)$$

System noise temperature:

$$T_{SYS} = \frac{T_A}{L_{fr}} + T_0 \left(1 - \frac{1}{L_{fr}} \right) + T_R \text{ [K]} \quad (4.12)$$

G/T of ground station:

$$G/T = \frac{G_R}{T_{SYS}} \left[\frac{1}{K} \right] \quad (4.13)$$

Downlink C/No without power robbing:

$$\left(\frac{C}{N_0} \right)_{D'} = \frac{(EIRP_{SAT} - OBO) \cdot G/T}{k \cdot L_{FS} L_A L_d L_{fr}} \text{ [Hz]} \quad (4.14)$$

Downlink C/No with power robbing:

$$\left(\frac{C}{N_0} \right)_D = \left(\frac{C}{N_0} \right)_{D'} / L_{rob} \text{ [Hz]} \quad (4.15)$$

Total C/No:

$$\left(\frac{C}{N_0} \right)_{TOT} = \frac{1}{1 / \left(\frac{C}{N_0} \right)_U + 1 / \left(\frac{C}{N_0} \right)_D} \text{ [Hz]} \quad (4.16)$$

Downlink C/N

$$\left(\frac{C}{N} \right)_D = \left(\frac{C}{N_0} \right)_D / B \quad (4.17)$$

$$SNR = \left(\frac{C}{N_0} \right)_{TOT} / B \quad (4.18)$$

Especially equation (4.18) will play a recurring role in this thesis as a general quality measure of the link. All of the above equations were taken from [21], where they are thoroughly explained.

Obviously, as one can see from equations (4.1) through (4.17), there is a multitude of factors which influence either the uplink, or downlink, or even both. Many of those include decisions on the location and hardware to choose when designing such a system.

In order to link these physical calculations into actual data throughput, we need to add the modulation and coding schemes, usually referred to as Modcod. The modulation tells us how the carrier is altered in order to transmit the symbol, but for this thesis, the order of the modulation provides enough information: How many bits are transmitted per symbol. The code rate is usually included in the name. In equation (4.19), we can see how we calculate the bits per symbol transmitted, where k denotes the effectively transmitted bits, while n is the total bits:

$$\eta = \frac{\underbrace{k}_{\text{Coderate}}}{\underbrace{n}_{\text{Modulation Order}}} \log_2(\# \text{Symbols}) \quad (4.19)$$

This thesis will mostly use PSK and APSK modulations with different code rates. Figure 4.3 shows an example from [22] for the placement of the symbols for an exemplary 16APSK modulation. In section 6.2, all used Modcods and their efficiencies will be introduced.

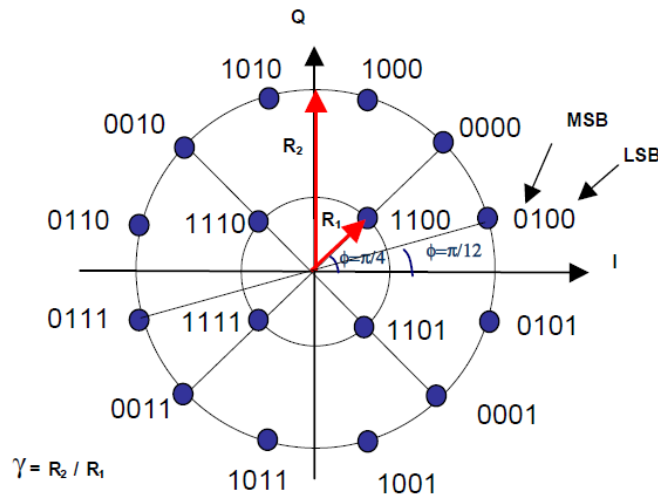


Figure 4.3: 4+12APSK Modulation Symbol Positions from [27]

The overall quality of the up- and downlink is usually described by Total C/No, given in equation (4.16). Higher C/No values allow for better coding schemes which increases the total data throughput the system can achieve.

This already reveals a lot of parameters in the satellite system, which can be tuned: One could change the location of the ground station to change the weather effects on the signal, but also vary the frequency of the carrier. A more complex problem involves splitting up the total available bandwidth by multiple carriers, while keeping the total output power constant.

Each carrier could employ different modulation and coding, further increasing the complexity.

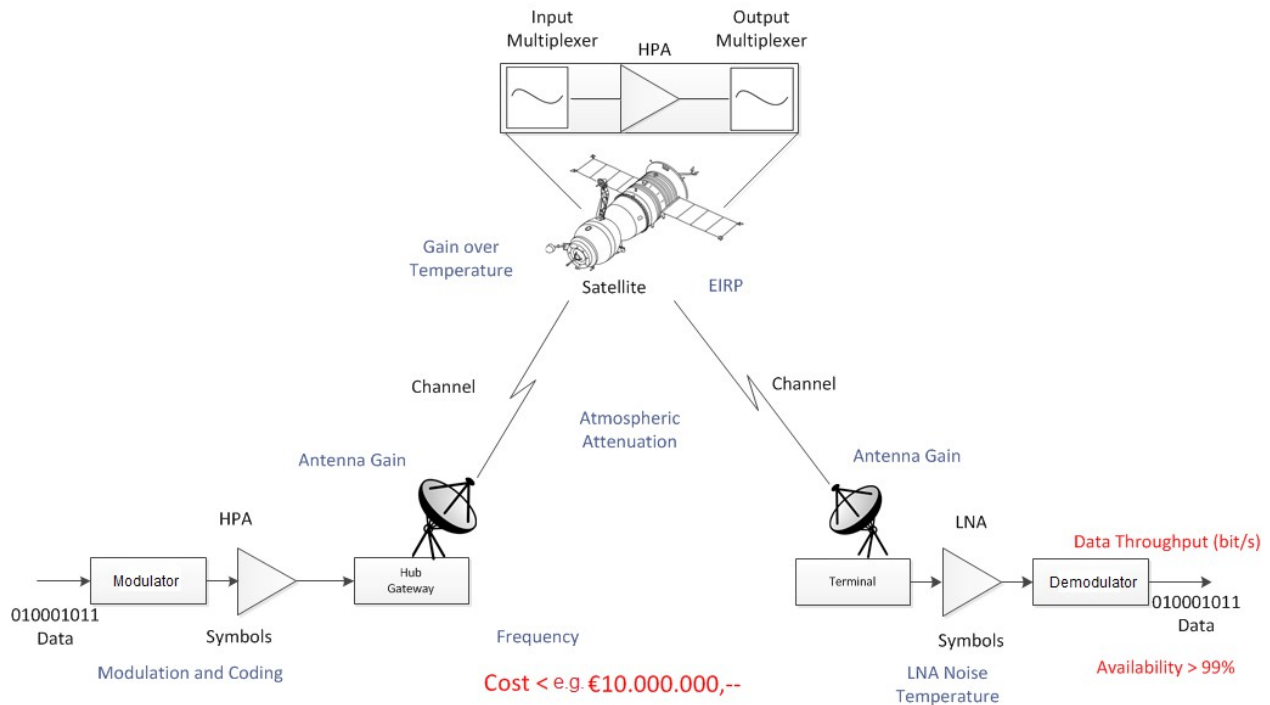


Figure 4.4: Simplified View of a Satellite System, with a few possible variables, constraints and an objective

Figure 4.4 again shows the previous satellite system, but this time also depicts a few of the multitude of parameters a designer could choose from, when designing such a system. They are depicted in light blue text. Additionally, two constraints (total cost and availability) and one objective, the data-throughput, are marked as red.

When designing satellite systems, the design could require a minimum bit-error-rate (objective), while keeping the costs below, or the link availability above a certain threshold (constraint). Although in [9], it was shown that Particle Swarm Optimization could handle constraints, dealing with multiple requirements through constraints is actually a sub-par way, since constraint-violating solutions are usually simply invalidated. However, since there is a hard decision boundary, the knowledge that we are close to violating a constraint cannot be used. A better way would be to use multi objective optimization algorithms, planned for later, but at first we will stick to the approach using constraints.

4.4 Multi-Beam Systems

The link budget, previously discussed in section 4.3 lays the theoretical foundation for further analysis of satellite system scenarios. It can provide sufficient information for a satellite

system with a single transmitting antenna, which covers the whole area which needs to be served. Nevertheless, newer satellite systems are equipped with multiple antennas, usually referred to as (spot) beams, where each beam covers only a part, or even only a fraction of the area to be covered.

According to [22], we can compute the linear gain a symmetrical-aperture antenna according to equations (4.20) and (4.21). The parameter D denotes the antenna diameter [m], ϵ is the aperture efficiency (typically 0.6), λ describes the wavelength [m], J_1 represents the first order Bessel function of the first kind, and θ denotes the off-axis angle measured in radian:

$$G = \epsilon \cdot \left(\frac{\pi D}{\lambda} \right)^2 \left[\frac{2 \cdot J_1(\mu)}{\mu} \right]^2 \quad (4.20)$$

$$\mu = \left(\frac{\pi D}{\lambda} \right) \sin(\theta) \quad (4.21)$$

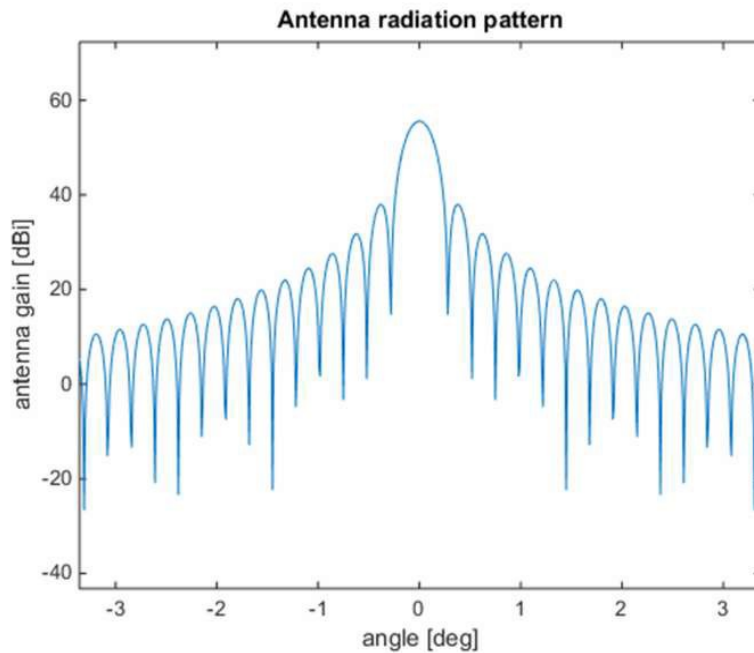


Figure 4.5: Exemplary Generated Radiation Pattern [dBi], Cartesian representation

In figure 4.5, such a generated radiation pattern on a logarithmic scale (dBi) can be seen for the specifications: $\epsilon=0.6$, $\lambda=0.015 m$, $D=3.7 m$, taken from [23].

How close the spot beams are placed to each other and therefore where their pattern overlap

is dependent on the so-called crossover-level. It is described by the difference in dB towards the maximum gain at zero degree. In figure 4.6, we can see the crossover angle for the radiation pattern in figure 4.5 for a crossover level of 3dB. It is approximately 0.12° for the above mentioned case.

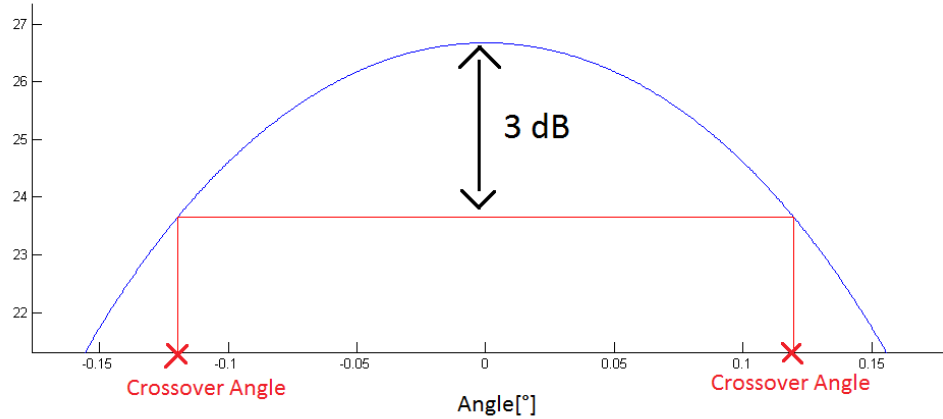


Figure 4.6: Closeup of Figure 4.5 on the Main Lobe with marked Crossover Angle of 3dB

We depict one spot beam as a circle, where the perimeter marks the given crossover-level, as can be seen in figure 4.7.

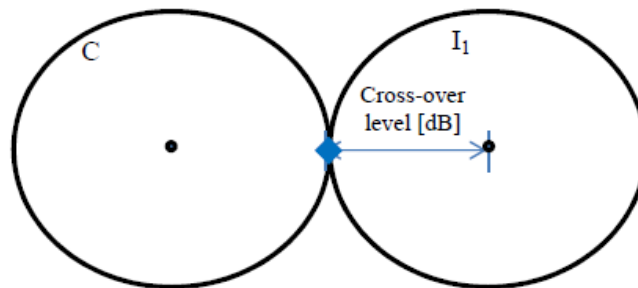


Figure 4.7: Visualization of a Spot Beam

In order to bring these definitions into physical dimensions, we have to define another few equations:

Knowing the angle α at the crossover-level, we can view the signal propagation as a cone. Thus, the diameter of the circle, covering the earth's surface can be calculated according to equation (4.22). We use the simple circle area formula, shown in equation (4.23) to calculate the area covered on the surface.

$$d=2 \cdot h \cdot \tan(\alpha/2) \quad (4.22)$$

$$A=\frac{\pi \cdot d^2}{4} \quad (4.23)$$

Combining equations (4.8) and (4.23), the received power on a given surface area can be calculated according to equation (4.24):

$$P_{Ground}=\Phi_{GS} \cdot A \quad [W] \quad (4.24)$$

Typical spot beam patterns employ placement like shown in figure 4.8. The central spot beam, marked as blue, is the user spot beam, where the diamond represents the user position. Since the spot beams are placed closely to each other in order to cover the whole needed area, they also influence each other. The ratio between the power of the signal of the main beam, transmitting data to the desired user, and the total interference power caused by the neighboring beams is called $\frac{C}{I}$ and simply expressed in terms of equation (4.25), where C is the power of the carrier and I is the total power of all interferers, both measured in dBW.

$$\frac{C}{I}=C-I \quad [dB] \quad (4.25)$$

Using the radiation pattern depicted in figure 4.5 with a crossover-level of 3dB on the spot beam pattern in figure 4.8, we receive the overlapping radiation patterns from figure 4.9, where the normalized power at the chosen user position is marked as a circle on each pattern.

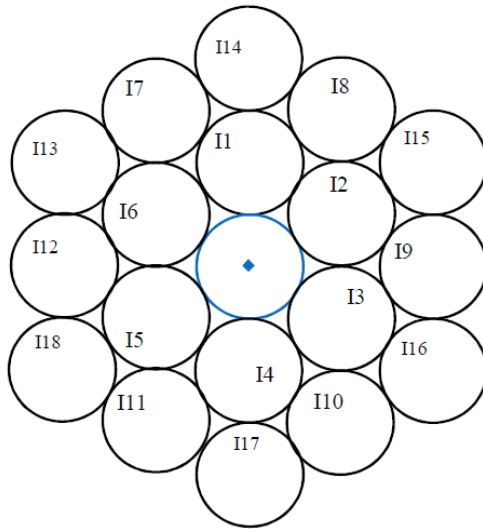


Figure 4.8: Hexagonal Spot Beam Pattern in use

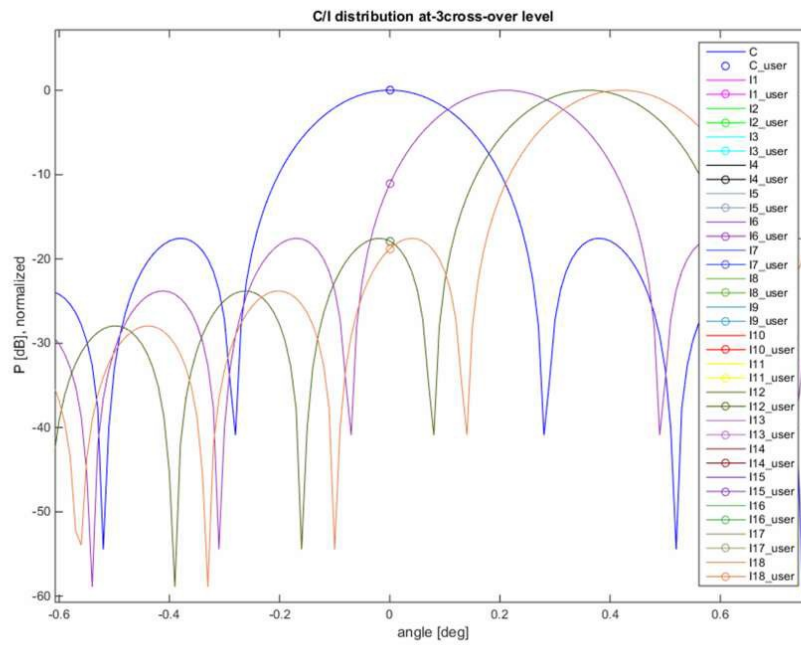


Figure 4.9: Normalized C and I values for the scenario shown in figure 4.8, from [23]

It is to be noted that not all radiation patterns of all spot beams are shown.

If we now wish to compute the desired total I value and subsequently $\frac{C}{I}$ from equation

(4.25), we transform the interference powers back into watts and sum them all up. This is expressed in equation (4.26), where I_i is the power of the i -th interferer in dBW, and n is the total number of interferers taken into account.

$$I_{Total} = 10 \cdot \log_{10} \left(\sum_{i=1}^n 10^{(I_i/10)} \right) \quad (4.26)$$

Since we still have to keep the C/N value in mind, we use the definition of the SNIR in order to express the total ratio between the noise and the signal. In equation (4.27) the exact calculation can be seen, which we will use to calculate the actual error rate and the modulation and coding scheme to use:

$$SNIR = -10 \cdot \log_{10} \left(10^{-\frac{C}{N}/10} + 10^{-\frac{C}{I}/10} \right) \quad (4.27)$$

We will try to use a modular approach, calculating the values for the link budget and the interferences in multiple modules. We will use existing structures to pass the information between the modules and separate the calculations to ensure the modularity of the system. The planned flow of information and modules can be seen in figure 4.10.

The two link budget modules at the top calculate the C/N₀, already presented in equation (4.15), which is one component of the calculation of the SNIR value. In the Beam Setup module, the beam grid should be constructed and the position of all beams needs to be determined. With this information the module Beam Interference will randomly decide a position for the user and calculate the total interference at this position, according to equation (4.26).

Continuing with this information, the SNIR module calculates the SNIR value and passes it to the Modem, which will adaptively or statically choose a Modcod.

The final module named Overall Performance will record average error rates, availability, throughputs and goodputs throughout one simulation.

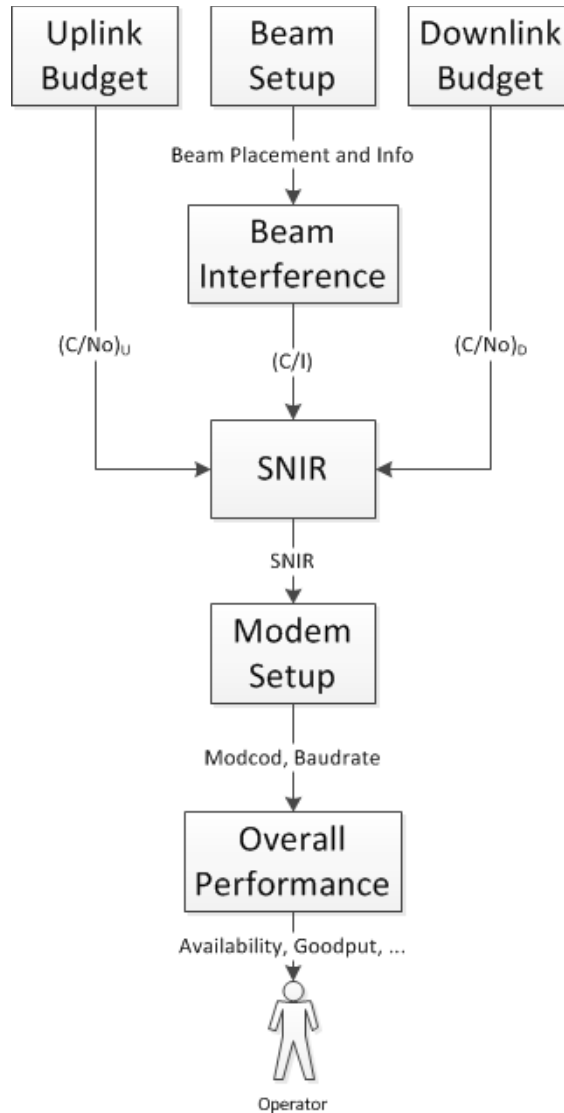


Figure 4.10: Planned Modules and Information Flow between them

4.5 Benchmark

Like already mentioned in section 3, when ranking optimization algorithms, we usually reduce the used procedure to empirical ranking on a set of complex problems. The planned benchmark class will provide the means to automate this procedure for optimization algorithms, targeted at the simulation software. The set of problems will be comprised of all functions, presented at the CEC 2013, which can be found in [24].

The test conditions are very similar to the conditions presented in the paper:

- All 28 problems will be run available in the dimensions: $D \in \{2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$
- $\mathbf{x} \in [-100, 100]^D$ where D is the number of dimensions
- Real-valued, meaning arbitrary step-sizes
- Maximum number of function evaluations = $D \cdot 10^5$
- Optimum found if $|f^* - f| < 10^{-8}$, where f^* is the global optimum
- 51 runs for each problem

The computational costs of the algorithms themselves are not recorded, since they are probably negligible in comparison to the computational costs of the simulation.

The following simplified values will be evaluated and visualized through Matlab scripts as a result:

- Minimum squared error of all runs over dimensions, for each problem (best-case-analysis)
- Maximum squared error of all runs over dimensions, for each problem (worst-case-analysis)
- Mean squared error of all runs over dimensions, for each problem with standard deviation

There will be no hard criterion, on which to decide which algorithm to take, since the CEC problems probably cannot be compared to the satellite problems at hand. Still, the intent is to identify robust and global algorithms which have more chances at finding feasible solutions.

To directly compare two algorithms, some papers like [15] use the Mann-Whitney U-Test. We will also use this test for new algorithms, to directly compare a new algorithm to the best old one.

4.6 Output & Result Evaluation

An additional part of this thesis is the evaluation of the results obtained by the optimizer through Matlab scripts. The planned design involves storing immediate output from the optimizer in a human-readable format to provide instant feedback on how well the optimization is doing. Additionally, binary files will be created and written with in-depth information on the exact parameter settings and the respective fitness evaluations. Through the simulation file, the user should be able to choose between multiple ways and the location of storing the obtained information. Matlab scripts will be written, which can read this raw data and use this information for example to estimate the cost function through evaluated

points.

4.7 Planned Algorithms

At the start, we will restrict ourselves to the more simple algorithms for optimization. Since the simulations do not usually offer a close formula, we can only employ derivative-free optimization algorithms. One could use gradient approximation, but many simulations are not expected to be continuous, so the gradient would probably not be very useful, or maybe even mislead the algorithm. The planned order of implementation is the following:

- Nelder-Mead Method
- Particle Swarm Optimization
- Artificial Bee Colony Optimization (ABC)
- Self Adaptive Differential Evolution (SaDE)

and if multi objective optimization becomes relevant:

- NSGA-II

These algorithms should already provide a first estimate, how efficiently we can solve the problems at hand. All listed algorithms require only a small amount of parameters, which we will gather from empirical analyzes by other researchers like in [25]. In a later step, we may combine several algorithms into a portfolio, to increase the performance and verify the results from [15], mentioned in section 3.1.

5 Implementation

This section provides an insight of the previous state of the simulator and the way things were changed in order to make optimization and other discussed features possible.

5.1 Current Structure

This subsection describes the state of some parts of the simulator, as they were found at the beginning of the implementation phase.

5.1.1 Simulation Core

The simulation core is the head of the simulation. It is a class only to be instantiated once, but does not use the Singleton pattern. It contains the essential information about the simulation like the path in which the simulation was executed, the path to the simulation file and the state of the simulation. The methods to parse the simulation file, instantiate and track the modules and push the parameters into the modules are also contained.

It is also responsible for getting the intermediate and final results from the modules and write them to the result file.

5.1.2 Burst Container

The burst container is basically a simple aggregate structure of data and variables to be passed between modules. The exact composition of the container depends on the project to be compiled and can contain a wide variety of information. At the start of the program, a fixed number of bursts will be generated and stored in a burst pool, from which they are drawn when needed, in order to avoid dynamic allocation during runtime.

5.1.3 Simulation Main Loop

The simulation main loop controls the execution and initialization of the modules. To be more precise, it is responsible for fetching the burst containers from the available pool and execute the modules in the right order, handing burst pool from module to module. The number of times this procedure is executed per simulation is dependent on a parameter in the simulation core.

However, this simplistic sequence of operations is only true if the simulation only consists of one section, whereas section 5.1.4 will show how a simulation can be split up into multiple sections for better performance.

5.1.4 Sections and Multi Threading

In the previously presented approach in section 5.1.3, all modules are executed according to the order in the simulation file, where bursts are handed from the top to the bottom module, before the procedure starts over once again. One single thread will handle this task. Alternatively, simulations can be split up into different sections, where each section can be processed by one or multiple threads. This complicates the handling of the burst containers: Each section now needs a queue at its beginning and end for the containers needed to be processed in this section and the ones which are ready for the next section respectively. A section can be marked as a parallel section with the tags `PAR_START` and `PAR_END`, meaning that all modules in it will be processed in parallel. The number of available threads for a parallel section is given at runtime as a command line argument. Alternatively, the command `PIPELINED` splits the flow into different sequential sections. A visualization of both cases is shown in figure 5.1.

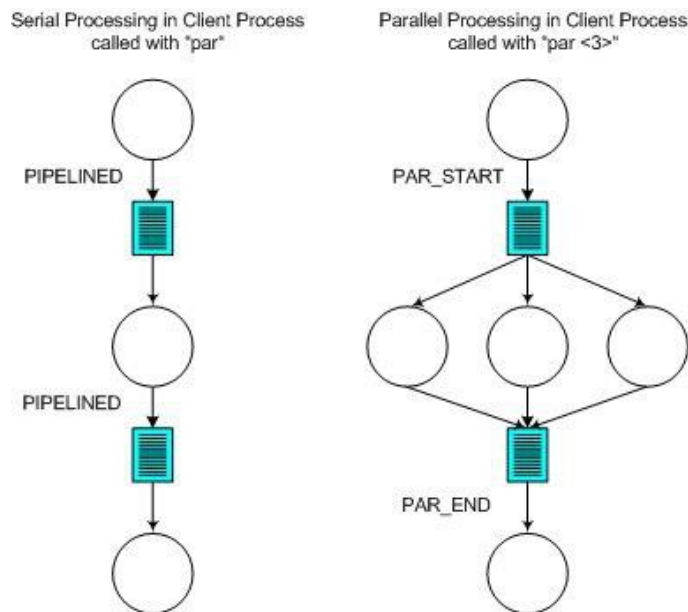


Figure 5.1: Visualization of the Workflow of the Simulation with different Sections

For parallel sections, all modules to be processed in parallel need to be instantiated for each thread.

5.2 Restructuring

Following the state prior to the changes of this thesis, shown in section 5.1, this section will provide a rough overview of the changes and additions made to the already existing software.

5.2.1 Simulation Core

The importance of the central core for the simulation was already discussed in section 5.1.1. Likewise, the optimizer is also an important central aspect and therefore the optimization module was integrated into the core. Additionally, methods to set parameters as variables, constraints, objectives or outputs were also added.

5.2.2 Burst Container

The burst container was not modified for the optimization process, but the modules for the link budget calculation and multi-beam scenario, presented in sections 4.3 and 4.4 respectively, needed to pass a few parameters between them. These were added in the burst container.

5.2.3 Simulation Main Loop

For the addition of the optimizer, the main loop of the simulation was extended: While it already contained three cascading loops for iterating multiple times each simulation, for each x_1 and each x_2 , another loop was added which runs the simulation until the optimizer converges or the maximum number of function evaluations has been reached.

The optimization algorithm has a flag at its disposal to signal premature convergence. If it is never set, the process is interrupted after the optimization has evaluated the simulation a given number of times, set by the operator in the simulation file.

The optimization module is responsible for setting the parameters for each simulation, since the main loop needed to remain as unaltered as possible.

There are currently no plans to use the optimization in conjunction with x_1 or x_2 parameters, although the possibility exists.

5.2.4 New Modules

A few major modules were added, which we will take a look at in this section.

Optimizer

Due to the fact that the optimizer demanded features similar to modules, like dynamic instantiation and parameter setting, it was implemented as one. However, the exact class hierarchy is not revolved around the base class Module, but around another base class named Optimizer, as we will later see in section 5.3.

Listing 5.1 shows how the optimizer can be instantiated. The module name is Optimizer, followed by the name of the algorithm to be used. In the file moduleswitch.h, the string after Optimizer is extracted and passed to the optimizer constructor as a parameter.

```

MODULE OptimizerPSO
  total_func_evals      4000
  population_size      29
  w                    -0.4349
  phi_p                -0.6504
  phi_g                2.2073
  //Saves precise binary data for matlab evaluation. Format: 'nr|path|nr|path|...', where nr = filehandler_id
  //For available filehandlers, see file_writer.h.
  //Two files will be created. 'fitness.bin' and 'params.bin' will be appended to the path name
  filehandlers         '2|C:\modemsim\sims\link_budget - 3 vars\matlab_data_|'

```

Listing 5.1: Exemplary instantiation of the Optimizer Module

The specific parameters for the algorithm are registered after this creation, like w, φ_p, φ_g in listing 5.1, needed for PSO (equation (2.5)), whereas *total_func_evals* and *filehandlers* are algorithm-independent parameters, responsible for the maximum number of function evaluations an algorithm may use and the binary log file for Matlab evaluation respectively.

The base class already provides a lot of functionality to ease the development of upcoming algorithms:

Number of Function Evaluations

We track the total number of exhausted function evaluations in the Optimizer-Module class, because it is limited to the number given in the simulation file by the user. The optimizer algorithm may signal that it has converged, ending the optimization prematurely, but if it is not converged, the module will stop the optimization after the total available number of function evaluations have been exhausted.

Manage Objectives, Constraints and Variables

The base module already handles the objectives, constraints and variables, by providing the simulation core with a way to register all parameters for the optimization. The algorithms itself want to track their own points, but they need to be independent of the parameters which they vary and the parameter(s) they want to optimize, ignoring the actual simulation parameter behind the scene. For evaluation, the current fitness is read and passed to the algorithm in the evaluation function.

Interface between Parameter Descriptions and Points (Vectors)

Optimization requires many, possibly complex calculations of vectors, whereas the simulation is handled through so-called 'Parameter Descriptions', storing information about the type and location of the parameter. Since we wanted to hide the implementation details of the simulator for the optimization algorithms, the Point-class acts as a simplified vector-like class

used for calculation. Arithmetic operations like addition, subtraction, scalar multiplication and scalar division have been implemented through operator-overloading to retain readability and ease-of-use of the algorithms. One such example can be seen in listing 5.2, where the loop to calculate and update the velocity of all individuals of the PSO algorithm is shown. This code snippets computes the necessary calculations, which were already presented in equation (2.5).

```
for (auto it = individuals->begin(); it != individuals->end(); it++)
{
    double r_p = global_rand.rand_uniform(), r_g = global_rand.rand_uniform();

    //Update velocity of the individual
    it->velocity = w * it->velocity + phi_p * r_p * (it->best_personal_position - it->current_position)
        + phi_g * r_g * (*best_global_position - it->current_position);

    //Bound velocity to the search space, so we will not violate min/max limitations of the variables
    boundPoint(*it);

    //Update position
    it->current_position = it->current_position + it->velocity;

    //Queue the individual for evaluation
    this->queued_evals->push_back(it->current_position);
}
```

Listing 5.2: Code to update the Velocity of each Particle, according to Equation (2.5)

In order to use a calculated point for evaluation, it can be placed into a queue, also provided by the base class. The base class will dequeue the points in the right order, set the parameters accordingly and run the simulation for each enqueued point before returning control to the optimization algorithm, with all calculated fitnesses. It is assumed, that after initialization and after every evaluation, some points were enqueued for evaluation.

To improve performance, the points which were set are remembered in a map to avoid multiple evaluations of the same simulation configuration. If points were enqueued, which were already evaluated, the simulation for this point will be skipped and a simple map lookup will provide us with the fitness.

Initialization is also simplified, since the algorithms for uniformly-distributed points already exist.

Tracking of Evaluations

In addition to the already mentioned map, another list of all function evaluations is kept. This list retains the information of the order, in which the points were evaluated, and also stores to which generation each configuration belongs. All points queued for evaluation at the

same time are considered to be within the same generation. Variable generation sizes are also supported, but this was not yet tested.

Furthermore, the base class always keeps track of the best known solution and one can always retrieve it.

Maximization/Minimization

Maximization, as well as minimization can both be done by the optimizer, where we exploit the theorem we already provided in equation (2.2): The base class will automatically multiply the fitness by -1, so that we can always view the optimization problem as a minimization problem. When storing the fitness into the output file, it will be translated back to the correct value.

Result Output

Since the operator wants immediate, readable results, a textual format is necessary to provide feedback of the optimization, especially during runtime. The optimization probably takes a long time and early termination might be required if the progress is insufficient. In such case, the applied algorithm or total optimization setup needs to be changed immediately to limit the computational time loss.

Consequently, a simple output file is produced, which immediately shows the current best found configuration. Listing 5.3 shows how such a file could look like. Every line contains the iteration a better solution is found, where the first iteration is always taken regardless of its fitness if it does not violate any constraints, the whole parameter setting and all output variables. As soon as a better solution is found, it is recorded and flushed to the file.

Binary Logging

The optimizer also provides the means for binary storage of all tried evaluations, after the simulation has been finished, for later in-depth analysis. The logging is initiated by setting the parameter *filehandlers* of the optimizer-module with the way of storage and the respective path. Results are written to the disk at the end of the simulation. More detail on how the data is stored and how it is loaded into provided scripts, will be available in section 5.2.5.

Output Variables

During development and a first few tests, the need arose to track some parameters, like availability, even though they are unrelated to the actual optimization, since they are neither variables, nor objectives, nor constraints. They are stored similar to variables.

```

Iteration  avg_goodput  crossover_point,  |  availability,  most_used_modcod,
-----
1   5587.73  1.75037,        |  0.9751,  5,
33  6484.17  1.14233,        |  0.9587,  1,
169 6507.47  1.07934,        |  0.94935,  1,
212 6518.42  1.02388,        |  0.92525,  1,
227 6524.6   1.08746,        |  0.95085,  1,

```

Listing 5.3: Exemplary Text Output File of an Optimization

Link Budget

In contrast to the original plan of implementing all of the link budget calculation in one module, hinted in listing 4.3, it was split between two modules, for up- and downlink respectively. This new concept now needed the burst container to store the results of the calculation, but this was necessary, since advanced simulations like interference mitigation will need the results of the calculation anyway later on.

```

//Atmosphere Models:  0: Uplink Graz, 1: Downlink Graz, 2: Uplink Tito, 3: Downlink Tito
//
MODULE UplinkBudgetModule
  antenna_diameter      3      //[m]
  antenna_efficiency    0.54   //[%/100]
  antenna_feed_loss     0      //[dB]
  distance_to_satellite 38175  //[km]
  depointing_loss       0.3    //[dB]
  power                 200    //[W]
  frequency              47.85 //[GHz]
  atmosphere_model      0

  sat_g_over_t          4.4    //[dB/K]

MODULE DownlinkBudgetModule
  antenna_diameter      3      //[m]
  antenna_efficiency    0.59   //[%/100]
  antenna_feed_loss     0.96   //[dB]
  distance_to_satellite 38175  //[km]
  tracking_loss         0.2    //[dB]
  frequency              37.85 //[GHz]
  mean_sky_temperature  270    //[K]
  ground_temperature    30     //[K]
  lna_noise_temperature 205    //[K]
  lna_transmission_loss 0.52   //[dB]
  atmosphere_model      1

  max_sat_eirp         39     //[dBW]
  sat_obo              1.1    //[dB]

```

Listing 5.4: Final Version of the Link Budget Modules

The two modules and an exemplary setting of their parameters can be seen in listing 5.4.

Interference

In order to implement the former mentioned scenario of multiple spot beams in section 4.4,

many more modules were implemented to achieve this task.

Listing 5.5 shows the script of a simulation file with the interference modules, which were already planned and presented in figure 4.10. In this listing we can also see some of the parameters of the modules. The beam setup records the planned coloring scheme (frequency reuse), crossover-level, available bandwidth in addition to the actual setup. Within the beam interference module, we record the path to the file containing the radiation pattern of the antenna. The modem setup obviously needs the used rolloff and modcod.

```
MODULE BeamSetup
  total_available_bw      50
  coloring_scheme         2  //1 = 1 color, 2 = 2 colors, 3 = 4 colors
  crossover_point         3  // [dB]
  coverage_area           543965e6  // [m2]

MODULE BeamInterference
  aimt                    1
  theta_min               -3.2  // [°]
  theta_max               3.2   // [°]
  radiation_pattern_file  'G:\Interference Mitigation\radiation_pattern.bin'

MODULE TransponderSetup
  number_of_transponders_per_beam 5
  number_of_carriers_per_transponder 5

MODULE SNIR

MODULE ModemSetup
  acm                      1
  rolloff                   0.2

MODULE OverallPerformance

END
```

Listing 5.5: Interference Modules

If the number of beams is set to zero and the crossover-level is known, the given coverage area defines the needed number of beams to cover the given area. We already looked at the covered area of a circle on the earth's surface, given the aperture angle and the distance to the antenna of the satellite in equations (4.22) and (4.23), but as we can see from figure 4.8 the circles do not cover the whole surface. The boundaries between the beams and the decision, to which beam a user belongs can better be expressed in terms of a hexagonal pattern as can be seen in figure 5.2. We have both red and blue hexagons, which demonstrate how the pattern could look like for a two color frequency reuse scenario. The centers stay the same and this way, the transition between the beams is seamless. For now, the grid constructed is as quadratic as possible, with $\lceil \sqrt{n} \rceil$ rows and $\lfloor \sqrt{n} \rfloor$ columns, where the last row is only filled until we have n beams in total. Assume a case where we need 200 spot beams:

The grid will consist of 14 rows with 14 columns and the 15th row will only be composed of 4 beams, since $14^2=196$.

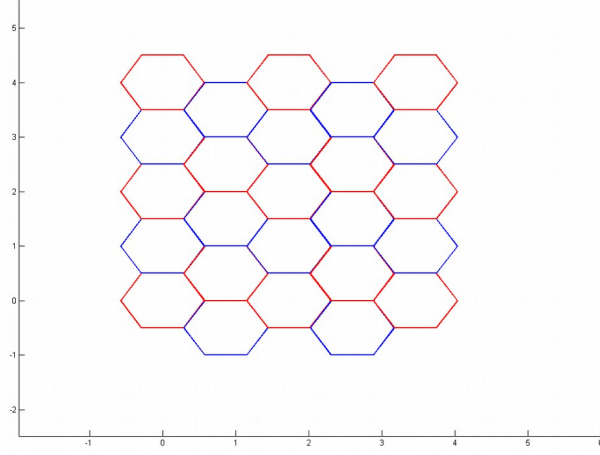


Figure 5.2: Hexagon Pattern in use

In order to determine the position of the user for which we will calculate the total interference, we choose a random hexagon in which we will place the user, randomly sample one of the six triangles it contains and then use equation (5.1) presented in [26] to uniformly pick a position P for the user within the triangle, where r_1 and r_2 are uniformly distributed random numbers within the interval $[0, 1]$ and A, B, C are the vertices of the triangle:

$$P = (1 - \sqrt{r_1})A + \sqrt{r_1}(1 - r_2)B + \sqrt{r_1}r_2 C \quad (5.1)$$

Due to the fact that our calculations are now based on the coverage of hexagons, the area calculation in equation (4.23) now became invalid for our application. Therefore we used equation (5.2) to compute the circumradius of the hexagon, and equation (5.3) to calculate the total area of the hexagon, which serves now as the area covered by a spot beam:

$$r_{circum} = \frac{2 r_{inner}}{\sqrt{3}} \quad (5.2)$$

$$A_{Hexagon} \approx 2.598 r_{circum}^2 \quad (5.3)$$

5.2.5 Matlab Tool

The implemented Matlab module is a simple collection of functions which serves the purpose of analyzing the binary data collected in the process of optimization. In this section, we will first take a look at the used binary format and the reasoning for it and continue with the implemented Matlab functions.

Binary Format

Since a simple stored list of numbers would be too less information for the binary file, some kind of meta data header was implemented to provide at least a bit of information about the file in use. The requirements demanded at least the names of the variables which were varied, on the one hand, and the objective function on the other hand. Since the meta data format is probably subject to change in the future, a version number is included in the meta data which should be changed each time the meta data structure is modified in order to ensure backwards compatibility.

The meta data as the day of writing is shown in table 5.1. Each string is terminated by a comma as the delimiter, which is thus a forbidden character in all strings.

(String)* means that there might be zero to n strings.

Name	Size [Bytes]	Data Type
Version Number	4	Unsigned Integer
File Handler ID	4	Unsigned Integer
Optimization Algorithm	Variable	String
Objective Name	Variable	String
Max-/Minimization	1	Boolean
Number of Variables	4	Unsigned Integer
Variable Names	Variable	(String)*
Number of Constraints	4	Unsigned Integer
Constraint Names	Variable	(String)*
Number of Outputs	4	Unsigned Integer
Output Names	Variable	(String)*

Table 5.1: Current Meta Data Structure

Since we are perhaps not interested in all gathered information but only part of it, there were multiple ways implemented of storing the known evaluations. For this purpose, another simple ID is saved alongside the meta data to identify the representation used. The currently available 'file handlers', as they are called in the software, include:

ID 1

This file handler, currently known as 'SingleObjectivePerformancePreprocessor', flattens the retrieved values, in the sense that we drop the information to which generation a configuration belongs (look at section 5.2.4 for exact information) and only saves the current best known configuration. When a better configuration is found, the iteration it was found is stored as an unsigned integer, followed by the configuration. The size of the resulting file is heavily dependent on the optimization problem.

ID 2

If we choose the file handler 'SingleObjectiveFlattenedRawSave', the data will again be saved in a flattened format, but this time each function evaluation will be saved every iteration along with its parameters, no matter how bad or good it is. This can provide us with all gathered evaluations of the whole optimization, but will drop the generation information,

which could provide us with further insight how well the optimization algorithm is doing for each generation.

ID 3

The file handler 'SingleObjectiveRawSave' with the ID 3, is the file handler which stores the maximum amount of information. Like the previous file handler, it stores all evaluations, but before it starts to save another generation of fitnesses or parameters, it writes the size of the upcoming generation on the file. This way we can reconstruct to which generation a configuration belongs. In the next section, we will use such information to track the fitness of each particle as it rises and descends on a function.

ID 4

With this file handler, we only save the best retrieved configuration, thus the name 'SingleObjectiveOnlyBest'. This may seem to be too less information to use, since the text output file, which we previously addressed in section 5.2.4, already provides more information without the need for scripts. However, this format may be useful for Benchmark applications since we can immediately spot the best solution obtained, without using much space on the HDD and easily parse the value obtained into a program.

Matlab Functions

The Matlab Functions were kept as simple as possible. To load the information gathered by the Optimizer, one simply needs to use the function 'readData' to load the meta data, parameters and fitnesses, like shown in listing 5.6.

```
[meta_data, params, fitnesses] = readData('data/pso_low_fes/params_raw.bin', ...  
                                         'data/pso_low_fes/fitnesses_raw.bin');
```

Listing 5.6: Command to call the Script for Processing the Binary Files

The function will load the data according to both the version number of the meta data and the ID of the file handler. Any modifications and additions to these values, file handlers or the meta data structure have to be reflected on this function. The whole 'readData' script is shown in the Appendix.

To demonstrate the possibilities of reading the evaluation data and analyzing it in Matlab, we again considered the examples of the CEC, presented in [24]. The surface of the chosen function with the number eight can be seen in figure 5.3. It poses a lot of complexity with a very noisy surface and only a very narrow chasm which holds the global minimum, surrounded by many local minima in which algorithms tend to get stuck.

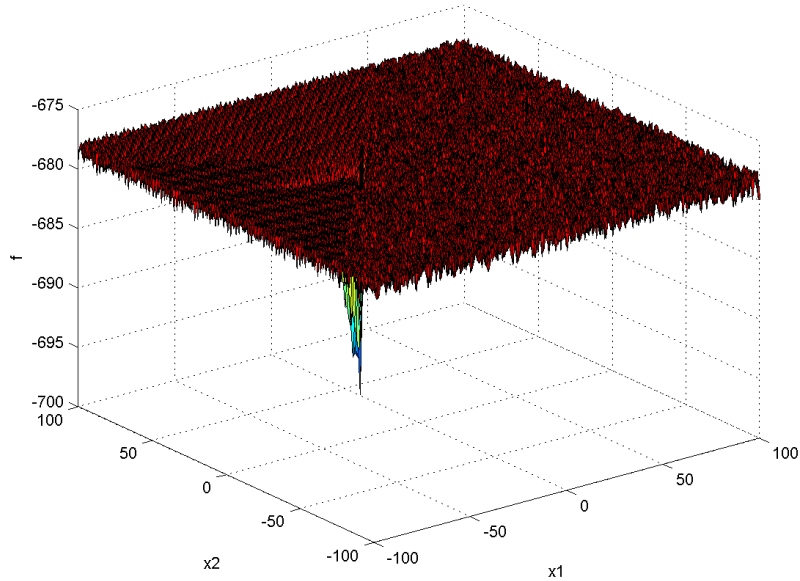


Figure 5.3: Surface of Function Number Eight of the CEC 2013 challenge [24]

To find the global minimum, we applied the PSO algorithm to the function, with only 500 function evaluations. Adding the fact that we had a two-dimensional problem at hand (chosen for easier visualization), we decided to choose the following parameter setting according to [25]:

- Total number of Particles (S): 29
- Inertia (ω): -0.4349
- Cognitive Factor (φ_p): -0.6504
- Social Factor (φ_g): 2.2073

```

clear all;
close all;

func_nr = 8;

showfunction(func_nr, 1);

[meta_data, params, fitnesses] = readData('data/pso_low_fes/params_raw.bin', ...
                                         'data/pso_low_fes/fitnesses_raw.bin');

%Code for Interpolation from
%http://de.mathworks.com/help/matlab/visualize/representing-a-matrix-as-a-surface.html
params_flat = reshape(params, [2, 29*18]);
x = params(1,1:500);
y = params(2,1:500);
fitnesses_flat = reshape(fitnesses', [1, 29*18]);
z = fitnesses_flat(1:500);
xlin = linspace(min(x),max(x),33);
ylin = linspace(min(y),max(y),33);
[X,Y] = meshgrid(xlin,ylin);
f = TriScatteredInterp(x',y',z');
Z = f(X,Y);

fig = figure;
mesh(X,Y,Z) %interpolated
axis tight; hold on
plot3(x,y,z, '.', 'MarkerSize',15) %nonuniform
xlabel('x1');
ylabel('x2');
zlabel('f');
hold off;

saveas(fig, 'interpolated.png');

```

Listing 5.7: Script to interpolate the Surface from the Function Evaluations

The script shown in listing 5.7 loads the parameters from the file and uses the built-in Matlab function 'TriScatteredInterp' to interpolate a surface using the sampled points, resulting in the surface shown in figure 5.4. The blue dots mark the evaluated points, which were considered for the interpolation. Comparing figures 5.3 and 5.4, we can see that the algorithm was able to reconstruct the surface fairly well, also building a chasm with the global minimum. It is not as deep as in the original function, but some of the properties of the surface remain nonetheless.

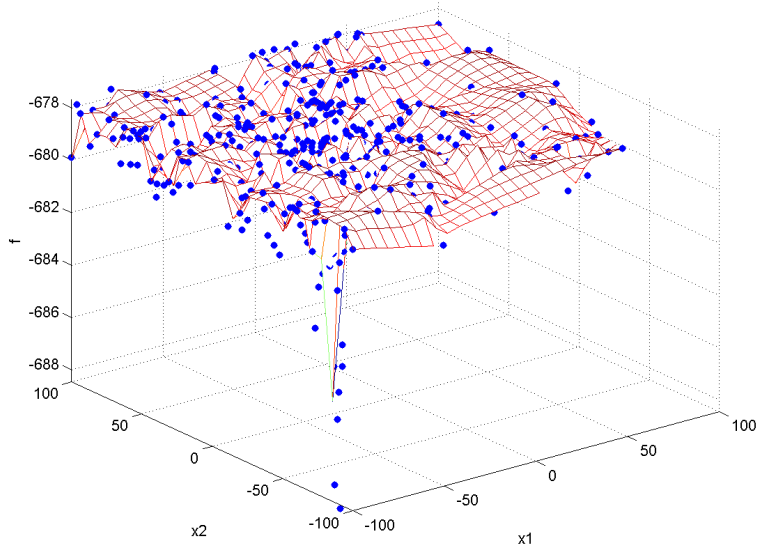


Figure 5.4: Interpolated Surface Reconstruction. Blue Dots mark the sampled Points by the PSO algorithm

In the script shown in listing 5.7, we receive the parameters as a $2 \times 29 \times 18$ matrix, since we have two parameters, with a generation size of 29 and the procedure was limited to 500 function evaluations (the last 22 entries of the matrix are filled with NaN entries). With the gathered information we can also track the evaluation and path of each particle. We decided to plot the fitness evolution of each particle for demonstration purposes, which can be seen in figure 5.5. Some particles have one evaluation less since 500 function evaluations cannot be evenly divided among 29 particles.

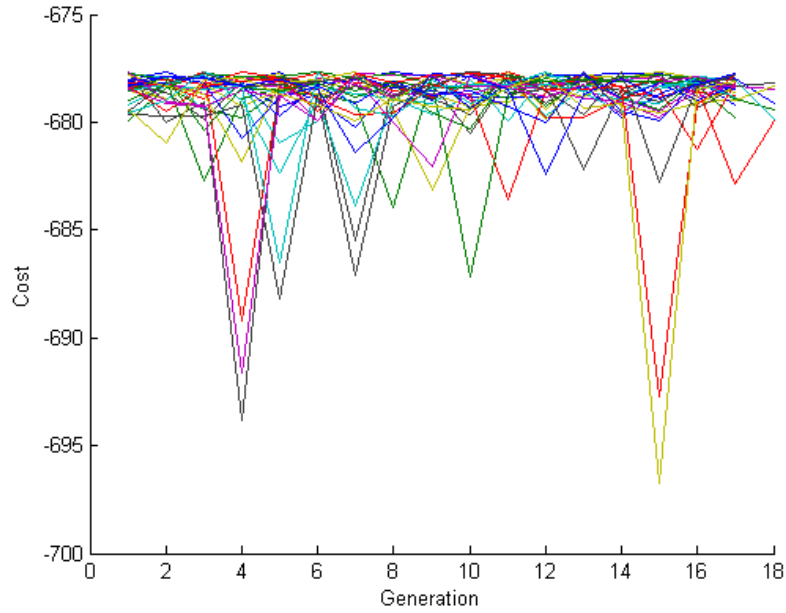


Figure 5.5: Performance Evaluation for every Iteration of every Particle

5.3 Final Class Diagrams

After the implementation phase, the most important classes, their attributes and methods have been recorded to create a class diagram. Figure 5.6 presents the classes associated with the Optimizer, while the classes created for the link budget and interference experiments were separately visualized in figure 5.7.

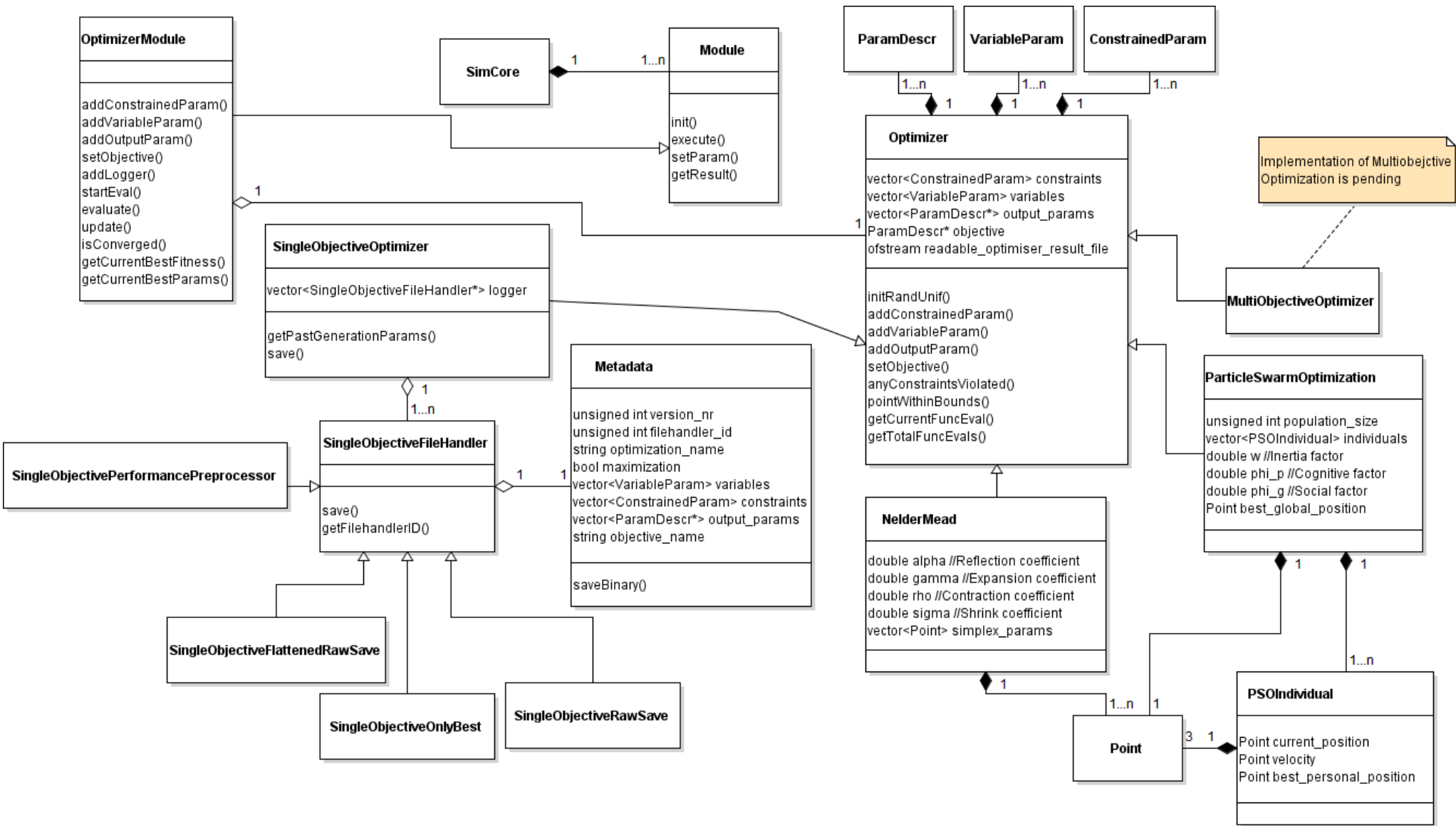


Figure 5.6: Class Diagram of the Optimization Classes

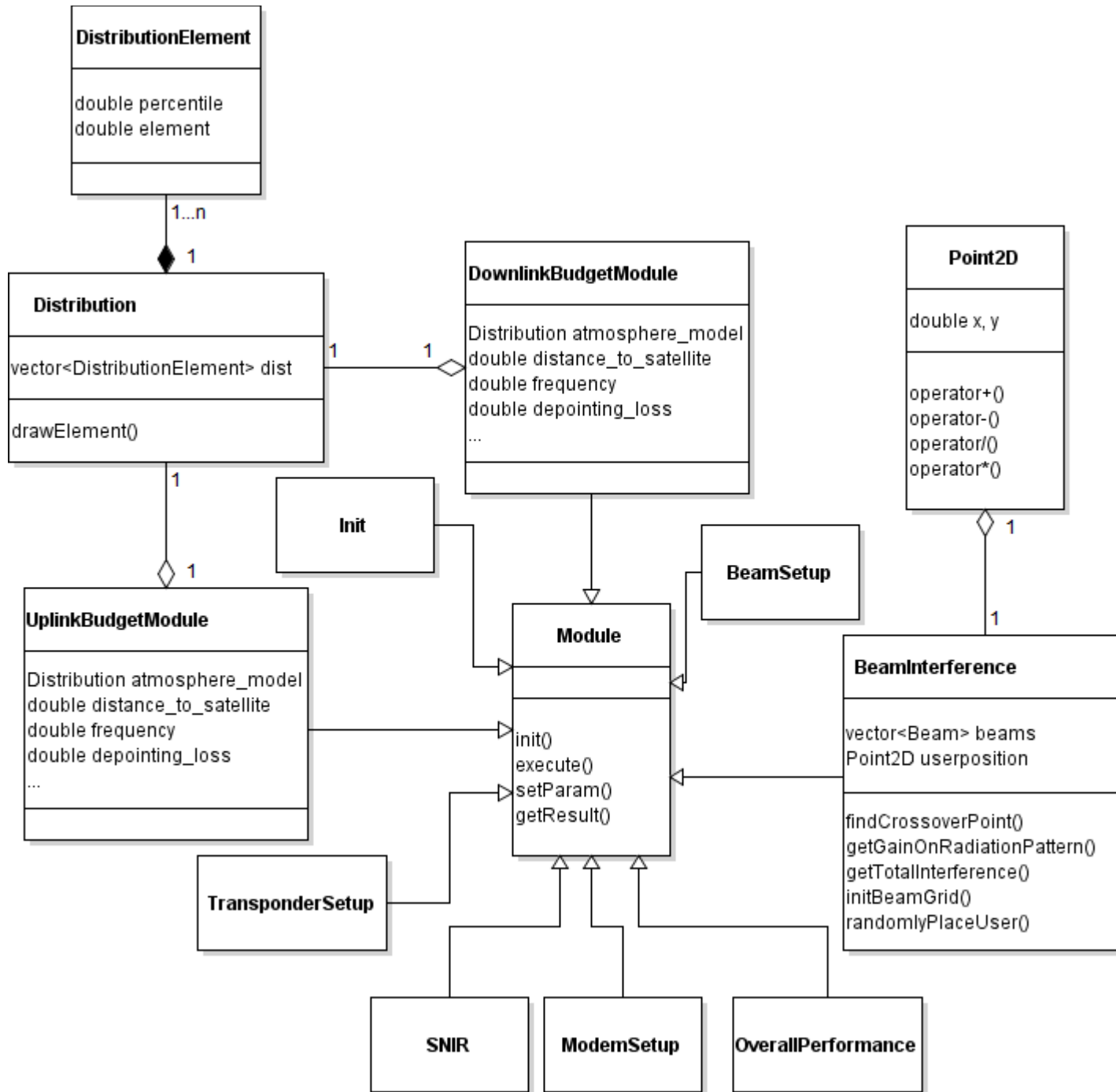


Figure 5.7: Class Diagram of the Link Budget and Interference Classes

6 Evaluation

This section covers discussing the results which we obtained by the software developed in this context. We will start off by looking how well the optimization algorithms achieve on the CEC functions in section 6.1. Furthermore, we will analyze if the parameters we chose from the literature for PSO can perform well, even on high dimensional functions, or if we will need to implement other optimization algorithms for such complex optimizations.

Sections 6.2 through 6.4 deal with optimization experiments of increasing complexity and their in-depth analysis.

6.1 Benchmark

In contrast to the proposed benchmark in section 4.5, where we suggested to test problems up to 100 dimensions, we deemed the benchmark to be sufficient for the dimensions $D \in \{2, 5, 10, 20, 30\}$, since our first experiments will surely not exceed these dimensions. We stuck to the CEC function set in [24].

Surprisingly, even though Nelder-Mead is a very simple, deterministic algorithm, it can severely outperform PSO in some occasions. On the first function, which is a simple sphere function given by equation (6.1), out of 51 runs, Nelder-Mead could always find the minimum of the function, whereas PSO struggles from 10 dimensions upwards.

$$f_1(x) = \sum_{i=1}^D (x_i - o_i)^2 - 1400 \quad (6.1)$$

On more complex functions, like the Ackley function, shown in figure 5.3, both algorithms have similar best and average errors, but PSO was not able to clearly outperform Nelder-Mead in the total of 51 runs.

For two reasons PSO is still the algorithm of choice for the upcoming experiments: Firstly, it provides better general exploration by not getting stuck in a local minima, which we can deduce from the fact that the best, worse and average case are closer together than for Nelder-Mead (compare figures 6.1 and 6.2). Secondly, because applying Nelder-Mead would demand a restart feature, which runs the simulation several times, demanding additional computational resources.

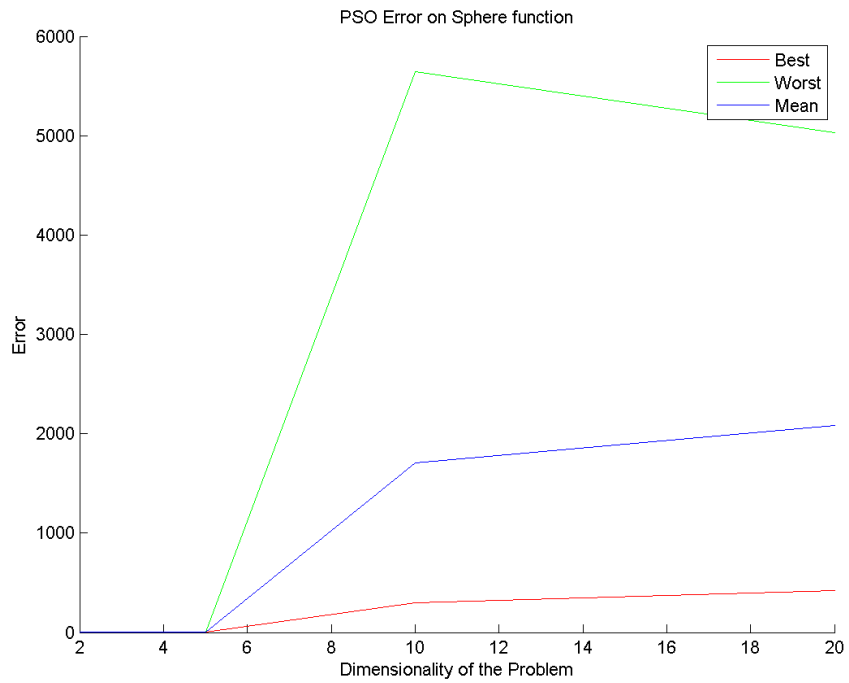


Figure 6.1: Best, Worst and Average Performance of PSO on the Sphere Function

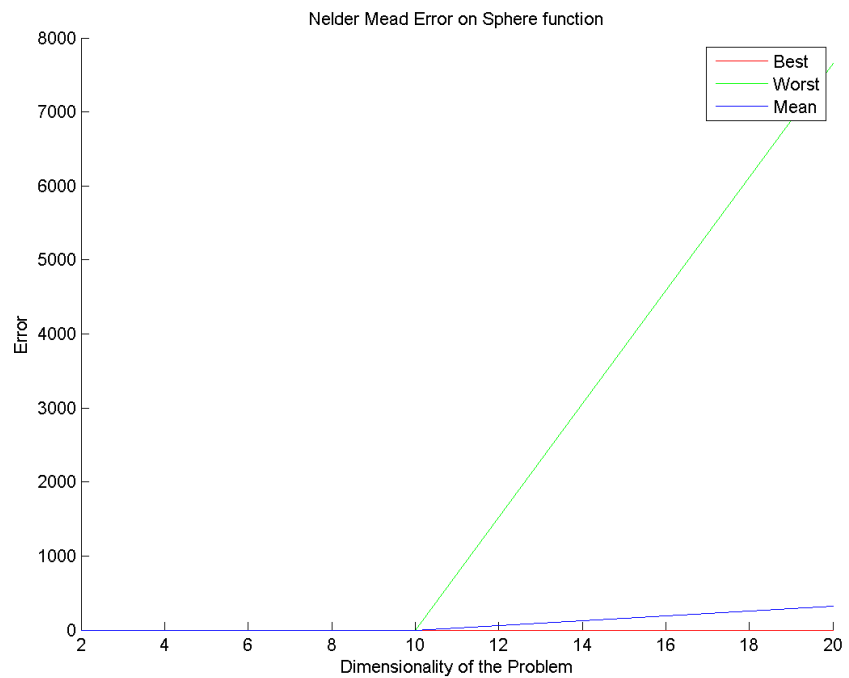


Figure 6.2: Best, Worst and Average Performance of Nelder-Mead on the Sphere Function

6.2 Simple Crossover Experiment

For this experiment, we analyze the dependency of the average goodput, of the multiple spot beam scenario presented in section 4.4, on the crossover-level of the spot beams. We first calculate how many spot beams we need to cover the given area, then place a user randomly per simulation point on the covered area. For this user, the whole link budget calculation presented in section 4.3 is performed, the total C/N and C/I values are calculated and we use these to get the total SNIR value. The atmospheric attenuation is independently drawn from a distribution given in the simulation file for each simulation step. If the ACM algorithm is turned on, we simply choose the most effective modulation and coding scheme for the current SNIR, ignoring the possible delay between setting the scheme on the ground station and receiving the change on the satellite. This feature would give rise to the need of interactive fade slopes when drawing from the atmospheric attenuation, perhaps overcomplicating the planned experiment with only little impact on the results.

The average goodput is defined as the average bits correctly received per transmission. We decided that a transmission is erroneous, simply if the calculated SNIR is lower than the threshold according to table 6.1, where the values for this table were taken from [22]. These are the Modcods which are used in Alphasat Aldo Paraboni Payload experiment [1].

SNR threshold [dB]	Modcod	Efficiency [bits/s/Hz]
1.3	QPSK-1/2	1
2.9	QPSK-3/5	1.2
3.6	QPSK-2/3	1.33
4.4	QPSK-3/4	1.5
5.5	QPSK-5/6	1.66
6.6	QPSK-8/9	1.77
7.3	8PSK-2/3	2
8.5	8PSK-3/4	2.25
9.7	16APSK-2/3	2.66
10.7	16APSK-3/4	3
12.3	16APSK-5/6	3.33
13.4	16APSK-8/9	3.55

Table 6.1: Modcod Thresholds from [27]

If we use the ACM algorithm, the only case where a frame error occurs, is when the SNIR is lower than the threshold required for the least efficient Modcod, which is 1.3dB for QPSK-1/2 in our case.

For the first setup, we will consider a land mass of $543,965\text{km}^2$ ³, which is approximately the area of metropolitan France. Weather conditions are assumed to be similar to the Austrian atmospheric models, shown in tables 6.2 and 6.3 for the downlink and uplink respectively.

3 According to http://www.auswaertiges-amt.de/DE/Aussenpolitik/Laender/Laenderinfos/01-Nodes_Uebersichtsseiten/Frankreich_node.html (retrieved at 23rd August 2015)

Annual Time [%]	Atmospheric Attenuation [dB]
75	0.5
50	0.6
10	1.6
5	3.7
1	8.1
0.5	10.8
0	25

Table 6.2: Distribution of the Atmospheric Attenuation, Q-Band, Location Graz

Annual Time [%]	Atmospheric Attenuation [dB]
75	1.6
50	1.8
10	3.2
5	6.3
1	12.4
0.5	16
0	30

Table 6.3: Distribution of the Atmospheric Attenuation, V-Band, Location Graz

For the calculation of the C/No-value in the link budget, the following configuration was considered:

Uplink

- Antenna Diameter: 3m
- Antenna Efficiency: 54%
- Antenna Feed Loss: 0 dB
- Distance to Satellite: 38,175 km

- Depointing Loss: 0.3 dB
- Power: 200W
- Frequency: 47.85 GHz
- Wavelength: 0.00626 m
- Antenna Gain (equation (4.2)): 60.87 dB
- G/T: 4.4 dB/K

Downlink

- Antenna Diameter: 0.6m
- Antenna Efficiency: 0.59
- Antenna Feed Loss: 0.96 dB
- Distance to Satellite: 38,175km
- Tracking Loss: 0.2 dB
- Frequency: 37.85 GHz
- Wavelength: 0.00792 m
- Antenna Gain (equation (4.2)): 45.239 dB
- Mean Sky Temperature: 270K
- Ground Temperature: 30K
- LNA Noise Temperature: 205K
- LNA Transmission Loss: 0.52 dB
- Satellite EIRP working point: 56.9 dBW
- 2 Color Frequency Use
- Bandwidth: 50 MHz -> 25 MHz per Color
- Crossover Point: [0.5, 5]dB
- Antenna Radiation Pattern similar to figure 4.5
- Advanced Interference Mitigation
- ACM on

- Rolloff: 0.2

Advanced Interference Mitigation was simply implemented by ignoring the effect of the strongest two interferers for the two color case, or the strongest six interferers for scenarios with only one color.

We used PSO with 400 available function evaluations and the setting we already used in section 5.2.5. The parameter to optimize is the average goodput calculated by the 'OverallPerformance' module (see figure 4.10), which measures the effective throughput of the whole system on the whole covered surface area. The only parameter varied is the crossover point.

The output of the optimization can be seen in listing 6.1, showing the best found solution is obtained using a crossover-point at approximately 1dB, with a total average goodput of 7.6GB/s, an availability of 97% and the most used Modcod is QPSK-1/2 (index 1). The inefficient Modcod is required since the interference becomes more and more of an issue with the decreasing crossover-level. Solutions with higher Modcods, like the first one recorded, are less efficient, since they cannot match the additional goodput obtained by having more spot beams available.

Iteration	avg_goodput	crossover_point,		needed_beams,	availability,	most_used_modcod,
1	7371.09	1.344,		216,	0.99385,	4,
17	7373.31	0.953875,		302,	0.93915,	1,
27	7591.45	1.03798,		279,	0.98245,	1,
83	7603.64	0.990534,		288,	0.97635,	1,
178	7629.02	1.00395,		288,	0.9784,	1,
366	7630.61	0.992675,		288,	0.97735,	1,

Listing 6.1: Output of the Optimization, showing the best Results

To increase the information gained, also the raw binary format for Matlab evaluation was used to get more insight into the function. The result, visible in figure 6.3, shows the average goodput as a function of the chosen crossover-level. Since we selected arbitrary step sizes, the optimizer can even try out small variations in the parameters and we can verify the obvious result that around the maximum we have many more evaluations than in other areas.

We can see that if we lower the crossover-level to less than the 1dB, the goodput dramatically decreases, since the interference becomes too much of an issue, impacting on the calculated SNIR. The diminishing goodput at higher crossover-levels will partially be compensated by better Modcods.

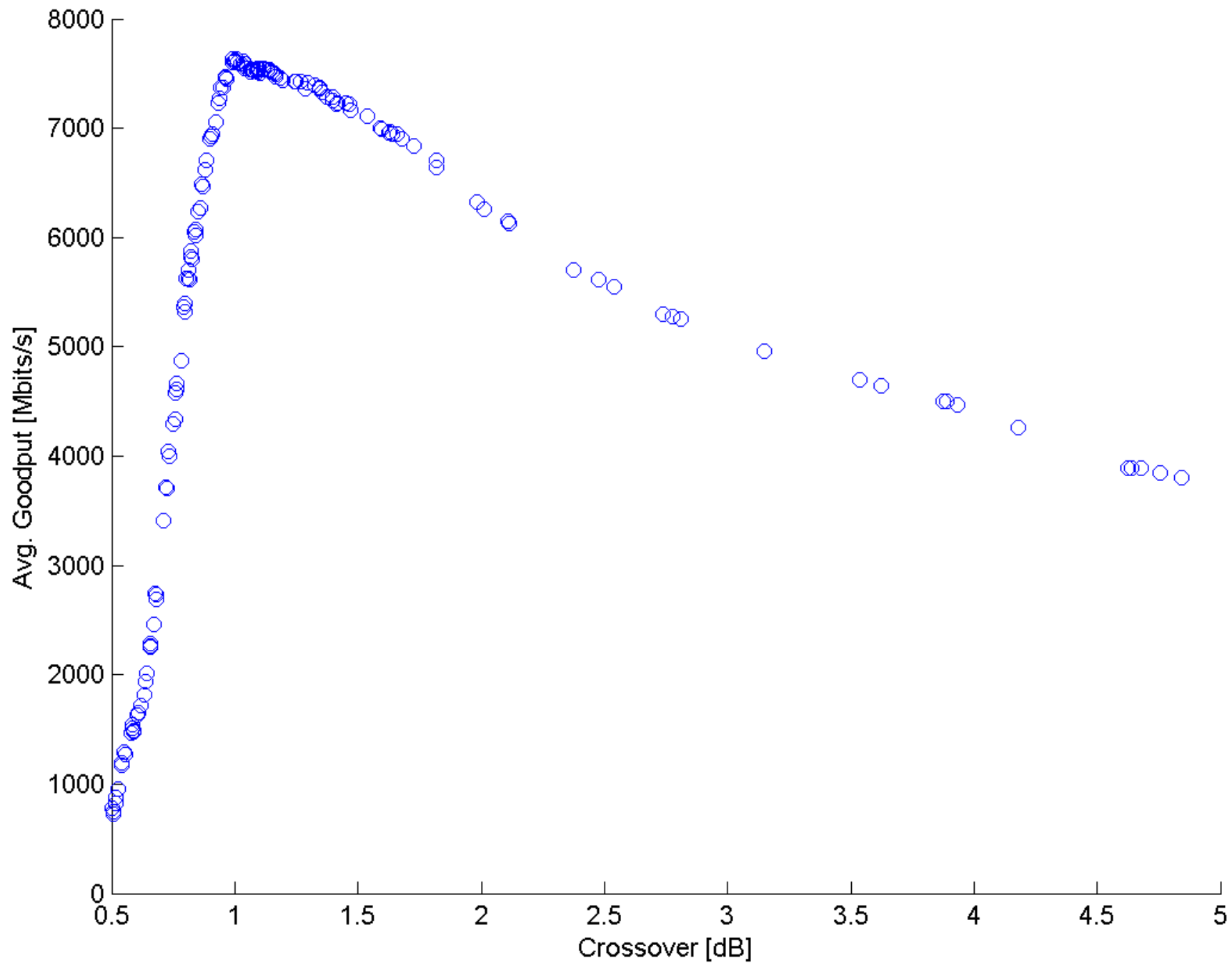


Figure 6.3: Crossover vs Avg Goodput on the whole Surface

Let's consider the plausibility of this example: One would assume that the dominating factor of how many beams can be employed is the available power, since, as we increase the number of beams, we need to split the available power on the satellite between them. However, the power flux density at the ground, calculated according to equation (4.8), is constant since proportionally to the lower power of the beam, its coverage area is also decreased.

If we lower the crossover-level to such minimal values like shown 1.1dB, we need 288 beams to cover the before mentioned area of 543,965km², and each beam only covers less than 2,000 km². In order to achieve this, we would need to build a satellite with more and bigger spot beam antennas to keep the aperture angle low enough. These are the optimal solutions, because we do not take any physical or financial limitations into account, or even consider them in the target function to be optimized.

We can conclude that crossover-levels of this magnitude are currently unrealistic, but the optimization showed that theoretically more beams are always better until we cross the threshold of losing too much SNIR.

This simple experiment serves more the purpose as a proof-of-concept, than a real optimization problem: One could simply use the already implemented 'x1' tag to mark the crossover-level as a variable parameter and look at the resulting average goodput.

6.3 Advanced Interference Mitigation Experiment

For this example, we do not change the parameters of the link budget and interference calculations from the previous experiment, but this time we let the optimizer compare different possible scenarios regarding the frequency reuse:

- 1 vs 2 color vs 4 color frequency reuse (50 vs 25 vs 12.5 MHz Bandwidth per color)
- Advanced Interference Mitigation on/off (no effect on the 4 color case)

The result this time is not as clear as the previous if we think about it: In the 4 color case, we have less interference and additionally a better C/N value, since according to equation (4.17), C/No needs to be divided by the available bandwidth. The better SNIR should allow for higher Modcods, but since the baudrate per carrier is calculated according to equation (6.2), where B is the occupied bandwidth and α is the rolloff factor (0.2 for our case), the effective baudrate per beam is halved.

$$\text{Baudrate} = \frac{B}{1+\alpha} \tag{6.2}$$

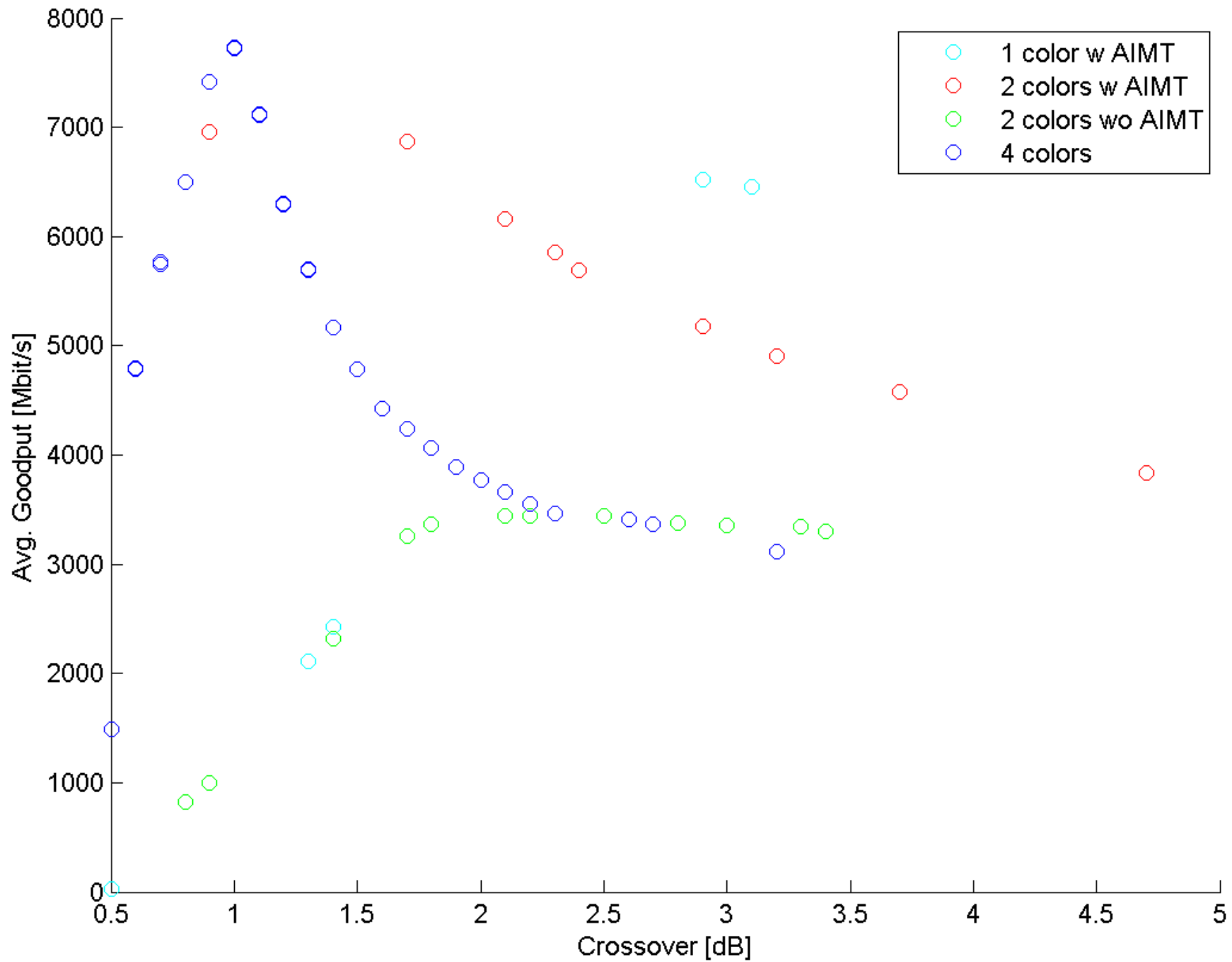


Figure 6.4: Second Experiment, comparing different Frequency Reuse Settings

The trace of the optimization can be seen in figure 6.4, also demonstrating how much inferior the 2 color case without AIMT is. Note that for faster optimization we've chosen a step size of 0.1 dB steps, since it also catches the properties of the dependency on the crossover-level.

Both the 2 and 4 color cases have their maximum near the low crossover-level of 1dB, already described in the last experiment. However, while the performance of the 4 color case drastically drops by increasing the crossover-level, the effect is much less apparent on the 1 and 2 color cases, making them much superior for the more realistic scenarios.

The reason behind the significant drop is related to the used Modcods and their efficiency, already presented in table 6.1: Through plotting the efficiency of the used Modcod over the linear SNIR required, we received the graph in figure 6.5. It is clear that by some very early point already, the increased SNIR does only little to increase the efficiency. Thus, the better Modcod cannot compensate for the loss of bandwidth in each beam.

However, the superiority of 1 color over 2 colors for higher crossover-levels is again probably not realistic, since the AIMT algorithm would need to completely cancel the effect of all six closest interferers in order to achieve this result. AIMT is probably much more effective for 2 colors, than it is for 1 color.

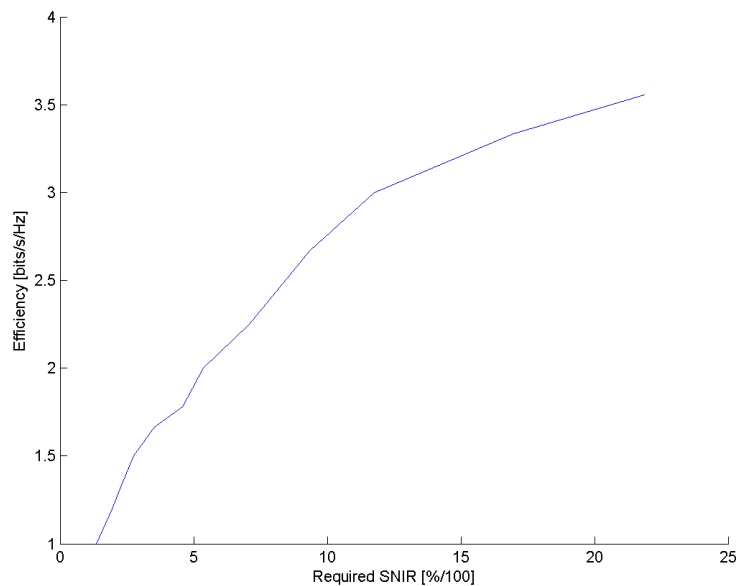


Figure 6.5: Modcods and their required linear SNR (Threshold)

6.4 System Cost Experiment

For this experiment, we now consider the cost of the satellite system under design, which we

left out until now, favoring expensive systems for the sake of high throughput. As a first test, we only take the power of the satellite's HPA and the number of required spot beams into account, where we calculate the cost of the system according to equation (6.3). The EIRP of the satellite is measured in dBW, whereas the resulting costs have no real relation to any currency. n is the number of spot beams. This artificial cost function was used for demonstration purposes, but satellite operators should replace this function with the exact costs of a system to get more realistic optimization results.

$$\text{Cost} = 10^{EIRP/10} \cdot 15 + 3n \quad (6.3)$$

Because we want a good balance between goodput and the costs and not simply minimize or maximize one of them, which is a trivial optimization, we additionally defined the target function f in equation (6.4), where $\overline{\text{goodput}}$ is the average goodput of the whole system.

$$f = \frac{\overline{\text{goodput}}}{\text{Cost}} \quad (6.4)$$

Since the optimization algorithms compute the finite differences, the absolute values have only a negligible influence on the optimization. The more important aspect is the order of relation between the input variables and the output, similar to complexity classes for measuring the computational requirements of algorithms.

For this setup, we redefine a few of the setup parameters:

- Crossover Point: [0.5, 5]dB
- Advanced Interference Mitigation on
- Satellite EIRP working point: [5, 100] dBW
- Bandwidth: 500 MHz

Since the costs increase exponentially with respect to the HPA power in dBW, the optimizer will try to find the trade-off between transmission power for a better C/N value and subsequently a higher Modcod, and the cost, at which the additional goodput comes.

Since we now have more variables, we also adapted the used parameters for the PSO algorithm and increased the number of available function evaluations to 4000 according to [25]:

- Total number of Particles (S): 156

- Inertia (ω): 0.4091
- Cognitive Factor (φ_p): 2.1304
- Social Factor (φ_g): 1.0575

In figure 6.6, the resulting point cloud is shown of all evaluations the algorithm has made. We can see that the 4 color case dominates all other cases at about 54 dBW EIRP satellite power. For higher EIRP values, the comparison between the different color cases is again similar to the previous experiments, showing that two and four color can both offer the same performance. Looking at the result file, we can get more insight into the reason for this performance: The power is reduced and with it, the availability of the link drops to about 84% and the modulation is once again only QPSK-1/2. However, the saved money outweighs the loss in terms of goodput. The optimal configuration found is shown in table 6.4.

<i>Target</i>	<i>Variables</i>			<i>Outputs</i>		
<i>f</i>	EIRP [dBW]	#Colors	Crossover [dB]	Avg. Goodput [Mbits/s]	Most used Modcod	Availability [%]
0.00811327	53.9	4	0.9	29881.4	QPSK-1/2	84.46%

Table 6.4 Best found Solution for the Cost Experiment without any Constraints

The question left is, why is the 4 color case so much superior for cheap systems. The reasoning is that by choosing a low crossover-level, we place the beams very close to each other and the AIMT is only able to eliminate the two strongest interferers for 2 colors. However, the next beams of the same color are close enough to play an important role in the interference, once again. In comparison, the 4 color setup has only half the interferers around the main beam.

Since we do not want to run the satellite at its minimum working point in order to save as much money as possible, we introduced a constraint, demanding at least 99% link availability from the system. We again plotted another point cloud, seen in figure 6.7, but we now receive much less points since many configurations became invalid, giving them $-\infty$ fitness.

Once again, the two color case becomes superior in realistic scenarios: Many tried configurations now have higher crossover-levels from 2.5 up to 4 dB, the power of the satellite is dependent on the crossover-level, but 56-58 dBW is a good measure and due to the high availability, the most used Modcod now is at least 8PSK-2/3, up to the most efficient available Modcod of 16APSK-8/9. We have only given intervals for these solutions, since we

get different results in multiple runs.

This is one hand due to the before mentioned fact that PSO has problems with hard constraints and on the other hand that PSO is probably not well fitted for these sensitive problems. The algorithm does insufficient exploration and the parameters would need to be tuned towards the problem.

In table 6.5, we can see one of the best solutions found.

<i>Target</i>	<i>Variables</i>			<i>Outputs</i>		<i>Constraints</i>
f	EIRP [dBW]	#Colors	Crossover [dB]	Avg. Goodput [Mbits/s]	Most used Modcod	Availability [%]
0.00310763	58.8	2	2	34305.2	8PSK-3/4	99.485

Table 6.5 Optimal Solution after adding the Availability Constraint to the former Case

<i>Target</i>	<i>Variables</i>				<i>Outputs</i>		<i>Constraints</i>
f	EIRP [dBW]	Bandwidth [MHz]	Nr. Colors	Crossover [dB]	Avg. Goodput [Mbits/s]	Most used Modcod	Availability [%]
0.00252593	57.2	165	4	1	19886.6	8PSK-3/4	99.04
0.00200722	58.8	233	4	1.2	22840.9	8PSK-2/3	99.03
0.00228311	59.9	276	4	1	33469.1	8PSK-3/4	99
0.00264635	55	104	4	1	12555	8PSK-3/4	99.07
0.00259688	56.2	135	4	1	16240.6	8PSK-3/4	99.04

Table 6.6 Multiple Optimal Cases found after adding the Bandwidth as a Variable

To raise the bar even higher, we now added the bandwidth as a variable for the optimization. The bandwidth can be varied from 100 to 500 MHz in 1 MHz steps with no additional costs attached.

There is a simple linear relation between the throughput and the bandwidth, shown in equation (6.2), suggesting that by simply increasing the bandwidth, we can increase the total

goodput, but if we take a look at equations (4.6) and (4.17) for calculating the C/N value, the situation becomes more complicated: The lower EIRP on the satellite decreases the C/No value, but we could compensate this effect by reducing the bandwidth, saving money by employing a cheaper HPA. Indeed, as shown in table 6.6, we now receive solutions where we do not make use of the full available bandwidth. We optimized several times, because we received many different solutions with similar fitness. In table 6.6, we have written the configurations obtained in five different optimization runs, showing the diversity of the results and the importance of a proper cost function.

The things that stay the same throughout the runs are the availability which is kept above 90% in order to avoid violating the constraint, the Modcod which is somewhere in the upper half of the possible Modcods, using four colors to make a cheap setup with a low crossover-level possible, which also stays the same for nearly all results obtained. Like in figure 6.6, the system tends to lean towards a very cheap system with minimal throughput, satisfying the constraint.

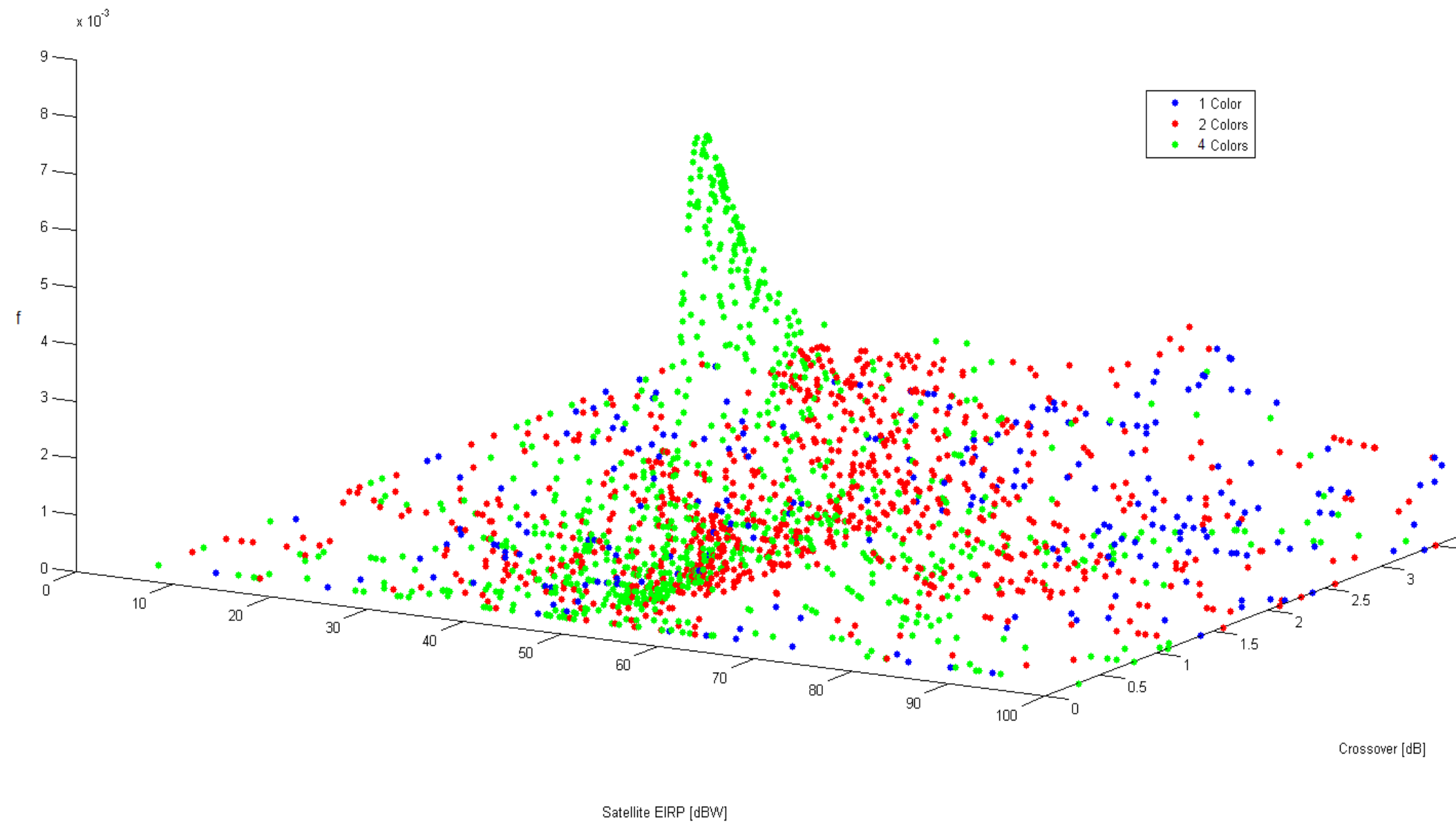


Figure 6.6: Point Cloud of the Third Experiment: Varying the EIRP and Crossover

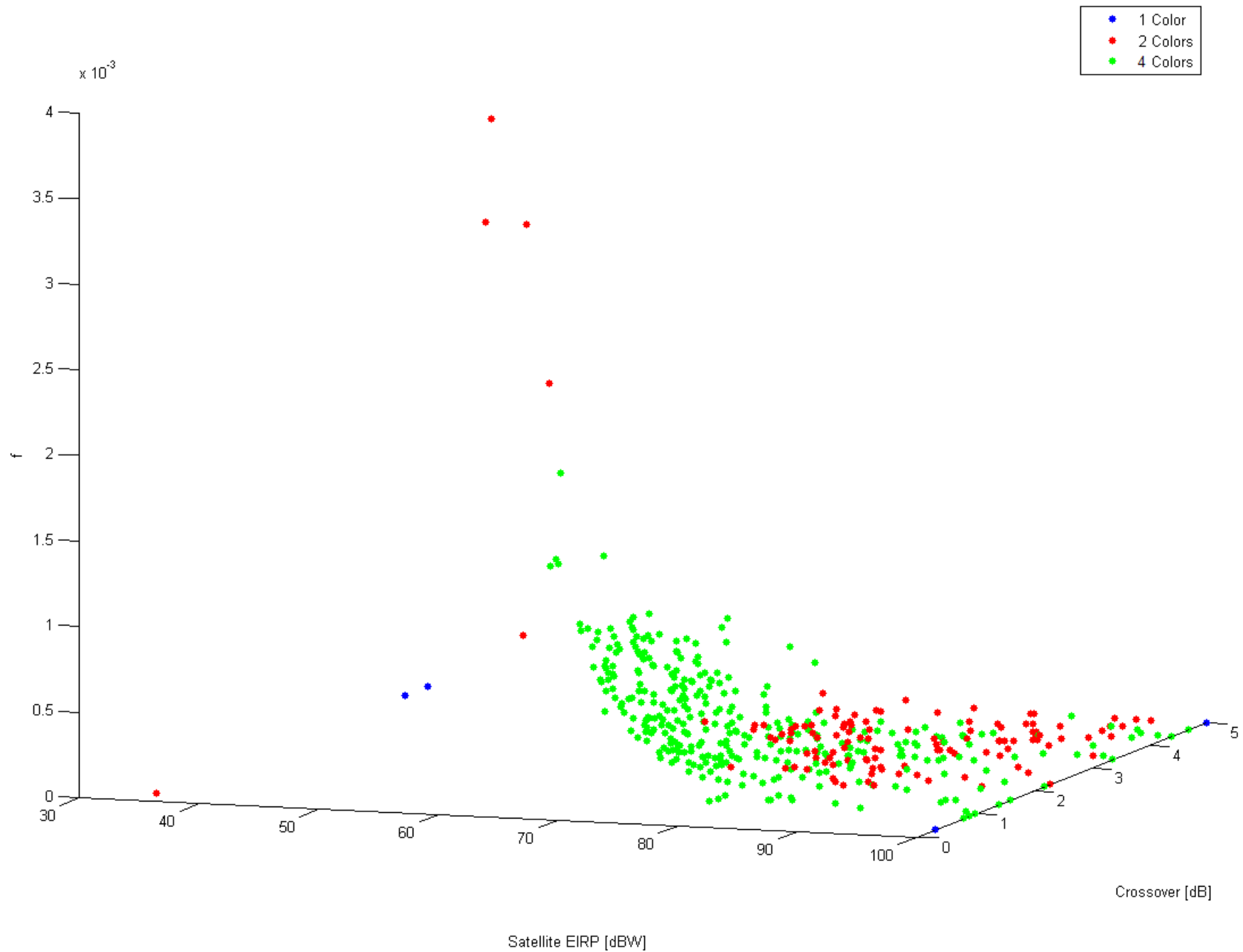


Figure 6.7: Point Cloud of the Third Experiment with the Constraint 'Availability > 99%'

7 Conclusion and Future Work

It's time to take a look back on the goals accomplished, to compare the results demanded and obtained by this thesis.

This thesis aimed at providing an interface for the existing simulation software of Joanneum Research, which was accomplished by integrating additional modules into the existing project.

In section 2, we focused on introducing a small set of optimization algorithms since the objective of this thesis is not to provide complex algorithms, but to show the possibilities of optimization on satellite communication systems, even with very simple ones. We also showed the basic mathematical principles we applied to the algorithms in the software in order to enable the optimization within the software. Some of the basic principles allow for the easier development of upcoming optimization algorithms, like the minimization/maximization trick in equation (2.2). Unfortunately though, there was only time to develop two basic optimization algorithms (Nelder-Mead and PSO) and the implementation of multi objective optimization is still pending.

The related work presented in section 3 already offered a promising prospect of how well optimization can perform on system level optimization on the one hand, and the possibilities of robust algorithms on the other hand. The satellite resource scheduling optimization was one such task, proving that PSO was able to optimize systems that have to satisfy a multitude of constraints.

We took much knowledge from the related work into our design that we introduced in section 4. Along with the link budget and multi-beam scenario, the planned extensions to the software, new information flows and the planned tags were presented. Most of the design was already advanced enough to find its way into the software, especially the whole set of equations used for the link budget, as well as the calculations for the multi-beam scenario. We also used the already existing syntax in the simulator to ensure the easy integration for the users. At its core, the implemented classes retained the relations shown in the class diagram in figure 4.1, although many more classes, relations, methods and variables were added.

Although it was not possible to leave the simulation software completely untouched, the changes were isolated into a few parts of the program. Like shown in section 5.2, we mainly made small alterations to parts in the simulation core, the burst container and the simulation main loop. The new modules, including the optimizer module, the link budget modules and interference modules, were discussed later in section 5.2.4 in great detail, revealing the exact course of events of an interference simulation. In addition to this, many of the implementation features of the software were presented and the chapter concluded with a simplified class diagram in section 5.3, showing the most important aspects and relations

between the new classes.

From the evaluation in section 6, we learned that simple problems up to three variables can be extensively explored to fully analyze the fitness function at hand, like we saw in figure 6.3 and 6.4. We need to keep in mind that the search space was small by the fact alone that some of the variables, like the color, had very narrow bounds.

However, this may be achieved by simpler algorithms that may not use optimization at all, so in the third experiment, we used the gained knowledge and applied it to a more complex example, considering the system costs in ratio to the goodput. We quickly came to the conclusion that the system tries to create cheap systems with minimum specifications to keep the system up. This is due to the fact that in our created target function, the cost of the HPA is by far the dominating factor. We tried to circumvent this problem by introducing a constraint, demanding at least 99% availability. The results showed that PSO is able to handle this constraint well enough, now providing us with a totally different solution, shown in table 6.5. By further increasing the complexity through adding the bandwidth we now receive different results for nearly every run, as we can see in table 6.6. This seems logical, since there are many trade-offs that can be achieved, but there is already a significant difference in the quality of the target function for each of these solutions found.

Nevertheless, if we compare tables 6.5 and 6.6, we see that without the variable bandwidth, we have found a far superior solution. The question at hand is, why are the latter solutions so different, again favoring cheap solutions? We can deduce the answer to some extent from figure 6.7: 4 colors offer better solutions in general, whereas the 2 color case is superior in specific settings. The PSO algorithm will stick to the more useful solutions, found for the 4 color cases and only try and explore the 2 color case once in a while. The good solutions however, demand a high variation of the parameters: We need to increase the bandwidth up to its maximum, add more power to compensate the loss in C/N for 2 colors and increase the crossover-level to decrease the interference. If the algorithm fails to vary only one parameter in the right way, the solution will not be competitive enough, or even become infeasible with respect to the constraint.

This may be a shortcoming of the PSO algorithm, or our chosen parameters: The parameters we used from [25] are tuned towards bounded real-valued problems, whereas we have mixed real-valued and integer problems, where there is a strong dependency between each dimension. Future work should focus on trying more advanced, robust algorithms to prove if this is simply the fault of the algorithm, the cost function we used, or the problem domain at hand.

For the link budget calculation, we used the suggested weather distribution shown in tables 6.2 and 6.3, independently drawing each attenuation from the given CDF in each run. This simplistic weather model serves its purpose when it comes to expectation values, but for the

used ACM feature, we neglect the fact that switching Modcods is not an instantaneous action, but requires time in which we lose some data. This feature however, requires a sophisticated weather model, considering fade slopes during weather events. To compute more realistic simulations, we suggest adding simple fade slopes to the software in the future, where in each simulation step there is a chance to start such event, according to a Markov model.

As we discussed previously, the design of the target function can become a major issue of the optimization, basically defining which solutions the algorithm will provide us with. We consider the target function in equation (6.4) to be sufficient for our experiments, but in order to avoid using constraints like we did, terms for penalizing lower link availability would prove to be useful, since they alter the target function in a way, the particles can better 'understand'.

An additional simple feature planned would ease the use of letting the optimization run multiple times: Currently we need to change the directory and call the program for each run manually. A desired feature would include a simple parameter in the simulation file, or as a startup parameter, which tells the algorithm, how many runs we want and if they are to be parallelized.

Bibliography

- [1] J. Ebert, M. Schmidt, H. Schlemmer, E. Turkyilmaz, S. Kastner-Puschl, and J. R. Castro, “The Alphasat Aldo Paraboni experiment: Fade mitigation techniques in Q/V-band satellite channels, first results,” in *2015 IEEE Aerospace Conference*, 2015, pp. 1–9.
- [2] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.
- [3] J. A. Nelder and R. Mead, “A Simplex Method for Function Minimization,” *Comput. J.*, vol. 7, no. 4, pp. 308–313, Jan. 1965.
- [4] V. Černý, “Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm,” *J. Optim. Theory Appl.*, vol. 45, no. 1, pp. 41–51, Jan. 1985.
- [5] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Cambridge, Mass.: MIT Press, 1998.
- [6] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95*, 1995, pp. 39–43.
- [7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
- [8] J. B. J. R. Perez, “Comparison of Different Heuristic Optimization Methods for Near-Field Antenna Measurements,” *IEEE Trans. on Antennas Propag.*, no. 3, pp. 549 – 555, 2007.
- [9] L. Jian and W. Cheng, “Resource planning and scheduling of payload for satellite with genetic particles swarm optimization,” in *IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, 2008, pp. 199–203.
- [10] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
- [11] N. H. P. N. Suganthan, “Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization,” 2005.

- [12] F. G. Lobo, C. F. Lima, and Z. Michalewicz, *Parameter Setting in Evolutionary Algorithms*, 1st ed. Springer Publishing Company, Incorporated, 2007.
- [13] A. K. Qin, V. L. Huang, and P. N. Suganthan, “Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.
- [14] A. Meloni and M. Murrioni, “On the genetic optimization of APSK constellations for satellite broadcasting,” in *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2014, pp. 1–6.
- [15] S. Y. Yuen, C. K. Chow, and X. Zhang, “Which Algorithm Should I Choose at Any Point of the Search: An Evolutionary Portfolio Approach,” in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA, 2013, pp. 567–574.
- [16] S. Y. Yuen and X. Zhang, “On Composing an (Evolutionary) Algorithm Portfolio,” in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, New York, NY, USA, 2013, pp. 83–84.
- [17] J. A. Vrugt and B. A. Robinson, “Improved evolutionary optimization from genetically adaptive multimethod search,” *Proc. Natl. Acad. Sci.*, vol. 104, no. 3, pp. 708–711, Jan. 2007.
- [18] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces,” *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, Dec. 1997.
- [19] J. J. Grefenstette, “Optimization of Control Parameters for Genetic Algorithms,” *IEEE Trans. Syst. Man Cybern.*, vol. 16, no. 1, pp. 122–128, Jan. 1986.
- [20] M. Meissner, M. Schmuker, and G. Schneider, “Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training,” *BMC Bioinformatics*, vol. 7, no. 1, p. 125, Mar. 2006.
- [21] Gerard Maral, Michel Bousquet, Zhili Sun, “Wiley: Satellite Communications Systems: Systems, Techniques and Technology, 5th Edition.” [Online]. Available: <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470714581.html>. [Accessed: 11-Aug-2015].
- [22] ETSI, “Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part II: DVB-S2

- Extensions (DVB-S2X) Specification (Optional).” 2013.
- [23] K. Plimon and J. Ebert, “System Demonstrator for Advanced Interference Mitigation Techniques in Satellite Networks.”, Aug. 2015.
- [24] B. Y. Q. J. J. Liang, “Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization”, *Tech. Rep. 201212 Comput. Intell. Lab. Zhengzhou Univ. Zhengzhou China*, 2013.
- [25] M. E. H. Pedersen, “Good Parameters for Particle Swarm Optimization.” Hvass Laboratories, 2010.
- [26] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, “Shape Distributions,” *ACM Trans Graph*, vol. 21, no. 4, pp. 807–832, Oct. 2002.
- [27] ETSI, “Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2).” 2012.
- [28] “Pareto efficiency,” 16-Aug-2015. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=676363023. [Accessed: 31-Aug-2015].

Appendix

A. Matlab Evaluation Script

```
function [meta_data, params, fitnesses] = readData(params_file_name, fitness_file_name)
    fileID = fopen(fitness_file_name);
    meta_data = readMetadata(fileID);
    %Read everything else as double
    fitnesses = readFitnesses(fileID, meta_data);

    fclose(fileID);

    fileID = fopen(params_file_name);
    meta_data = readMetadata(fileID);
    %Read everything else as double
    params = readParams(fileID, meta_data);

if meta_data.filehandler_id == 1 || meta_data.filehandler_id == 2
    %Parameters are always stored in a flattened way
    params = reshape(params, size(meta_data.variables, 2), []);
end

    fclose(fileID);
end

function result = readParams(fileID, meta_data)
    %Doubles were saved in a flattened way. We get an (1xn) vector where
    %n = number of function evaluations
    if meta_data.filehandler_id == 1 || meta_data.filehandler_id == 2
        params = fread(fileID, 'double');

    %Results were saved in the raw way, to reconstruct the generation
    %parameters and fitnesses
    elseif meta_data.filehandler_id == 3

        params = [];
        while 1
            generation_length = fread(fileID, 1, 'uint32');
            if feof(fileID)
                break
            end
            %TODO: Preallocate...
            tmp = inf(1, max(size(params, 2), generation_length) * length(meta_data.variables));
            tmp(1:(length(meta_data.variables) * generation_length)) = fread(fileID,
                generation_length * length(meta_data.variables), 'double');
            tmp = reshape(tmp, length(meta_data.variables), []);
```

```

        %Concatenate along 3rd dimension
        params = cat(3, params, tmp);
        %params = [p
    end
else
    error('Unknown file handler with id %d', meta_data.filehandler_id);
end

result = params;
end

function result = readFitnesses(fileID, meta_data)
    %Doubles were saved in a flattened way. We get an (1xn) vector where
    %n = number of function evaluations
    if meta_data.filehandler_id == 1 || meta_data.filehandler_id == 2
        fitnesses = fread(fileID, 'double');

    %Results were saved in the raw way, to reconstruct the generation
    %parameters and fitnesses
    elseif meta_data.filehandler_id == 3
        fitnesses = [];
        while 1
            generation_length = fread(fileID, 1, 'uint32');
            if feof(fileID)
                break
            end
            %TODO: Preallocate...
            tmp = inf(1, max(size(fitnesses, 2), generation_length));
            tmp(1:generation_length) = fread(fileID, generation_length, 'double');
            fitnesses = [fitnesses; tmp];
        end
    else
        error('Unknown file handler with id %d', meta_data.filehandler_id);
    end
    result = fitnesses;
end

function meta_data = readMetadata(fileID)
    version_nr = fread(fileID, 1, 'uint32')
    switch(version_nr)
        case 1 %Preprocessed, non-increasing flat fitness
            meta_data = struct('version_nr', version_nr, 'filehandler_id', [], ...
                'algorithm_name', [], 'minmax', [], 'variables', [], 'constraints', []);

            %meta_data.variables =
            meta_data.constraints = struct('name', {}, 'min', {}, 'max', {});
    end
end

```

```

meta_data.filehandler_id = fread(fileID, 1, 'uint32')
meta_data.algorithm_name = textscan(fileID, '%s', 1, 'whitespace', ", 'Delimiter', ',');

meta_data.minmax = fread(fileID, 1, 'uint8')

meta_data.variables = readVariables(fileID)
meta_data.constraints = readConstraints(fileID);

```

case 2

```

meta_data = struct('version_nr', version_nr, 'filehandler_id', [], ...
'algorithm_name', [], 'objective', [], 'minmax', [], 'variables', [], 'constraints', []);

%meta_data.variables =
meta_data.constraints = struct('name', {}, 'min', {}, 'max', {});

meta_data.filehandler_id = fread(fileID, 1, 'uint32')
meta_data.algorithm_name = textscan(fileID, '%s', 1, 'whitespace', ", 'Delimiter', ',');
meta_data.objective = textscan(fileID, '%s', 1, 'whitespace', ", 'Delimiter', ',');

meta_data.minmax = fread(fileID, 1, 'uint8')

meta_data.variables = readVariables(fileID)
meta_data.constraints = readConstraints(fileID);

```

otherwise

```

error('Unknown version nr: %d', meta_data.version_nr);

```

end

end

```

function constraints = readConstraints(fileID)
nr_constraints = fread(fileID, 1, 'uint32');
constraints = struct('name', {}, 'min', {}, 'max', {}, 'step_size', {});

```

```

for i = 1:nr_constraints
    constraints(i).name = textscan(fileID, '%s', 1, 'whitespace', ", 'Delimiter', ',');
    constraints(i).min = fread(fileID, 1, 'double');
    constraints(i).max = fread(fileID, 1, 'double');

```

end

end

```

function variables = readVariables(fileID)
nr_variables = fread(fileID, 1, 'uint32');
variables = struct('name', {}, 'min', {}, 'max', {}, 'step_size', {});

```

```

for i = 1:nr_variables
    variables(i).name = textscan(fileID, '%s', 1, 'whitespace', ", 'Delimiter', ',');
    variables(i).min = fread(fileID, 1, 'double');
    variables(i).max = fread(fileID, 1, 'double');

```



```
variables(i).step_size = fread(fileID, 1, 'double');  
end  
end
```