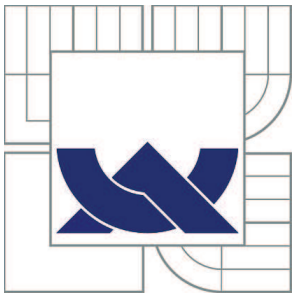

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH INTELIGENTNÍHO OPTIMALIZAČNÍHO MODULU PRO PODNIKOVÝ INFORMAČNÍ SYSTÉM

INTELLIGENT OPTIMIZATION MODULE FOR COMPANY INFORMATION SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. ĽUBOMÍR ZIGO

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ KREJSA, Ph.D.

SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Ľubomír Zigo

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Mechatronika (3906T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Návrh inteligentního optimalizačního modulu pro podnikový informační systém

v anglickém jazyce:

Intelligent optimization module for company information system

Stručná charakteristika problematiky úkolu:

Navrhněte a ověřte algoritmus využívající metody umělé inteligence pro optimalizaci plánů preventivní údržby výrobních závodů. Po úspěšném ověření algoritmu dojde k jeho začlenění do specializovaného průmyslového informačního systému zahraniční společnosti pro řízení a plánování údržby. Dané informační systémy již v současné době obsahují moduly pro sestavování plánů preventivní údržby strojů a výrobních zařízení, ale zatím bez možnosti jejich optimalizace. Při vytváření těchto plánů musí vedoucí pracovníci technických oddělení zohlednit řadu limitujících faktorů a hledat optimální rozložení úkonů preventivní údržby v daném časovém úseku, aby těmto faktorům vyhověli. Mezi takové limitující faktory patří například vyhrazené doby pro pravidelné odstávky strojů, maximální počet disponibilních pracovníků údržby, měsíční nákladové limity na spotřebu materiálu apod. Úkolem je najít vhodný algoritmus, který by zautomatizoval hledání optimálního nastavení plánu a rozšířil tak možnosti existujícího informačního systému.

Cíle diplomové práce:

1. Analyzujte charakter problému z pohledu optimalizačních metod a přístupů
2. Vyberte vhodnou metodu pro řešení této optimalizační úlohy
3. Implementujte daný algoritmus
4. Ověřte algoritmus na konkrétních úlohách z dané praxe a navrhněte doporučení pro další zvyšování jeho účinnosti

Seznam odborné literatury:

Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005

Vedoucí diplomové práce: Ing. Jiří Krejsa, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2011/2012.

V Brně, dne 10.10.2011

L.S.

_____	_____
prof. Ing. Jindřich Petruška, CSc.	prof. RNDr. Miroslav Doupovec, CSc.
Ředitel ústavu	Děkan fakulty

Abstract

This master's thesis deals with design and realization of intelligent optimization module for industrial information system. Optimization module creates all-year maintenance plans with even weekly amount of maintenance capacities and plans for production with regular shutdowns. In the beginning, optimization methods are studied and suitable methods are selected for phase of testing. Non-linear least squares from Matlab Optimization Toolbox, genetic algorithm and artificial bee colony algorithm (ABC) are tested for specified planning problem. Based on test results, ABC algorithm as best method will be implemented in information system. Information system complemented by optimization module enables to create better maintenance plans, what results to increasing overall company's efficiency.

Keywords

Optimization, Maintenance

ZIGO, Ľubomír: *Intelligent optimization module for company information system:* master's thesis. Brno: Brno University of Technology, Faculty of mechanical engineering, Institute of solid mechanics, Mechatronics and Biomechanics, 2012. 74 p. Supervised by Ing. Jiří Krejsa, PhD., SECRET VERSION

Abstrakt

Diplomová práca sa zaoberá návrhom a realizáciou inteligentného optimalizačného modulu pre podnikový informačný systém. Optimalizačný modul bude vytvárať celoročné plány údržby s rovnomerným množstvom údržbárskych aktivít v jednotlivých týždňoch a plány pre výrobu s pravidelnými odstávkami. V úvode je uvedený prehľad použiteľných optimalizačných metód, z ktorých sú vybrané metódy pre ďalšiu fázu, testovanie. Nelineárna metóda najmenších štvorcov z Matlab Optimization Toolbox, genetický algoritmus a ABC algoritmus budú otestované na zadanej plánovacej úlohe. Na základe výsledkov testu bude najlepšia metóda implementovaná v podnikovom informačnom systéme. Takto doplnený informačný systém umožňuje vytvárať kvalitnejšie plány údržby a tým zvýšiť celkovú efektivitu podniku.

Kľúčové slová

Optimalizácia, údržba

Declaration

I declare that I have elaborated my master's thesis on the theme of "O*ptimization module for company information system*" independently, under the supervision of the master's thesis supervisor and with the use of technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, concerning the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal copyright and I am fully aware of the consequences in the case of breaking Regulation S 11 and the following of the Copyright Act No 121/2000 Vol., including the possible consequences of criminal law resulted from Regulation S 152 of Criminal Act No 140/1961 Vol.

Brno …… ……

Author's signature

Acknowledgement

I would like to thank my master's thesis supervisor Ing. Jiří Krejsa, PhD. for supervision and advices during the work on master's thesis. Further, I would like to thank to Ing. Lubomír Sláma PhD., director of Czech division of Act-in company and my colleagues from Act-in office in Brno.

Table of Contents

List of figures

List of tables

Introduction

Information system is part of almost every modern company in these days. It helps company to work more effective and faster. Information system is usually composed of various modules, which take care of company's agenda. Act-in Maintenance Control is one of wide spectrum of information systems. It is focused on maintenance, but it can also handle other areas of production process. Maintenance Control contains module called Planned Maintenance. Planned Maintenance is already able to create all-year maintenance plan automatically, but problem is that, maintenance capacities are not divided evenly in weeks of year. Creating all-year maintenance plans with even weekly amount of maintenance capacities is primary requirement, because only this kind of plan can be effectively completed by internal maintenance capacities of company without hiring external technicians, because weekly working time is not rapidly changing. In contrary, when weekly amount of maintenance capacities is rapidly changing, maintenance technicians have plenty of work in several weeks and no work in other weeks. This is actual state, which is not effective. External technicians need to be hired in several weeks, but there is no work for internal technicians neither in other weeks. Secondary requirement is creating plans for production with regular shutdowns, where in weeks of shutdown amount of maintenance should be higher, to effectively use the time when production is stopped.

This master's thesis deals with design of intelligent optimization module, which can create optimized all-year maintenance plans with even weekly amount of maintenance capacities and also plans for production with regular shutdowns as secondary requirement. Solution will be an implementation of optimization algorithm as general tool of optimization module, which will extend company's information system. Because of wide spectrum of optimization methods, detailed information about them will be necessary. After that, selected methods will be tested on our planning problem and the best method will be implemented in company's information system.

Complete optimization module as part of information system Maintenance Control will become a part of Act-in industrial solutions which are used in Czech Republic, Netherlands, Belgium and Germany in more than one hundred companies.

Master's thesis goals

Intelligent optimization module designed in this master's thesis should create all-year maintenance plans with even weekly amount of maintenance capacities and plans for production with regular shutdowns. Solution consists of four stages:

- **Suitable methods searching**

  There are a lot of optimization methods, which can be theoretically used for planning problems. There are deterministic methods, which can find global optimum, but there can be a problem with long computation time and other limitations. Some of these methods are part of Matlab Optimization Toolbox. In contrary, probabilistic methods cannot find global optimum, but they can provide solution which is good enough in reasonable time. Evolutionary algorithms and swarm intelligence algorithms are probabilistic methods, which became very popular recently.

- **Suitable methods testing**

  Before starting testing of selected methods, our planning problem has to be described by objective function. Then, planning problem can be solved. Non-linear least squares from Matlab Optimization Toolbox were selected as deterministic method. Genetic algorithm and artificial bee colony algorithm were selected from group of probabilistic methods. They will be tested on our planning problem in Matlab environment. Matlab was selected because of its Optimization Toolbox and also good visualization possibilities.

- **Tested methods comparison**

  Selected methods, which were tested on our planning problem, will produce different results. Results will be compared by two criterions; time of computation and quality of solution. According to these two criterions, best method will be selected for implementation in company's information system.

- **Implementation of selected method**

  Best method will be implemented in company's information system Act-in Maintenance Control. Method's code will be translated to VB.NET language in Microsoft Visual Studio to be compatible with rest of code. Design of optimization module will be discussed with company's developers to make implementation simple and compatible with other modules and software structure.

1. Planning problem definition

Primary goal of this master's thesis is to design optimization module, which will be able to create all-year maintenance plan with even weekly amount of maintenance capacities. This means sum of working time spent for maintenance will be almost the same in all the weeks of year, so internal maintenance capacities can be effectively used. Secondary goal is creating of maintenance plans for production with regular shutdowns. Maintenance activities, which should be done regularly, are called *maintenance rules*. Each maintenance rule is defined by three values:

- *Interval* – this value represents number of weeks, when maintenance rule should be repeated. This value is constant.
- *Working hours* – this value represents number of hours needed to complete maintenance rule. This value is constant.
- *Start week* – this value represents week, when maintenance rule is done for the first time. This value is variable and will be changed by optimization algorithm. Minimal value (first possible week to start) is first week of year for all the rules. Maximal value (last possible week to start) is equal to interval value.

Actual state

Actual automatic planning sets start week for all the maintenance rules equal to one and repeats them based on interval value. This means all the rules are planned in first week and next repetitions depends on interval value. It can result to plan, where all work should be done in one week and next weeks have no planned work. Weekly sum of maintenance capacities is rapidly changing as shown in last row of figure 1. This plan is not suitable for practical use, because weekly maintenance capacities are not even.

Maintenance capacities in first and fifth week are very high, so internal technicians could not handle it and external technicians need to be hired. In contrary in second, third or fourth week, there is no work neither for internal technicians, so actual automatic plan has to be manually updated to make it suitable for practical use. Optimization module should provide automatically generated plans with even weekly amount of maintenance capacities and avoid manual updates.

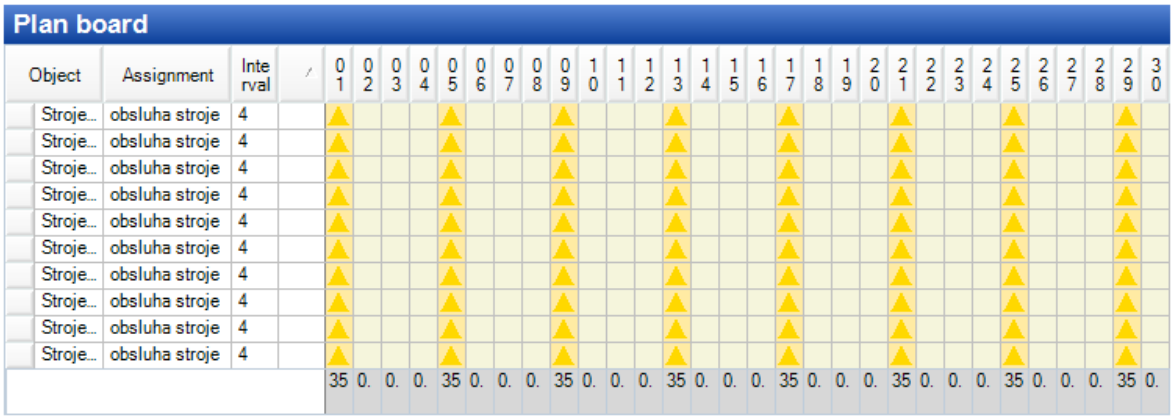

Figure 1: Actual state

Ideal state

Ideal plan should have constant weekly amount of maintenance capacities. This means, working time spent for maintenance rules will be exactly the same in all the weeks. This state represents global optimum. Ideal plan is shown in table 1, where sum of working hours is exactly the same in all the weeks. Creating ideal plan is the best possible solution for our planning problem, but *it is not possible to create it commonly*, because of problem constraints. Following this plan is effective, because working time for technicians is exactly the same in every week. Company can complete plan with its internal capacities and technicians work same time in every week.

	Interval	**Working hours**	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Rule 1	**3**	**2**	2			2			2			2
Rule 2	**3**	**2**		2			2			2		
Rule 3	**3**	**2**			2			2			2	
Total			2	2	2	2	2	2	2	2	2	2

Table 1: Ideal plan

Optimal state

Actual plan shown in figure 1 is not good solution for our planning problem and ideal plan, shown in table 1, cannot be reached commonly. *Optimization module will look for optimal plan*, which should be much better than actual plan and as close as possible to ideal plan. Difference of optimal plan and ideal plan will be represented by *objective value*, which will be described later. Optimization module will not set all start weeks equal to one, but it will set them optimally to create plan with even weekly amount of maintenance capacities.

	Interval	Working hours	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Rule 1	**3**	**1**	1			1			1			1
Rule 2	**3**	**1,5**		1,5			1,5			1,5		
Rule 3	**6**	**2**			2						2	
Total			1	1,5	2	1	1,5	0	1	1,5	2	1

Table 2: Optimal plan

Following optimal plan in table 2 is not as effective as plan in table 1, but it is much more effective than plan in figure 1. Working time is changing, but not so rapidly as in figure 1. There is also one week with no planned work, but this is caused by constraints of this problem. In this plan, maintenance capacities are divided as evenly as possible, so plan can be completed by internal capacities of company, external capacities will not be necessary.

Objective function

Objective function will mathematically describe our planning problem and determines plan quality by its objective value *f*. Quality of plan is defined as distance of current plan and ideal plan. Objective value of ideal plan is equal to zero. To count objective value, we need to know, what ideal state is. To find it, we need to count sum of all the working time for maintenance in year and divide it by number of weeks. This number represents ideal weekly time for maintenance. Square of difference of sum in week and ideal sum in week represents sub-objective value. Sum of all sub-objective values represents objective value (1) or plan quality.

$$f = \sum_{w=1}^{52} (sum(w) - idealWeeklySum)^2 \tag{1}$$

Planning with regular shutdowns

Creating all-year maintenance plans with even weekly amount of maintenance capacity is primary planning problem. Secondary problem is called planning for production with regular shutdowns. This means, there are weeks in year, when part of company does not work, so maintenance capacities should be higher. For example, in every tenth week of year one of production lines is stopped and maintenance capacities should be doubled in this week. Optimization module should handle this requirement and increase maintenance capacities in this week, while keep them even in other weeks.

2. Optimization methods recherché

As long as humankind exists, we strive for perfection in many areas. We want to reach a maximum degree of happiness with the least amount of effort. In our economy, profit and sales must be maximized and costs should be as low as possible. Therefore, optimization is one of the oldest of sciences which even extends into daily life.

If something is important, general, and abstract enough, there is always a mathematical discipline dealing with it. Global optimization is the branch of applied mathematics and numerical analysis that focuses on, well, optimization. The goal of global optimization is to find the best possible elements x from a set X according to a set of criteria $F = \{ f_1, f_2, .. f_n \}$. These criteria are expressed as mathematical functions, the so-called objective functions. [1]*

2.1 Optimization methods overview

The variety of optimization method is very wide today. We are not able to study all of them, so we have to pick only a few and test them on our planning problem. Basically, we can divide optimization methods in two classes: deterministic and probabilistic methods. Deterministic methods represent clear relation between problem definition and its solution. They can find the best solution for defined problem. Best-known member of this group is state space search. Their disadvantage is that they can be applied only in simple problems, where state space is quite small. In complex problems deterministic methods results in a lot of equations, which cannot be solved even on most modern computers of this time.

In these complex problems probabilistic methods comes into play. Studies of these methods started pretty late, only 60 years ago. These methods cannot find global optimum, but their solution is good enough to be accepted. Probabilistic solution is always worse than deterministic solution, but it can be found in reasonable time in compare with global optimum, which can be found in 10^{100} years time.

Both deterministic and probabilistic methods use heuristic problem definition. Heuristic is mathematical representation of problem, which helps to decide quality of possible solutions and it is specific for every problem. Well defined heuristic provides good results in process of optimization. In deterministic methods, heuristic defines processing order of possible solutions. In probabilistic methods, heuristic defines which of possible solutions will be considered in further computations.

Very important group of probabilistic methods is Evolutionary Computation. This group contains all the methods which start with setting multiple initial solutions, usually called population, which are iteratively refined. Most important members are evolutionary algorithms and swarm intelligence. These methods will be described later. There are also another methods copying physical processes like Simulated Annealing, Parallel Tempering and Raindrop method and also methods without any real-world model like Tabu search and Random optimization. [1]

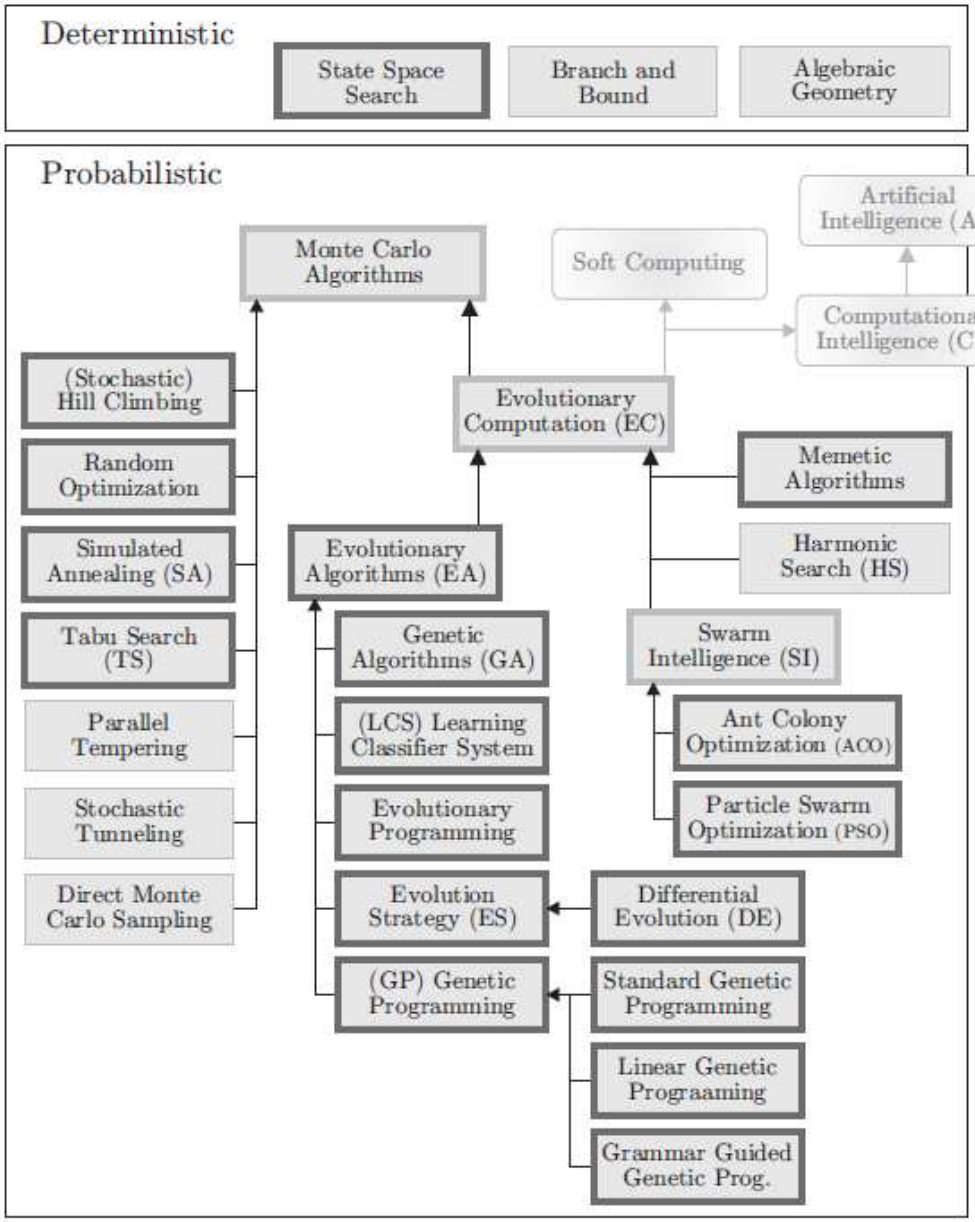

Figure 2: Optimization methods overview [1]

2.2 Structure of optimization task

Although there are many optimization algorithms, base structure of optimization task is common for all of them. This chapter defines structure of optimization task and its notation based on [1].

Set of all possible solutions is called problem space X. Problem space X contains elements x, which represent possible solutions. Problem space usually has logical and practical constraints, which can make it smaller, so the problem is easier to solve. Elements of problem space X are called solution candidates x. In evolutionary algorithms solution candidate is called *phenotype* and problem space is called *phenome*. Union of all solutions is solution space S. Solution space always contains global optima set X^*, but there may exists valid solutions $x \in S$, which are not members of global optima set X^*.

Search space G is a set of elements g which can be inputs to optimization process. In evolutionary algorithms (EA) search space G is also called *genome* and element $g \in G$ is called *genotype*. Units of information in *genotypes* are called *genes*. Value of specific *gene* is called *allele* and position of specified *gene* in *genotype* is called *locus*.

In some problems, the search space G can be identical to problem space X. This means, that elements g of search space G are also elements x of problem space X. For these problems, there is no need to deal with search and selection techniques. Optimization methods like genetic algorithms and swarm intelligence methods can be applied directly on this kind of problem. If there is a difference between search space and problem space, there is a translation required, usually called *mapping*.

Element name	El. sign	Description
Problem space	X	Set of all possible solutions
Solution candidate	x	Possible solution
Solution space	S	Union of all solutions
Global optima set	X*	Union of global optimums
Search space	G	Set of inputs (genome in EA)
Input	g	Element of search space (genotype in EA)

Table 3: Optimization task elements

2.3 Suitable methods selection

There is very wide spectrum of optimization methods available in these days. Present approaches to optimization tasks and especially planning tasks prioritize classic deterministic methods, which can provide global optimum. But there are also modern probabilistic methods, which can be used to solve our planning problem. They usually cannot provide global optimum, but they can provide good enough solution. Usage of these methods will be studied too. These methods represent new approach to optimization tasks and they can probably provide better solution than classic deterministic methods.

Methods of Matlab Optimization Toolbox will be tested from group of deterministic methods. It uses variety of these methods with built-in functions, which are simple to implement. Matlab Optimization Toolbox was recommended by supervisor of this thesis. Genetic algorithm from category of Evolutionary algorithms and Artificial Bee Colony algorithm from category of Swarm Intelligence will be tested as probabilistic methods. Genetic algorithm was recommended by Dr. Lubomír Sláma and Artificial Bee Colony algorithm was recommended by supervisor of this thesis.

2.4 Matlab Optimization Toolbox

Optimization Toolbox provides functions for work with standard and large-scale optimization. Algorithms are able to solve both constrained and unconstrained optimization problems. It is also able to solve both continuous and discrete problems. Toolbox implements linear programming, quadratic programming, nonlinear optimization, nonlinear least squares and multi-objective optimization. These methods can be used for searching optimal solutions, perform various analyses, e.g. parameter estimation of dynamic systems. [2]

Toolbox provides a lot of optimization methods. They can be divided in three groups. Minimization provides methods for searching minimum of specified function by changing its input parameters. There is *fmincon* function [3] for solving constrained nonlinear multi-variable systems. Equation solving group is oriented on solving various systems of linear and nonlinear equations define as $F(x) = 0$. Most important function of this group is *fsolve* function [4]. This function solves system of nonlinear equations. Third group deals with Least squares. There is *lsqnonlin* function [5] which is able to solve nonlinear constrained problems. Their parameters are lower bound and upper bound for optimized parameters, what is necessary in planning tasks.

2.4.1 Minimization

Minimization group provides functions for searching for minimum of specified functions. There are functions for linear or nonlinear, constrained or unconstrained problems. They can be defined in continuous or discrete domain. Our planning problem is nonlinear and also constrained, so there is *fmincon* function for this kind of problem. [3]

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
      fun – objective function
      x0 – initial guess of parameters (ones vector)
      A, b – linear constraint matrices
      Aeq, beq – linear constraint vectors
      lb – lower bound for parameters (ones vector)
      ub – upper bound for parameters (period vector)
```

2.4.2 Least Squares

Least squares group provides functions for data-fitting and other optimization problems. There are linear or nonlinear, constrained or unconstrained functions too. For our nonlinear constrained problem there is *lsqnonlin* function [5]. This function tries to minimize output value of objective function (fun) by changing optimization parameters. Initial guess of parameters should be defined as another parameter. There are no constraint matrices and vectors in compare to *fmincon* function [3], so this function should be more convenient for our problem.

```
x = lsqnonlin(fun,x0,lb,ub)
      fun - objective function
      x0 – initial guess of parameters (ones vector)
      lb – lower bound for parameters (ones vector)
      ub – upper bound for parameters (period vector)
```

2.5 Evolutionary Algorithms

Evolutionary algorithms (EA) are metaheuristic optimization algorithms inspired by nature. EA applies biological operations like mutation, crossover, selection and survival to improve problem specific solution iteratively. They were introduced in late 1960s by Holland and Fogel. [6, 7]

Different theories tried to explain this process. They were based on Darwin's Natural Selection Theory [8] and Mendel's work [9] describing genetic inheritance. Evolution is described as process, which not operates with organism directly. It operates with coded information called chromosomes. Chromosome is information about organism, which is

passed from one generation to next one. Chromosomes are updated with genetic operations to create organism with better properties in next generation.

While evolution is not direct process with unknown final state, evolutionary algorithms are trying to get to final state, which has to be defined. Final state is mostly global optimum, which will not be reached. Goal is to get as close as possible in reasonable period of time. Because of this, there are two approaches in construction of evolutionary algorithms:

- Simulate nature – reproduce natural principles with high accuracy
- Inspire by nature – use principles and adapt them on specific problem

First approach has given rise of optimization methods called Swarm Intelligence [13] or Artificial Life, which will be discussed in next chapter. Second approach is typical for evolutionary algorithms.

Advantage of evolutionary algorithms is their black-box character, because implementation for different problem differs only in objective function, which describes current problem. Objective function has to mark current solution candidate with objective value, which represents quality of current candidate. Well-defined objective function is a key for EA to work, because structure of algorithm remains always the same. This makes implementation for different problems easier. [1]

2.5.1 How EA works

Evolutionary algorithm is stochastic iterative process, which improves solution for task. Algorithm is working with set of possible solution candidates, called *population*. Each member of population consists from one or more *chromosomes*. *Chromosome* represents encoded information, which is modified in iterations of algorithm. *Chromosomes* can be divided to smaller units called *genes*. *Gene* is smallest unit of information and it represents specific value, which is called *allele*. *Allele* can be discrete, numeric or any other value, based on problem coding.

Initial population is created pseudo-randomly based on constraints of task. Each member of initial population is marked with objective (or fitness) value, which represents quality of this solution. Now iterative process can be started. Population is updated iteratively using genetic operators of selection, mutation, crossover and survival. Algorithm is finished when stopping criterion is reached. Stopping criterion can be defined by maximum number of iterations or objective value, which is good enough to be accepted. Evolutionary algorithm scheme is described in next chapter. [1]

2.5.2 Phases of evolutionary algorithm

Structure of EAs is based on principles of evolution. It is common for various applications, which makes implementation and coding very simple. Structure is based on [1]. EA structure is represented in pseudo-code:

1. *Creating Initial Population* - Initial population is first step of EA and is generated pseudo-randomly based on problem specific criteria.

2. *Evaluation and Fitness assignment* - Objective function values are computed for all members of initial population. This value is computed via objective function, which is the most important part of EA. Well designed objective function provides good results. Fitness value is counted based on objective value. This step can be skipped and objective value can be used instead of fitness value.

3. *Sorting* - Population is sorted according to fitness value and best members are selected for creating next generation of population.

4. *Reproduction* - New generation is created via possible operations like selection, crossover and mutation.

5. This process is repeated until *stopping criterion* is reached.

2.5.3 Evolutionary algorithms subclasses

Evolutionary algorithms were introduced in 1960s. In these years variety of scientist started to use them for different problems. As result we know three different evolutionary algorithms models:

- Evolutionary Programming – this group of EA algorithms has origins in work of *Fogel et al* [7]. Evolutionary programming is focused on adaptation of individuals more than on evolution of their genetic information, what results to more abstract view on the process of evolution. EP does not modify genetic information directly, but it modifies behavior. Genetic information is not modified directly, it is modified via behavior.
- Evolutionary Strategies – this group of EA was developed in Germany by Rechenberg and Schwefel. They were originally developed to solve engineering problems. They are most often manipulating large arrays of floating-point numbers. They use standard genetic operation like mutation and recombination.

- Genetic algorithms – this group of EA is most widespread variant. They were introduced by Holland [6] and his work had big influence in later years. Main search tool is recombination, sometimes crossover. Different parts of solutions can be combined to find better solution. Mutation is only background operator, which by importing new information to population.

2.6 Genetic Algorithms

Genetic algorithms (GA) are a subclass of evolutionary algorithms, where elements of search space are binary strings or arrays of various data types. These methods have roots in 1950s, but they were formalized in work of Holland [6] at the University of Michigan in 1960s. Holland introduced GAs as new approach for problem solving. Today, there are a lot of applications in science, research and also in industry. There are various forms of Gas, like human-based genetic algorithms.

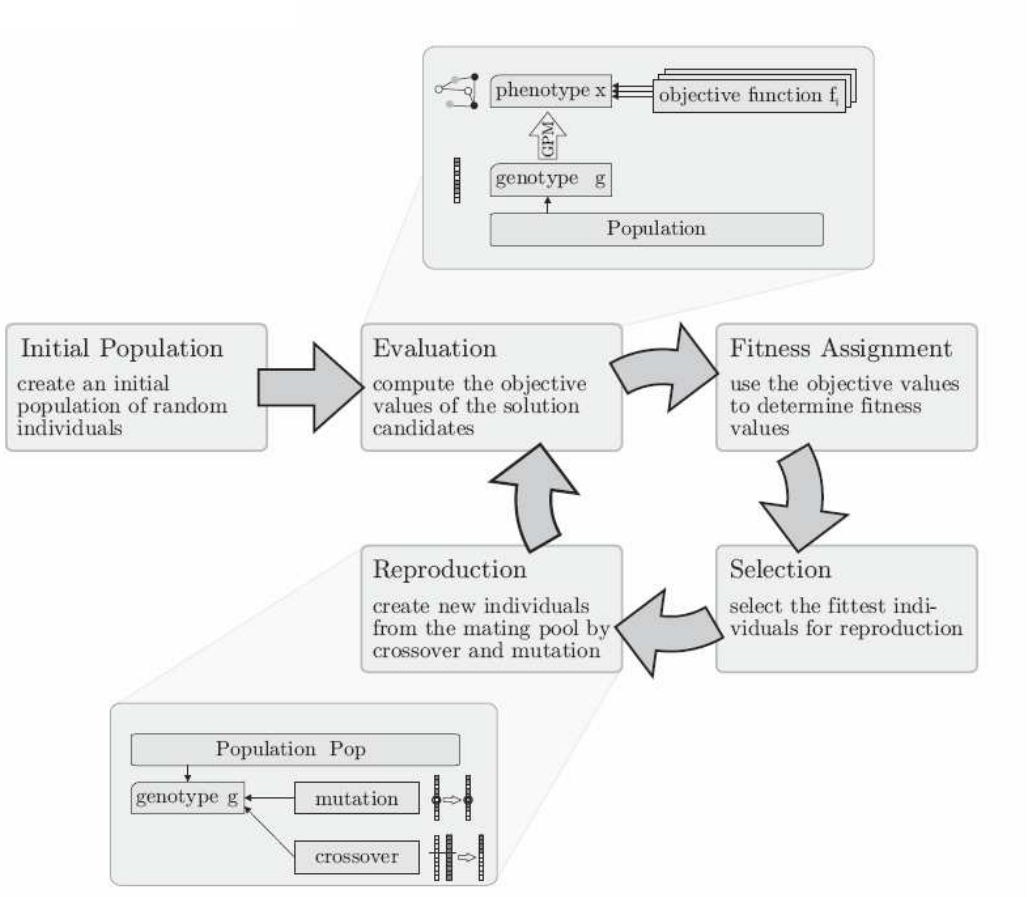

Figure 3: Cycle of genetic algorithm [1]

As subclass of evolutionary algorithms, their structure is very similar. They are specific in kind of coding information of search space. It is usually coded as binary string with 0s and 1s, but also can be coded in different ways. Also character strings or integer strings coding is possible.

2.6.1 Genetic operators

Most important part of GA optimization includes genetic operators. They are involved in process of create new, better population based on current one. Basic operator is called *selection* or *elitism*, which copies best members of current population to next generation. Number of copied members is determined by *elitism* coefficient (usually 5 – 10%).

Next operator is mutation. Mutation makes small changes in solution candidates by random changing of one or more elements. Mutation changes value (*allele*) of selected gene. As shown in figure 4, there are more forms of mutation. Mutation which changes only one gene is called single-gene mutation. Mutation which changes more genes is called multi-gene mutation. There is also special form of mutation, called permutation, which changes two values of genes by changing them. It keeps algorithm from stuck in local optimum.

Crossover, third operator, combines solution candidates (*genotypes*) to create new solution candidates. It can combines two solution candidates by splitting it into two, three or more sections. Members of these sections are chosen to new solution candidate in crossed order. One crossover can result in one or two new solution candidates as shown in figure 5. Crossover is main operator of GAs. [1], [8], [9]

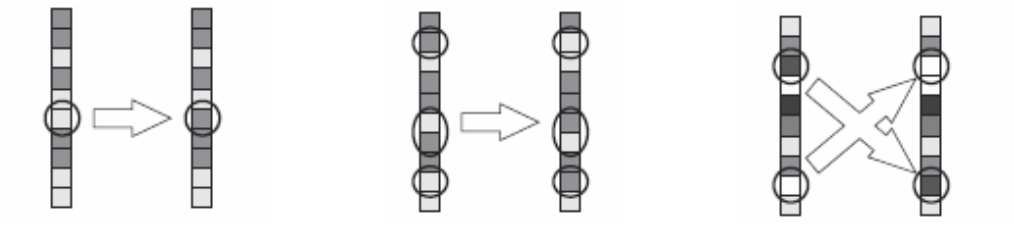

Figure 4: Single-gene mutation, multi-gene mutation and permutation operators [1]

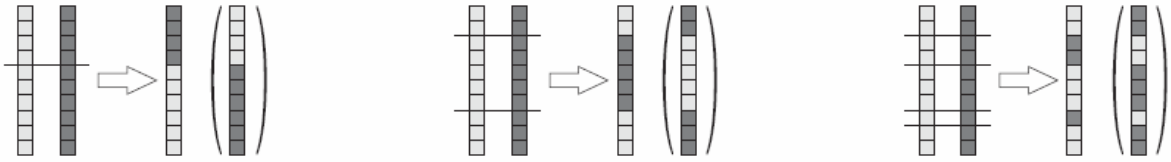

Figure 5: Single-point crossover, tw-point crossover, multi-point crossover [1]

2.6.2 Parameters of genetic algorithm

Influence of genetic operators in process of creating next generation of population is set by following parameters. There is no parameter for crossover operator, because it is applied for all the members of population as main genetic operator.

- *Size of population* – represents number of solution candidates in each generation
- *Maximum number of iterations*
- *Elitism [%]* – number of best solution candidates in generation, which are copied directly to next generation without change
- *Mutation [%]* – number of solution candidates where is applied mutation operation

2.6.3 Genetic algorithm in practical use

Very simple example of GA is Hello World Genetic algorithm [12]. Key is to find "Hello World" phrase based on randomly generated strings. Objective value is defined as number of character which differs from "Hello World" string. Initial population (genome in GAs) is created as set of random strings with length of eleven characters. This is length of "Hello World" string. Members of initial population (genotypes in GAs) are evaluated by objective function what determines number of different characters from goal phrase. For example:

Goal phrase:	"Hello world"	Objective value:	0
Random string:	"Hartbn juhg"		10
	"Hgllo wbrld"		2

Each member of population was evaluated by objective value. Next step is to count fitness value. Fitness value also represents quality of solution as objective value, but it is counted in different way. Fitness value is not necessary in GAs because objective value can be enough, so fitness value can be omitted. Based on objective value (or fitness value) best solution candidates (in GAs phenotypes) are selected for reproduction. Genetic operator (selection, mutation, crossover) are applied on selected members to create next generation of population. Mutation changes character in input string with randomly generated character. Crossover combines two strings to create new one. For example:

Mutation:	"H**u**lle wurld"	->	"H**e**lle wurld"
Crossover:	"**Hulle** wurld"	->	"Hulle morld"
	"Mojje **morld**"		"Mojje wurld"

These operations are applied iteratively until one of members of population is not equal to goal string “Hello world”. Number of iterations can be also limited. There can situation happens, that maximum number of iterations was reached, but goal state was not reached. This is possible because of metaheuristic character of GAs. Because of this, algorithm can be run many times and best solution is saved as result.

2.7 Swarm Intelligence

There is a lot of scientist working in optimization area, whose model and solve various optimization problems using natural principles. These methods inspired by nature are often used for problems, where classic optimization methods came out of play. They are mostly used for large-scale and non-linear problems, where classic algorithms have problems with flexibility, when they are adapted for different kinds of problems. They are limited by type of objective and constraint functions and by type of variables used for problem modeling. They are also dependent of size of solution space, what limits their use for large-scale problems. When we want to get rid of these limitations another algorithms have to be used. In last decades a lot of algorithms were created based on natural principles. Genetic algorithms, simulated annealing and tabu-search are algorithms from this group. Experimented proved that results of these algorithms are far better than classic ones. Another and younger group of nature inspired algorithms is called swarm intelligence (SI) [13]. It was introduced by Gerardo Beni and Jing Wang in 1989 in [14].

Swarm intelligence as branch of optimization is inspired by insect behavior. These algorithms remove limitations of classic optimization methods by providing robust solution for variety of problems with problem specific objective function. They are focused on behavior of insect, which is used for solving of optimization problems with very good results. Implementation and coding of swarm intelligence algorithms are simple and quick for different problems. Algorithms of this group are shown in next figure. Best known algorithms are honey bee algorithms and ant colony algorithms. [14]

Swarm intelligence is defined in [13] as system with many individuals. Each individual interacts with environment and other individuals to gather information. Information is shared and used for improving society’s environment in future.

Figure 6: Swarm intelligence areas [15]

2.7.1 Honey Bees behavior

Insect colonies represent dynamical system gathering information from environment and changing its behavior based on gathered information. Bee system consists of two components:

- Food sources
- Foragers

When we assume, that bees have no information about environment every bee starts as *unemployed forager*. If bee starts searching without any information, it becomes *scout forager*. There are 5% - 30% of scout bees in nest. If bee starts searching based on information from another bee, it becomes *recruit.*

When *recruit* finds food sources, it becomes *employed bee*. *Employed bee* memorizes the location and quality of food source. *Employed bee* takes nectar from food source to hive. There are three possibilities, what can happen now. If nectar level is low, *employed bee* abandons its food source and becomes *unemployed bee* as was on the beginning. If amount of nectar is high it can continue with gathering nectar without sharing information about food source. If food source is very good it can perform *waggle dance* to inform other mates in hive about food source. Other category is *experienced forager*, who uses its historical information about food sources. It can control quality of discovered food source. It can also rediscover food sources after *waggle dance* of another forager. It can also become *scout forager*, if its food source is exhausted and also recruit bee. [16]

2.8 Artificial Bee Colony algorithm

Artificial Bee Colony algorithm [15] is modern metaheuristic algorithm introduced by Turkish professor Dervis Karaboga in 2005. ABC algorithm as member of swarm intelligence algorithms is based on improving initial population by members of swarm. Algorithm is motivated by intelligent behavior of honey bees while foraging food. It is part of Bee Colony optimization and uses only simple control parameters like bee colony size and maximum limit of iterations. ABC is an iterative population based algorithm, where individuals called food

sources are modified by bees. Bees try to discover food sources with higher amount of nectar. Finally best source is found and this one represents result of algorithm. Bees are divided into three groups:

- Employed bees
- Onlooker bees
- Scout bees

Employed bees and onlooker bees are exploring its multidimensional search space. They use their own experience and also experience of their mates, while looking for food. Scout bees does not use any previous experience, they just fly and test whether new source is better than previous one. ABC combines these local search methods with global search methods, what prevents from stuck in local optimum and reaches better results than other similar methods. Algorithms from bee colony optimization family are very successful with scheduling problems. Each solution is represented by food source. Food source is modified iteratively by employed and onlooker bees. They are trying to improve way how to get to this food source, while minimizing cost of this way. Food source and depending solution with lowest cost is optimal.

Homepage of ABC [15] provides documentation and also source code for algorithm. There is detailed pseudo-code and also sample problem solution available. Source code is available in several programming languages like Matlab, C and also Delphi. This source code will be used and updated for our problem in chapter about testing ABC algorithm (chapter 6).

2.8.1 Parameters of ABC algorithm

- *Colony size* - number of bees used by algorithm. There are two groups of bees, employed bees and onlooker bees and colony size is the sum of both. These two groups will be described in next chapters.
- *Food number* - number of food sources, which is equal to half of colony size.
- *Limit* - maximum number of tries to improve food source, after reaching limit value, food source is abandoned.
- *Maximum number of iterations.*
- *Objective function* – defines problem
- *Lower bound of parameters* – defines lower bound for parameters
- *Upper bound of parameters* - defines upper bound for parameters

2.8.2 ABC pseudo code

1. Initialize population $x_{i,j}$
   Food sources are initialized in range of bound of solved problem

2. Evaluate population
   Food sources are evaluated by objective function

3. While not stopping criterion

4. Produce new solutions $\theta_{i,j}$ in neighborhood of current solutions using formula (2), where k is a solution in neighborhood of i, Φ is random number in range [-1,1] and evaluate them.

$$\theta_{i,j} = x_{i,j} + \emptyset_{i,j}(x_{i,j} - x_{k,j}) \tag{2}$$

5. Apply greedy selection between current and new food positions
   Decide whether new food sources have higher amount of nectar than previous solutions.

6. Calculate probabilities P_i according formula (3) for solutions x_i according their fitness values (2).

$$fit_i = \begin{cases} \frac{1}{1+f_i} & if\ f_i \geq 0 \\ 1 + abs(f_i) & if\ f_i < 0 \end{cases} \tag{3}$$

$$P = \frac{fit_i}{\sum_{i=1}^{n} fit_i} \tag{4}$$

7. Produce new solutions θ_i for onlookers from solutions x_i depending on P_i and evaluate them by objective function.

8. Apply greedy selection for onlookers solutions
   Decide whether new food sources have higher amount of nectar than previous solutions.

9. Check scout bees.
   Determine whether some solutions should be abandoned and replaced with new random solution. If solution could not be improved in limit-iterations it will be abandoned and replaced.

10. Memorize best food source

11. Return to step 4 until stopping criterion is reached.

Based on [15].

3. Defining objective function

Optimization methods were described in previous chapter. Now, we need to define objective function. Objective function describes problem mathematically and determines quality of current solution candidate by objective value. It is defined as square of distance of current solution candidate from global optimum. This means objective value of global optimum is equal to zero. Now, we need to define objective function and specify input parameters. Output parameter is objective value, quality of current solution candidate.

In our planning problem, objective function determines quality of created plan. Plan is based on vector of start weeks of maintenance rules, which is the parameter for optimization. Start week's minimal value is equal to one; maximal value is equal to interval value for each maintenance rule. For each week sum of working time is counted and also total sum of working time in year is counted. When year total sum is divided by number of weeks in year (usually 52), optimal weekly working time is known. Square of difference of actual value in week and optimal value in week is counted as sub-objective value. Sum of these sub-objective values for all the weeks of year represents objective value (5) of current solution candidate.

$$f = \sum_{w=1}^{52} (sum(w) - idealWeeklySum)^2 \tag{5}$$

3.1 Input and output parameters

Each maintenance rule is defined by four values. These four values are used to create plan, count objective value and other necessary optimization tasks.

- Period, interval (in weeks)
- Time spent (in minutes)
- Flexibility (in weeks)
- Start week (in weeks, parameter for optimization)

First three of them (period, time spent and flexibility) are defined globally because they are constant. Start week value is changing in process of optimization, so this value is set as input parameter for objective function. Output parameter is objective value.

3.2 How to create plan

Creating of plan is first step of objective function. Plan will be represented by two-dimensional integer array, initialized by zero values. According to start week and interval value, plan will be filled with working time values as shown in figure 7. Based on input parameters, plan is created separately for each maintenance rule. This is represented by *WHILE* cycle for each rule. Plan is generated for whole year (starting in first week, ends in 52nd week). This is done by code below.

MATLAB Code:

```
for i = 1:ActivitiesCount
  j = GAStart(i,k);
  Plan(i,j) = Cas(i,1);
  while (j < (WeeksOfYear + 1 - Perioda(i,1)))
    j = j + Perioda(i,1);
    Plan(i,j) = Cas(i,1);
  end
end
```

Sample values:

```
ActivitiesCount = 8;                    % number of activities
Period = [4;2;12;3;2;4;6;6];            % period [weeks]
TimeSpent = [1;2;5;1;3;10;15;4];        % time [minutes]
StartWeek = [2;1;1;1;1;4;2;6];          % start week [weeks]
```

Output (8 activities, first 25 weeks):

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.	19.	20.	21.	22.	23.	24.	25.
U 01	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
U 02	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
U 03	5	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	5
U 04	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
U 05	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3
U 06	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0
U 07	0	15	0	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0
U 08	0	0	0	0	0	4	0	0	0	0	0	4	0	0	0	0	0	4	0	0	0	0	0	4	0

Figure 7: Plan

3.3 How to count sum

Now, we have plan based on input parameters (start week) for all the maintenance rules. In next stage, we need to count sum of working time spent for maintenance for each week and total sum of working time spent for maintenance in year. Plan is extended by one row as shown in figure 8, which saves sum value for each week. There is also new variable added – *TotalSum*. This variable saves summary working time for maintenance in year.

MATLAB Code:

```
for i = 1:WeeksOfYear
  for j = 1: ActivitiesCount
    Sum(1,i) = Sum(1,i) + Plan(j,i);
  end
TotalSum = TotalSum + Sum(1,i);
End
```

Output (8 activities, 25 weeks):

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.	19.	20.	21.	22.	23.	24.	25.
U 01	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
U 02	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
U 03	5	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	5
U 04	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
U 05	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3
U 06	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0	0	0	10	0
U 07	0	15	0	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0	15	0	0	0	0	0
U 08	0	0	0	0	0	4	0	0	0	0	0	4	0	0	0	0	0	4	0	0	0	0	0	4	0
Suma	11	16	5	11	5	5	6	25	5	2	5	14	11	16	5	11	5	5	6	25	5	2	5	14	11

Figure 8: Plan with sum

3.4 How to count objective value

Total sum of working time spent for maintenance was counted in previous step. When we divide this number by number of week in year (usually 52), we get optimal weekly sum of working time. Then, we can count sub-objective value for each week, which is square of difference of optimal sum and current sum. Objective value is a sum of sub-objective values for all the weeks.

MATLAB Code:

```
OptimalWeekSum = TotalSum / WeeksOfYear;
for i = 1 : WeeksOfYear
  SubObjectiveValue (1,i) = (OptimalWeekSum – Sum (1,i)) ^ 2;
  ObjectiveValue(1,k) = ObjectiveValue (1,k) + SubObjectiveValue (1,i);
end
```

Objective value is returned by objective function and helps to decide, whether is this solution better than other or not. Well defined objective function is main goal to reach good results in whole optimization process. This objective value is design for all-year planning. In next chapter objective value for planning with regular shutdowns will be described.

3.5 How to count objective value for planning with regular shutdowns

Objective function for planning with regular shutdowns is extended version of objective function for all-year planning. It is defined by two value *rsStartWeek* and *rsInterval*. These values represents week with first shutdown and interval, when shutdown repeats. According these values, vector of optimal weekly sum is counted by code below. This vector is used to count objective value, instead of single value used in previous chapter.

MATLAB Code:

```
rsStartWeek = 2
rsInterval = 6
upperCoeficient = 2
lowerCoeficient = 2 / (rsInterval - 1)
for i = 0 : WeeksOfYear
  optimalSumVector(i) = yearSum / 52 * lowerCoeficient
end
maxCount = rsStartWeek - 1
while maxCount < WeeksOfYear + 1
  optimalSumVector(maxCount) = yearSum / 52 * upperCoeficient
  maxCount += rsInterval
end
for i = 1: WeeksOfYear
  SubObjectiveValue (1,i) = (optimalSumVector (i) - Sum(1,i))^2;
  ObjectiveValue (1,k) = ObjectiveValue (1,k) + SubObjectiveValue (1,i);
end
```

4. Finding Global optimum

Global optimum is best possible solution for our problem. Finding global optimum for each plan should be best solution, but it is very hardly limited by dimension of problem. Process of finding global optimum needs to generate all possible solutions and count their objective values until solution with objective value equal to zero is found. This looked as easy way, but we found out problems very soon. Global optimum for ten activities can be solved in three minutes on average computer. Because standard plans contain hundreds of maintenance rules, time of computation will be very long. One of primary requirements for optimization module was ability to create plan in real time, so finding global optimum will be replaced by finding optimal solution using optimization algorithm in reasonable time. Because of computation time problems, global optimum will be counted only for small numbers of activities (8 - 12) and it will help us to decide which of following algorithms are suitable.

Global optimum pseudo code:

1. *Determine number of all possible solution candidates and generate them*
2. *Count objective value for each solution candidate*
3. *Stop when solution with zero objective value was found*

In next chapters, algorithms will be tested with same input parameters. Also global optimum will be counted for these input parameters. Test results will be compared with global optimum to help us decide, which algorithm provides best results. Algorithms will be compared by objective value of solution and also by time of computation.

5. Matlab Optimization Toolbox testing

Matlab Optimization Toolbox provides variety of methods for solving optimization problems. Based on study from chapter 2, there is function *lsqnonlin* [5], which suits directly on our planning problem. This function represents non linear least squares. Implementation of this method does need only objective function (*fun*), initial solution estimation (*x0*) and bounds for optimized parameters (*lb, ub*). There is also possibility to set options (*opt*).

Lsqnonlin syntax:

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,opt)
```

Because objective function was defined in chapter 3, there is no problem to test *lsqnonlin* function. Initial solution estimation can be set to *ones* vector. *Lower bound* of parameters will be set as *ones* vector too, because first week is first possible week to start planning. *Upper bound* will be equal to *interval* vector, because this value represents last possible week to start planning.

Lsqnonlin implementation:

```
optStartWeek = lsqnonlin(@planning, estStartWeek, lb, ub);
```

This implementation of *lsqnonlin* works, but does not solve our problem. Algorithm deals with start week parameter as real number, not integer number as we need. To specify start week parameter as integer value we need to use *options*. There are several options, which can be set. There are two algorithms to use – *trust region reflective* algorithm or *Levenberg – Marquardt* algorithm. Because our problem has bounds, only *trust region reflective* algorithm can be used. More important option for our problem is minimal and maximal change in optimized parameters. Because we want to work only with integer values, minimal change will be set to one. Maximal change will not be defined.

```
options = optimset('DiffMinChange',1);
```

Now we have objective function, initial solution estimation, lower and upper bounds and also minimal change in optimized parameters defined. *Lsqnonlin* function starts optimization and in first steps parameters are changing by minimal value equal to one. But in next iterations algorithm changes values not only in integer value, it comes to real values again. After few attempts in setting maximal change in parameters and other possible options, *lsqnonlin* function was not able to solve our problem.

6. Genetic algorithm testing

Next method for testing is genetic algorithm. Genetic algorithms are used in wide spectrum of optimization problems. They are inspired in nature, where each generation of population is trying to be better than ancestors. This principle is used in optimization to create better solution in each generation of population. Genetic algorithm is iterative process.

6.1 Parameters of genetic algorithm

On the beginning of genetic algorithm we need to define specific parameters:

- Size of population – represents number of solution candidates in each generation
- Maximum number of iterations
- Elitism [%] – number of best solution candidates in generation, which are copied directly to next generation without change
- Mutation [%] – number of solution candidates where is applied mutation operation

MATLAB Code:

```
PopSize = 200;
MaxIter = 100;
Elit = 0.05;
Mutat = 0.1;
```

6.2 Data initialization

In phase of testing we will use pseudo-randomly generated data. Now we need two vectors:

- Time spent vector - generated in range 1 - 15
- Period vector - generated in range 1 - 15

MATLAB Code:

```
for i = 1: ActivitiesCount
    TimeSpent(i,1) = round(rand() * 15) + 1;
    Period(i,1) = round(rand() * 15) + 1;
end
```

6.3 Variables initialization

To count and save all the necessary values we need global variables, which have to be initialized on the beginning of genetic algorithm. Variables are defined in necessary dimensions for holding all the necessary data.

MATLAB Code:

```
%% Variables
Plan = zeros(ActivitiesCount, WeeksOfYear);          % plan for current generation
Sum = zeros(1, WeeksOfYear);                         % sum in weeks [h]
SubObjectiveValue = zeros(1, WeeksOfYear);           % obj. value in weeks [h^2]
TotalSum = 0;                                        % total sum in year [h]
ObjectiveValue = zeros(1,PopSize);                   % obj. value
buffer = zeros(ActivitiesCount + 1,PopSize);         % plan for next generation
```

6.4 Initial population

First step of genetic algorithm is creating initial population. Initial population is generated pseudo-randomly. This means, each run of algorithm generates different initial population, which produces different results. If initial population is generated "well", solution will be "well" too. This means that solution depends on initial population. Because of this, algorithm can be run more times and best solution is accepted.

MATLAB Code:

```
for i = 1:PopSize
  for j = 1: ActivitiesCount
    GAStart(j,i) = round(rand()*(Period(j,1)-1)+1);
  end
end
```

6.5 Genetic algorithm cycle

Genetic algorithm is an iterative process, which tries to improve initial population. Objective value has to be counted in the beginning for each solution candidate in current population. Members of population are sorted based on objective value ascending. Then genetic operators are applied on members of population. Specified percent of best solution candidates are transferred directly to next generation based on elitism parameter. For the rest of current population crossover and mutation are applied. This phase is described in next chapter. Last step is to set newly generated population as current population for the next iteration of algorithm.

MATLAB Code:

```
for i = 1:MaxIter
  GA = CountObjectiveValue(GAStart);
  GAsort = SortObjectiveValue (GA);
  str = ['(',num2str(i),') ', num2str(GAsort(ActivitiesCount +1,1))];
  disp(str);
  GAvysl = GeneticOperations(GAsort, buffer);
  GAStart = GAvysl(1:Limit,:);
end
```

6.6 Genetic operators

Genetic operators are the core of genetic algorithm and they are used to create next generation based on current generation. Basic genetic operators are elitism, crossover and mutation. There are parameters of GA, which control influence of each operator in creating next generation. Main operator is usually crossover combined with small amount of mutation. Elitism coefficient is usually set to very small value, because only a few of best individuals should be transferred directly. Next generation consists from three groups.

- *Elitism group* – best solution candidates from previous generation are transferred directly without any change.
- *Crossover and mutation group* – rest of first half of next generation is created from previous generation using crossover and mutation
- *New group* – second half of new generation is generated pseudo-randomly as initial population

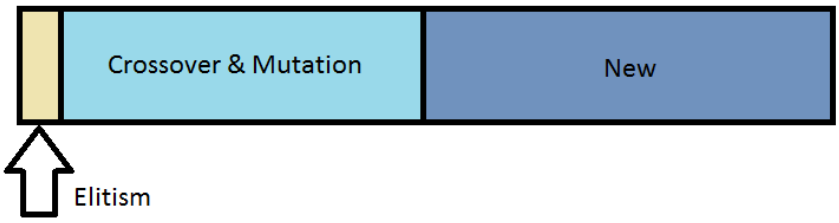

Figure 9: Next generation

MATLAB Code:

```
function buffer = GeneticOperations ( GA, buffer )
  % elitism
  buffer(:,1:PopSize*Elit) = GA(:,1:PopSize*Elit);
  for i = (PopSize*Elit+1):PopSize/2
    i1 = round(rand()*((PopSize/2)-1))+1;
    i2 = round(rand()*((PopSize/2)-1))+1;
    cpos = round(rand()* ActivitiesCount)+1;  % index of crossover edge
    mpos = round(rand()* ActivitiesCount)+1;  % index of mutated element
    % crossover
    buffer(1: ActivitiesCount,i) = [ GA(1:cpos,i1); GA((cpos+1):
ActivitiesCount,i2)];
    % mutation
    if (rand() < Mutat)
      buffer(mpos,i) = round(rand()*(Period(mpos,1)-1)+1);
    end
  end
  % new group
  for i = (PopSize/2+1):PopSize
    for j = 1: ActivitiesCount
      buffer(j,i) = round(rand()*(Period(j,1)-1)+1);
    end
  end
end
```

6.7 Results presentation

There are two ways for presenting results. First is command window of Matlab environment, where text form of result is displayed. Information about current iteration index and current objective value is displayed each iteration. Second output is chart presentation. First chart compares best solution candidate from initial population and best solution candidate from final population. Second chart displays progress of objective value in optimization process. Parameters of genetic algorithm are displayed in header of the chart.

Text output and chart output (100 activities):

```
(1) 23360.9808
(2) 18897.0769
(3) 18897.0769
(4) 18897.0769
(5) 17082.0577
```

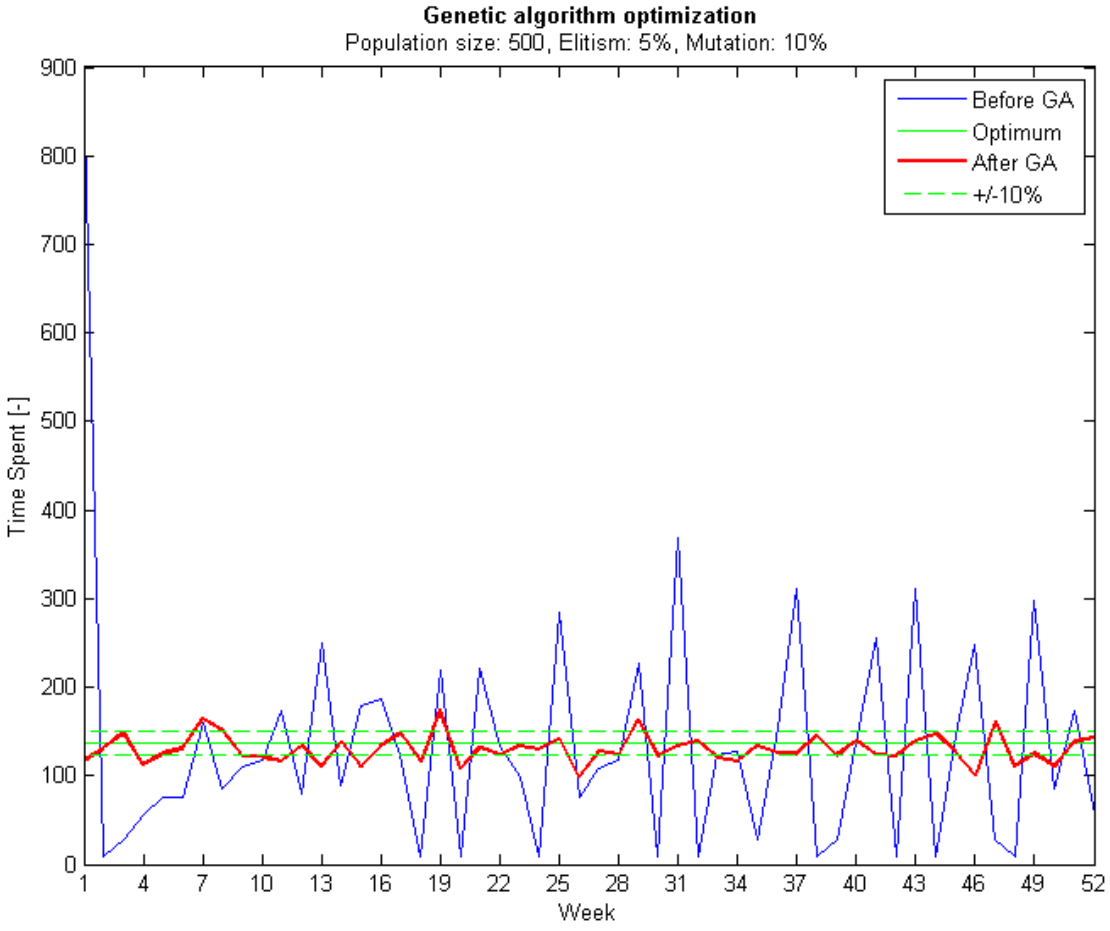

Figure 10: Genetic algorithm result

7. ABC algorithm testing

ABC algorithm is member of Swarm Intelligence and it was defined by Prof. Dervis Karaboga in 2005 [15, 16]. Algorithm requires problem specific objective function, which is defined as parameter of algorithm. Objective value returned by objective function is minimized by algorithm iteratively. Basically is implementation of ABC algorithm very simple, because you need only define your problem specific objective function to use it.

7.1 Parameters of ABC algorithm

- *Number of maintenance rules*
- *Colony size* - number of bees used by algorithm. There are two groups of bees, employed bees and onlooker bees and colony size is the sum of both. These two groups will be described in next chapters.
- *Food number* - number of food sources, which is equal to half of colony size.
- *Limit* - maximum number of tries to improve food source, after reaching limit value, food source is abandoned.
- *Maximum number of iterations.*
- *Objective function* – defines problem
- *Lower bound of parameters* – defines lower bound for parameters
- *Upper bound of parameters* - defines upper bound for parameters

MATLAB Code:

```
ActivityCount = 100;       % number of activities
NP = 100;                  % colony size (employed+onlooker bees)
FoodNumber = NP/2;         % number of food sources
limit = 10;                % limit value to abandon food source
maxCycle = 50;             % max number of cycles
objfun = 'planning';       % objective function
lb = ones(1, ActivityCount);  % lower bound
ub = Period;               % upper bound
```

7.2 Food sources initialization

Food sources represent solution candidates in ABC. All food sources are initialized on the beginning of algorithm. They are initialized in range of lower and upper bound, defined above. After initialization algorithm can be started. Objective value for all the food sources is counted. Food source with best (minimal) objective value is memorized. After that, ABC cycle can be started. Food sources are visited by bees, which tries to improve them proportionally according to its quality represented by amount of nectar.

MATLAB Code:

```
Range = repmat((ub-lb),[FoodNumber 1]);       % replicate and tile array
Lower = repmat(lb, [FoodNumber 1]);
Foods = round(rand(FoodNumber,D) .* Range + Lower);
ObjVal = feval(objfun,Foods);                 % count objective values
BestInd = find(ObjVal == min(ObjVal));        % save best food source
BestInd = BestInd(end);
GlobalMin = ObjVal(BestInd);
GlobalParams = Foods(BestInd,:);
```

7.3 ABC cycle

ABC cycle is iterative process, which tries to improve initial food sources. Bees as members of hive are exploring their environment and collect information based on their own experience and experience of their mates. According to this information food sources are updated by visitations of bees in three phases. There is employed bee phase, onlooker phase and scout bee phase. Each of these phases improves food sources in different way. These phases of ABC algorithm are described in next chapters.

Pseudo-code:

```
while (iter <= maxCycle)
   for (i = 1:FoodNumber)
   % Employed Bee Phase
   % Onlooker Bee Phase
   % Scout Bee Phase
   end;
end;
```

7.4 Employed Bee Phase

In employed bee phase is each food source modified by its employed bee. Food source is represented by vector of integer values. One of these values is randomly selected for modification. This value is modified in the range specified by lower and upper bound (*lb*, *ub*). Objective value for new solution is counted. If new solution is better than current solution, current solution is replaced by new solution, else trial counter is increased by one.

MATLAB Code:

```
for i = 1:FoodNumber
  Param2Change = fix(rand * ActivityCount) + 1;
  sol = Foods(i,:);
  sol(Param2Change) = fix(rand()*ub(Param2Change) + 1);
  ObjValSol = feval(objfun,sol);
  if (ObjValSol < ObjVal(i))
    Foods(i,:) = sol;
    ObjVal(i) = ObjValSol;
  else
    trial(i) = trial(i) + 1;
  end;
end;
```

7.5 Onlooker bee phase

Onlooker bee phase is similar to employed bee phase but solutions to change are chosen with probability, which is proportional to its quality. Probability can be counted with many formulas, current formula was chosen after testing various possibilities.

MATLAB Code:

```
i=1; t=0;
while(t < FoodNumber)
  if(rand() < prob(i))
    t = t + 1;
    Param2Change = fix(rand * ActivityCount)+1;
    sol = Foods(i,:);
    sol(Param2Change) = fix(rand()*ub(Param2Change) + 1);
    ObjValSol = feval(objfun,sol);
    if (ObjValSol < ObjVal(i))
      Foods(i,:) = sol;
      ObjVal(i) = ObjValSol;
      trial(i) = 0;
    else
      trial(i) = trial(i) + 1;
    end;
  end;
  i = i + 1;
  if (i == FoodNumber + 1)
    i = 1;
  end;
end;
```

7.6 Scout Bee phase

Scout bee phase is performed by only one scout bee in each cycle of algorithm. Scout bee looks for food source with maximal value of trial counter. If trial counter exceeds limit value, food source is abandoned and entirely new food source is created. Objective value of new food source is memorized. Scout bee removes inconvenient food sources and looks for new food sources with higher amount of nectar. Scout bee works without any information from other bees (employed and onlooker), so it brings random influence to prevent stuck in local optimum.

MATLAB Code:

```
ind = find(trial == max(trial));
ind = ind(end);
if (trial(ind) > limit)
  sol = round((ub-lb) .* rand(1,D) + lb);
  ObjValSol = feval(objfun,sol,1,D,ub,Cas);
  Foods(ind,:) = sol;
  ObjVal(ind) = ObjValSol;
end;
```

7.7 Results presentation

There are two ways for presenting results. First option is command window of Matlab environment, where text form of result is displayed. Information about current iteration index and current minimum of objective value is displayed each iteration. Second option is chart presentation. There are two charts; first chart compares best solution candidate from final population with theoretical optimum. Second chart displays progress of objective value in optimization process. Parameters of ABC algorithm are displayed in header of both charts.

Text output and chart output (100 activities):

```
(1) 23360.9808
(2) 18897.0769
(3) 18897.0769
(4) 18897.0769
(5) 17082.0577
```

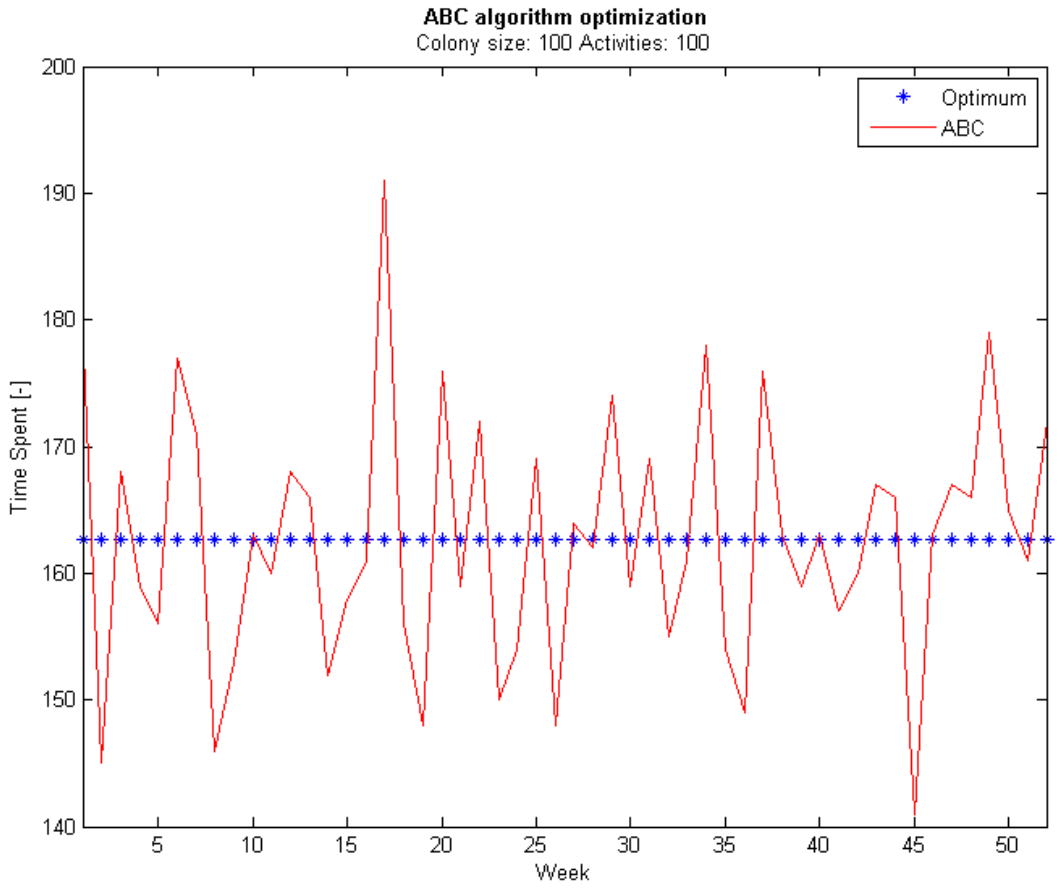

Figure 11: ABC algorithm result

8. Methods comparison

Three methods were tested in previous chapters. Matlab Optimization Toolbox was not able to solve this kind of problem, because planning problem was defined in integer values, but *lsqnonlin* function always solves the problem in real values. Genetic algorithm and ABC algorithm produced very good results. These two methods will be compared with same input values in this chapter. Quality of solution and time of computation will be compared.

8.1 Time of computation

Time of computation depends on number of maintenance rules. When number of rules is equal for both algorithms, differences in elapsed time are not significant. Results are shown in table 4. Elapsed time in chart is an average value for twenty runs of each algorithm.

Rules	Iterations	Population size / Colony size	Elapsed time [s]	
			GA	ABC
100	100	200	11	13
100	200	200	24	26
200	100	200	21	23
200	200	200	42	48
200	300	200	57	66
500	100	200	38	38
500	200	200	77	79
1000	100	200	75	77
1000	200	200	152	173
1000	300	200	235	220

Table 4: Elapsed time comparison

8.2 Quality of solution

Differences in elapsed time were not important. Now we try to compare results of both algorithms with global optimum. Finding global optimum is limited by number of maintenance rules and it was possible only up to ten rules. As shown in table 5, both algorithms are able to reach global optimum, but not in every run, what is caused by its metaheuristic character – initial population is created pseudo-randomly. For higher counts of maintenance rules we were not able to compare solution with global optimum. As shown in table 6, differences in quality of solution produced with these two algorithms are very significant. ABC algorithm produced better solutions for each tested number of maintenance rules. Differences are significant in full range of tested values. Objective value in chart is an

average value for twenty runs of each algorithm. While genetic algorithm has big problem with stuck in local optimum, ABC algorithm does not because of its scout bee phase. Scout bee phase of ABC algorithm prevents from stuck in local optimum by removing abandoned food sources and involving new food sources to algorithm. Genetic algorithm does not have this option, what results to worse results. This is shown in table 6, where we can see that increasing number of iterations results to improving solution using ABC algorithm, but not improving solution using genetic algorithm.

Rules	**Global optimum**	**Genetic Algorithm**					**ABC Algorithm**				
	Obj. value	Obj. value					Obj. value				
	-	1	2	3	4	5	1	2	3	4	5
8	**1265**	1274	1274	**1265**	**1265**	**1265**	**1265**	**1265**	**1265**	1274	**1265**
9	**1257**	**1266**	**1268**	**1257**	**1259**	**1266**	**1266**	**1268**	**1266**	1279	1277
10	**1076**	**1077**	**1082**	**1084**	**1077**	**1078**	1098	1120	**1076**	**1083**	**1082**

Table 5: Objective value comparison 1

Rules	**Iterations**	**Population size / Colony size**	**Obj. value**	
			GA	**ABC**
100	100	200	1936	**511**
100	200	200	2008	**398**
200	100	200	4961	**971**
200	200	200	5065	**638**
200	300	200	4340	**460**
500	100	200	17508	**6073**
500	200	200	17971	**2272**
1000	100	200	80681	**36948**
1000	200	200	55243	**10791**
1000	300	200	56070	**2971**

Table 6: Objective value comparison 2

8.3 Comparison summary

In stage of testing, methods of Matlab Optimization Toolbox turned as unsuccessful for our planning problem. Remaining two methods: genetic algorithm and ABC algorithm produced very good results. This is direct acknowledgement, that modern metaheuristic methods provide better results in practical use, than classic deterministic methods. These two successful methods were compared in this chapter.

While elapsed time was very similar for both algorithms, quality of solution represented by objective value was significantly better for ABC algorithm. Because of verification, comparison described in previous chapter was two-times repeated with same result. *ABC algorithm suits best for our planning problem*, so it will be implemented in optimization module.

9. Adding Flexi phase

ABC algorithm was selected as best method for optimization module. Its solution is represented by two charts, first chart represents weekly maintenance capacities and second chart represent progress of objective value in optimization process. Chart of maintenance capacities usually contains peaks.

Flexi phase represents idea of removing peaks from plan. Highest peak is found and then it should be removed. It is done by moving one of rules from week with highest peak to another week. Moving is limited by *Flexibility* value for selected rule. Flexi operation will only be applied if new objective value is better than previous one; in contrary Flexi operation will not be applied.

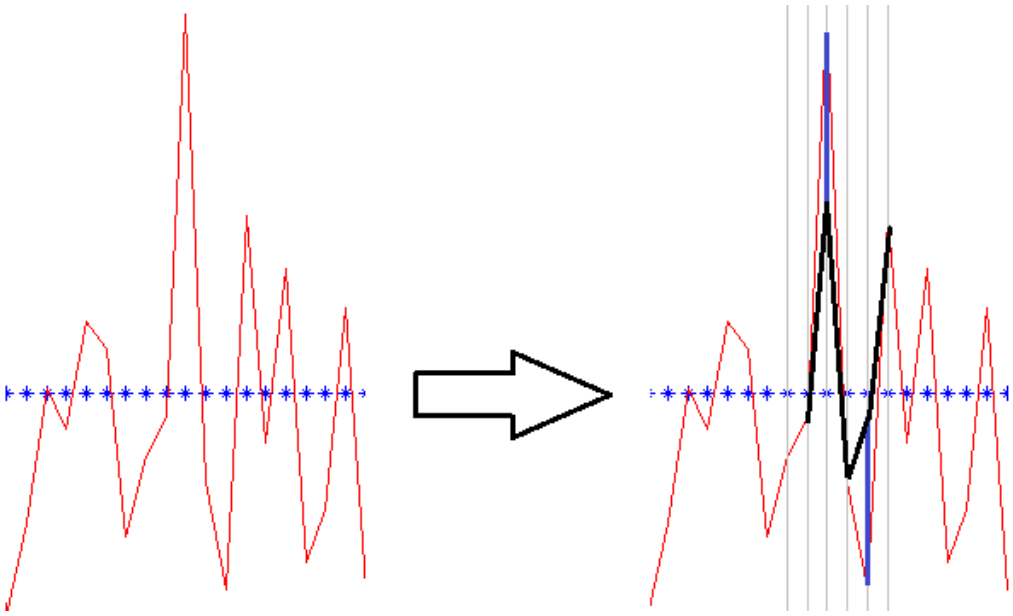

Figure 12: Flexi phase principle

Pseudo-code:

1. *Find biggest peak*
2. *Find place to move activity within Flexibility value*
3. *Move activity*
4. *Decide, whether moving rule improved objective value*

ABC algorithm accompanied by Flexi phase provides better results, because highest peaks in sum vector of ABC solution are be removed in Flexi phase. Flexi phase will be optional part of optimization module.

10. Implementation of FlexiABC algorithm

FlexiABC algorithm introduced in previous chapter is based on Artificial Bee Colony algorithm defined by Prof. Dervis Karaboga in 2005. [15] For purposes of optimization in maintenance planning it was partially changed and updated with Flexi phase. ABC phase is metaheuristics with very good results for specified problem. Flexi phase corresponds with human thinking, while creating optimal plan and it refines solution of ABC phase in very simple way: It finds peaks in plan and tries to get rid of them. Code of FlexiABC algorithm has two main parts; ABC phase and Flexi phase. They use problem specific objective function defined in chapter 3 and also visualization, which will be described later

Code consists from four parts:

- ✓ ABC Phase
- ✓ Flexi Phase
- ✓ Objective function
- ✓ Visualization

Each of these parts will have two classes:

- ✓ Base Class
- ✓ Main Class
    - o Inherits Base Class

In this stage, original data from Act-in database will not be used. Data for implementation stage will be generated pseudo-randomly based on original values. There will be three values necessary for each maintenance rule:

- ✓ Interval (generated in range 1 - 15)
- ✓ Working time (generated in range 1 - 15)
- ✓ Flexibility (generated in range 0 - 4)

There will be three possibilities how to create plan:

- ✓ All-year planning – standard planning problem defined in Planning problem definition
- ✓ From-To planning – standard planning problem with start week and last week specified
- ✓ Planning with regular shutdowns – secondary planning problem defined in Planning problem definition

10.1 Objective function

Code for objective function consists of two classes: *clsObjValueBase* and *clsObjValue*. First class is base class which defines base methods used in second class, which is used in algorithm. Base class contains three methods:

- CreatePlan
- CountSum
- CountObjectiveValue

Class *clsObjValue* in inherits base class, so it can use all the methods of base class and provides methods for FlexiABC algorithm. Methods of *clsObjValue* differ in input and output values. Method *FromWeekToObjValue* represents complete process:

Start week => Plan => Sum => Objective value

All other variants are partial, but they are necessary in algorithm:

- From Week to Objective value
- From Week to Plan
- From Week to Sum
- From Plan to Plan (updates sum vector)
- From Plan to Objective value

10.1.1 Objective value parameters

There are three types of maintenance plan, which can be generated by optimization module. Type of currently generated plan is determined by values of following parameters. First parameter *allYear* determines whether all-year primary planning problem is solved, or secondary planning problem – planning with regular shutdowns is solved. Next two parameters *outStartWeek*, *outLastWeek* are used to set limits of *From-To planning*. Last two parameters *rsStartWeek*, *rsInterval* are used for planning with regular shutdowns, defining week with first shutdown and interval of repetitions. When *allYear* parameter is set to false, shutdown planning can be started with these parameters. All the parameters are public, to be accessible outside its parent class.

VB.NET Code:

```
Public Shared allYear As Boolean
Public Shared outStartWeek As Integer
Public Shared outLastWeek As Integer
Public Shared rsStartWeek As Integer
Public Shared rsInterval As Integer
```

10.1.2 Method for plan creation

CreatePlan method is first of three methods in class *clsObjValueBase*. It takes start week for each rule as input and according to *WorkingHours* and *Interval* values generates the maintenance plan. This plan is an output value.

Input: vector of starting weeks for each maintenance rule

Output: maintenance plan

VB.NET Code:

```
Protected Shared Function CreatePlan(ByVal activities As _
  List(Of clsMaintenanceRule), ByVal startWeek As Integer()) As Integer(,)
  Dim plan(activities.Count, weeksCount - 1) As Integer
  Dim j As Integer
  For i As Integer = 0 To activities.Count - 1
    j = startWeek(i)
    plan(i, j) = activities(i).TimeSpent
    While j < weeksCount - activities(i).Period
      j += activities(i).Period
      plan(i, j) = activities(i).TimeSpent
    End While
  Next
  Return plan
End Function
```

10.1.3 Method for counting sum

CountSum method is second of three methods in class *clsObjValueBase*. It takes the plan from previous method as input and counts sum of working time spent for maintenance in each week of the year. This sum vector is an output value.

Input: Plan created by *CreatePlan* method

Output: Vector of working time sum for each week of year

VB.NET Code:

```
Protected Shared Function CountSum(ByVal activitiesCount As Integer, _
  ByVal plan As Integer(,)) As Integer()
  Dim sum(weeksCount - 1) As Integer
  yearSum = 0
  For i As Integer = outStartWeek To outLastWeek
    For j As Integer = 0 To activitiesCount - 1
      sum(i) += plan(j, i)
    Next
  yearSum += sum(i)
  Next
  Return sum
End Function
```

10.1.4 Method for counting objective value

CountObjValue method is third of three methods in class *clsObjValueBase*. It takes the sum vector from previous method as input and counts objective value for current solution. While there are two planning types, objective value is counted according to selected type of plan. Type is set by *boolean* parameter *allYear*. Objective value is an output value, which represents quality of current solution.

Input: Vector of Sum for each week of year
Output: Objective value of current solution

VB.NET Code:

```
Protected Shared Function CountObjectiveValue(ByVal sum As Integer()) As Double
  Dim fitness As Single = 0
  Dim objVal(weeksCount - 1) As Single
  Dim optimalSumVector(weeksCount - 1) As Single
  Dim numberOfWeek As Integer = outLastWeek - outStartWeek
  Dim upperCoeficient As Single = 2
  Dim lowerCoeficient As Single = 2 / (rsInterval - 1)
  For i As Integer = outStartWeek To outLastWeek
    optimalSumVector(i) = yearSum / 52 * lowerCoeficient
  Next
  Dim maxCount As Integer = rsStartWeek - 1
  While maxCount < outLastWeek + 1
    optimalSumVector(maxCount) = yearSum / 52 * upperCoeficient
    maxCount += rsInterval
  End While
  If allYear Then
    For i As Integer = outStartWeek To outLastWeek
      objVal(i) = (yearSum / numberOfWeek - sum(i)) * (yearSum / numberOfWeek -
sum(i))
      fitness += objVal(i)
    Next
  Else
    For i As Integer = outStartWeek To outLastWeek
      objVal(i) = (optimalSumVector(i) - sum(i)) * (optimalSumVector(i) - sum(i))
      fitness += objVal(i)
    Next
  End If
  Return fitness
End Function
```

10.1.5 Objective value class

Class *clsObjValue* inherits from base class *clsObjValueBase*, so it can use all the methods of base class. It has five methods. First of them is *FromWeekToFitness* and it represents complete process of counting objective value, starting from *start week*, ending with *objective value*. All other variants are partial, but they are necessary in algorithm. This class describes our problem. Well designed objective value class is a key to design working solution for our problem.

VB.NET Code:

```
Public Class ObjVal
  Inherits ObjValueBase
  Public Shared Function FromWeekToFitness(ByVal startWeek As Integer()) As Double
    Dim plan As Integer(,)
    Dim sumVector As Integer()
    plan = CreatePlan(startWeek)
    sumVector = CountSum(plan)
    Return CountFitness(sumVector)
  End Function
  ' ...
End Class
```

10.2 ABC Phase

ABC phase is first part of FlexiABC algorithm. It produces a solution, which can be later updated by Flexi phase. ABC phase is an iterative cycle, where ABC operations are used to improve solution, which repeats until stopping criterion is reached.

Code for ABC phase consists of two classes: *clsAbcBase* and *clsAbc*. First class is base class which defines base methods used in second class, which is used in algorithm. Base class consists from these methods:

- MemorizeBestSource
- InitAll
- CalculateProbabilities
- SendEmployedBees
- SendOnlookerBees
- SendScoutBees

10.2.1 ABC phase constants

There are constants which are necessary for ABC phase. They define parameters of algorithm and have influence on convergence and speed of algorithm. *NP* represents colony size, this means number of employed and onlooker bees. *FoodNumber* represents number of food sources, what is number of solution candidates. *Limit* represents value, after which food source is abandoned. *MaxCycle* value represents stopping criterion. *LowerBound* is always equals to 0, because this is the first possible week to start work (VB.NET starts indexing from 0 [18]). *Runtime* defines number of repetitions of algorithm to see its robustness.

VB.NET Code:

```
Public NP As Integer = 200
Public FoodNumber As Integer = Convert.ToInt32(NP / 2)
Public Limit As Integer = 20
Public MaxCycle As Integer = 150
```

10.2.2 Method for memorizing best source

MemorizeBestSource method checks quality of new solutions and looks for new global minimum. If one of new solutions is better than current global minimum, this solution is saved as new global minimum. This method is called in iterations after applying ABC phase operations to find and save the best solution, which was produced in the run of algorithm.

VB.NET Code:

```
Public Sub MemorizeBestSource()
  For i As Integer = 0 To FoodNumber
    If ObjVal(i) < GlobalMin Then
      GlobalMin = ObjVal(i)
      For j As Integer = 0 To ActivityCount
        GlobalParams(j) = FoodSources(i, j)
      Next
    End If
  Next
End Sub
```

10.2.3 Method for initialization of all sources

InitAll method initializes all solution candidates. This is done by calling *InitOne* method for all solutions. First solution is set to global minimum and its parameters are set to global minimum parameters. This is starting position for ABC phase and it's done only once before start of ABC phase cycle.

VB.NET Code:

```
Public Sub InitAll()
  For i As Integer = 0 To FoodNumber
    InitOne(i)
  Next
  GlobalMin = ObjVal(0)
  For i As Integer = 0 To ActivityCount
    GlobalParams(i) = FoodSources(0, i)
  Next
End Sub
```

10.2.4 Method for calculating probabilities

CalculateProbabilities method calculates probability with which this solution is picked by *Onlooker Bee* to be used and then updated. First the sum of objective values is counted and then probability for each solution is counted based on current objective value and the sum. There are more possibilities, how to count probability, but this one produced best results.

VB.NET Code:

```
For i As Integer = 0 To FoodNumber
  Probs(i) = ObjVal(i) / ObjValSum
Next
```

10.2.5 Method for employed bee phase

SendEmployedBees method is main method of ABC phase which changes current solution to new one, while trying to make the new solution better than current. Method changes solutions for all the food sources and compares their new objective value with old objective value. If there is an improvement, new solution will replace the old one. If not, trial counter increases.

One of the existing food sources is selected randomly. Another food source in neighborhood of selected food sources is selected too. First food source is modified by value of neighbor source. Randomly selected parameter of neighbor source is transferred to selected food source. Objective value of updated selected source is evaluated. If new objective value is better than previous, updated food source is memorized.

VB.NET Code:

```
r = Rnd()
Solution(param2change) = Convert.ToInt32(FoodSources(i, param2change) + _
(FoodSources(i, param2change) - FoodSources(neighbour, param2change)) + _
(r - 0.5) * 2)
If Solution(param2change) < LowerBound Then
  Solution(param2change) = LowerBound
End If
If Solution(param2change) > MyActivities(param2change).Per - 1 Then
  Solution(param2change) = MyActivities(param2change).Per - 1
End If
NewObjVal = MyObjValue.FromWeekToFitness(Solution)
If NewObjVal < ObjVal(i) Then
  Trials(i) = 0
  For j As Integer = 0 To ActivityCount
    FoodSources(i, j) = Solution(j)
  Next
  ObjVal(i) = NewObjVal
Else
  Trials(i) += 1
End If
```

After employed bees have updated food sources, onlooker bees are going to update food sources too. Onlooker bees are not improving all the food sources, but they improve food sources with probability which is proportional to their quality, represented by its objective value. These probabilities are calculated by *CalculateProbabilities* method.

10.2.6 Method for scout bee phase

SendScoutBees method looks for solution with maximum of unsuccessful trials to improve. If there is a solution with number of unsuccessful trials higher than *Limit* value, solution is newly initialized by calling *InitOne* method as was done in *InitAll* method on beginning of ABC phase.

VB.NET Code:

```
Public Sub SendScoutBees()
  Dim MaxTrialIndex As Integer = 0
  For i As Integer = 0 To FoodNumber
    If Trials(i) > MaxTrialIndex Then
      MaxTrialIndex = i
    End If
  Next
  If Trials(MaxTrialIndex) >= Limit Then
    InitOne(MaxTrialIndex)
  End If
End Sub
```

10.2.7 ABC phase class

Class *clsABC* inherits from base class, so it uses all the methods from base class and creates complete ABC Phase with pre-defined parameters. It initializes food sources and improves them iteratively. Best solution is saved as global minimum in each cycle. Progress of algorithm is visualized via *Draw* method. Class *clsABC* has only one shared method (static in C# [18]), so instances of this class cannot be created, it is used directly.

VB.NET Code:

```
Public Class clsABC
  Inherits clsABCBase
  Public Shared Function RunABC() As Integer()
    Try
      InitAll()
      MemorizeBestSource()
      For iter As Integer = 0 To MaxCycle
        SendEmployedBees()
        CalculateProbabilities()
        SendOnlookerBees()
        MemorizeBestSource()
        SendScoutBees()
        If iter = 0 Then
          DrawFirst()
        Else
          DrawABC()
        End If
      Next
    Catch ex As Exception
     MsgBox("ABC optimization failed!", MsgBoxStyle.ApplicationModal, "ABC Phase")
    End Try
    Return GlobalParams
  End Function
End Class ' clsABC
```

10.3 Flexi Phase

Flexi Phase is second part of FlexiABC algorithm. ABC phase is metaheuristic optimization method from category of swarm intelligence [13]. Flexi phase was completely designed and developed in this thesis and tries to refine solution computed by ABC phase by removing peaks in the sum vector. Flexi phase finds the highest peak in the sum vector in each

cycle and tries to get rid of it. So plan is becoming smoother, what is the goal of optimization.

While flexibility is not defined as parameter of maintenance rules in Maintenance Control [21], Flexi phase of optimization is optional. There is a *CheckBox*, which controls whether Flexi phase will be used or not. Flexibility for rules is set as fraction of period value (e.g. 1/4).

10.3.1 Method for preparing Flexi phase

Prepare method connects ABC phase with Flexi phase. It gets solution from ABC phase as input parameter for *Flexi* optimization. It counts its objective value and creates plan, both based on ABC phase solution. ABC phase solution is also visualized in time chart.

VB.NET Code:

```
Protected Sub Prepare(ByVal GlobalParams)
  FlexiFitness = MyObjValue.FromWeekToFitness(GlobalParams)
  _Plan = MyObjValue.FromWeekToPlan(GlobalParams)
  frmVis.chrtTime.Series("Series1").Points.Clear()
  For i As Integer = 0 To 51
    frmVis.chrtTime.Series("Series1").Points._
      AddY(_Plan(MyActivities.Count, i))
  Next
  frmVis.chrtTime.Update()
End Sub
```

10.3.2 Method for removing peaks

RemovePeak method is main method of Flexi phase. It looks for peaks and tries to get rid of them. It has to find week to change; week with highest amount of working time. After that, rule to change has to be found. Rule to change is non-zero minimum of working time in selected week. Then position where to move rule is generated pseudo-randomly based on *Flexibility* value. Plan is updated and also its objective value is updated.

VB.NET Code:

```
If MyActivities(ItemForChangeIndex).Flex > 0 Then
  While MoveValue = 0
    MoveValue = Math.Round(Rnd() * MyActivities(ItemForChangeIndex).Flex * 2)_
    - MyActivities(ItemForChangeIndex).Flex
  End While
End If
_PlanUpdate = _Plan
If (WeekWithMaxIndex+MoveValue > -1) And (WeekWithMaxIndex+MoveValue < 52) Then
  _PlanUpdate(ItemForChangeIndex, WeekWithMaxIndex + MoveValue) = _
       _PlanUpdate(ItemForChangeIndex, WeekWithMaxIndex)   ' Move value
  _PlanUpdate(ItemForChangeIndex, WeekWithMaxIndex) = 0  ' Reset origin
End If
FlexiFitnessUpdate = MyObjValue.FromPlanToFitness(_PlanUpdate)
_PlanUpdate = MyObjValue.FromPlanToPlan(_PlanUpdate)
```

10.3.3 Method for visualization

DrawFlexi method takes care about visualization of Flexi phase. If objective value of updated solution is better than current objective value, updated solution will replace current solution and its objective value is added to objective value chart. Maintenance capacities chart is updated too.

VB.NET Code:

```
Protected Shared Sub DrawFlexi()
  ' Compare original fitness and updated fitness
  If FlexiFitnessUpdate < FlexiFitness Then
    _Plan = _PlanUpdate
    FlexiFitness = FlexiFitnessUpdate
    frmVis.chrtFit.Series("Series1").Points.Add(FlexiFitnessUpdate)
    frmVis.chrtFit.Titles(1).Text = "Actual fitness value: " &_
       Math.Round(FlexiFitness).ToString
  Else
    frmVis.chrtFit.Series("Series1").Points.Add(FlexiFitness)
  End If
  frmVis.chrtFit.Update()
  frmVis.chrtTime.Series("Series1").Points.Clear()
  For i As Integer = 0 To 51
    frmVis.chrtTime.Series("Series1").Points.Add(_Plan(MyActivities.Count, i))
  Next
  frmVis.chrtTime.Update()
End Sub
```

10.3.4 Flexi Phase class

Flexi phase represents cycle, which removes peaks in sum vector. Peaks are removed iteratively limited by maximal number of iterations. If Flexi phase is allowed by user, Prepare method takes ABC solution as first step of Flexi phase. When this is done, Flexi cycle is started. In each cycle there is highest peak found and removed by *RemovePeak* method. When peak is removed successfully with improving objective value, operation is memorized and visualized in charts, in contrary operation is forgotten.

VB.NET Code:

```
Public Class clsFlexi
  Inherits clsFlexiBase
  Public Shared Sub RunFlexi(ByVal GlobalParams As Integer())
    Const maxCycle As Integer = 150
    Try
      Prepare(GlobalParams)
      For flexIndex As Integer = 0 To maxCycle - 1
        RemovePeak()
        DrawFlexi()
      Next
    Catch ex As Exception
      MsgBox("Flexi optimization failed!", MsgBoxStyle.ApplicationModal, "Flexi")
    End Try
  End Sub
End Class
```

11. Visualization of FlexiABC algorithm

Progress of FlexiABC algorithm is visualized in two real-time charts. First chart represents progress of objective value. Second chart represents weekly amount of maintenance capacities. Both charts are updated each iteration of algorithm, so they can show actual progress to user. Charts were designed via Microsoft Chart Control .NET components described in [19]. They provide simple and very good designed solution for creating charts in Microsoft .NET Framework.

First chart represents progress of objective value for iterations of ABC and Flexi phase. Actual objective value is added to the chart to provide comparison of actual solution with previous, what enables to see whether the algorithm improves solution or not.

Second chart represents actual weekly maintenance capacities. It shows whether the maintenance is divided evenly or not. Algorithm is trying to remove peaks (minimums and maximums) and we can see results in chart. It very easy to decide how to continue: Accept results and generate plan or restart algorithm to achieve better results.

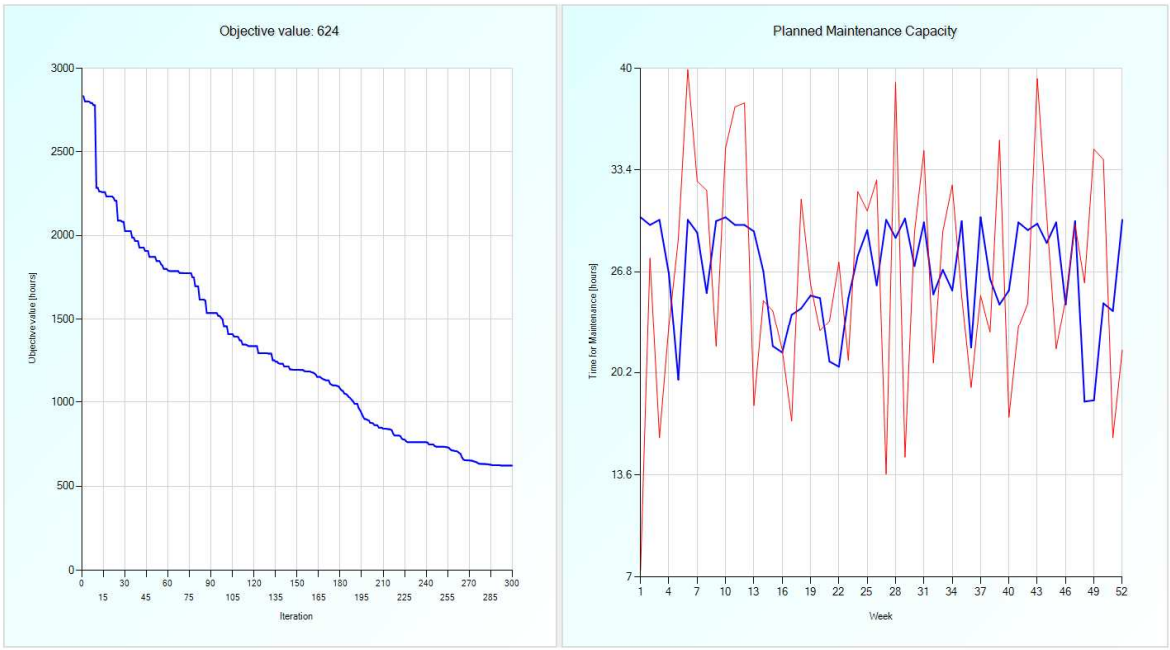

Figure 13: Visualization charts

Before the algorithm starts, charts annotations (title, axis names and units) are set. Type of both charts is set to line chart. When algorithm is running, it just adds the actual objective value (actual global minimum) to chart and updates graphical user interface (GUI). Maintenance capacities chart is redrawn in iterations to provide overview of actual solution, while result of first iteration and actual result are displayed for comparison.

VB.NET Code:

```
Public Sub FitnessChartAnottations()
  With chrtFit
    .Series("Series1").ChartType = _
       DataVisualization.Charting.SeriesChartType.Line
    .Series("Series1").IsVisibleInLegend = False
  End With
  With chrtFit.ChartAreas("ChartArea1").AxisX
    .Title = "Iteration"
    .Minimum = 0
    .Maximum = 300
    .Interval = 15
    .MajorGrid.LineColor = Color.LightGray
  End With
  With chrtFit.ChartAreas("ChartArea1").AxisY
    .Title = "Fitness Value"
    .MajorGrid.LineColor = Color.LightGray
  End With
End Sub

Public Sub DrawFitness()
  ' Update Fitness chart
  chrtFit.Titles(1).Text = "Actual fitness value: " & _
       Math.Round(GlobalMin).ToString
  chrtFit.Series("Series1").Points.Add(GlobalMin)
  chrtFit.Update()
End Sub
```

12. Incorporation of FlexiABC algorithm

Maintenance Control is industrial information system developed by Dutch company Act-in B.V. Maintenance Control is focused on maintenance management in factories, but also contains other modules necessary for complex industrial information system. All the objects of factory are defined in object structure module. Objects are sorted in tree-view, what simplifies orientation in really big factories. Notification module is an input for various types of tasks, which should be done. Failures, breakdowns and downtimes with its reasons are inserted to system. Notification is a key to create work order. Work order is document for employees, where their work is specified. Various types of documents and also material can be assigned to work order. After work order is completed, employee can confirm what was done, where problem was and what material did he need. Assigned material and Attached documents are modules for handling material and documents which are assigned to notifications and work orders. Other modules shown in figure 14 are optional; depends on company requirements.

Maintenance Control also contains module for planning called Planned Maintenance, which provides manual and also automatic planning. This module does not implement any optimization method, so its plans are not optimal. Each maintenance rule is planned in first week and then regularly based on its interval. This means, in first week is working time spent for maintenance very high and next week has almost no planned working time. You can also create your plan manually, but this is not very good choice for plan with high number of maintenance rules. Planning module will be accompanied with metaheuristic FlexiABC algorithm introduced in previous chapters to provide maintenance plans with even weekly amount of maintenance capacities and also plans for production with regular shutdowns.

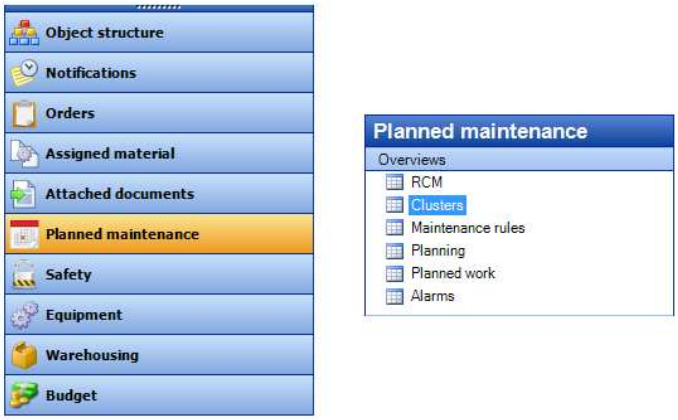

Figure 14: Maintenance Control modules [21]

12.1 Planned Maintenance module

Planned Maintenance [21] is one of modules of Maintenance Control, which handles functions for creating maintenance plans. Planned Maintenance defines maintenance rules, cluster of maintenance rules and creates plans. Maintenance rules are defined and separated to clusters. Rules in one cluster are dedicated to one group of employees e.g. mechanic maintenance. When maintenance rules are separated to its clusters, we can start to create plan. There is plan for each group and each year, e.g. mechanic maintenance plan for 2012.

12.2 Planning window

Planning window enables user to create plan for selected cluster and year. After cluster is selected, all the maintenance rules for this cluster are loaded from database and displayed in the plan board. If there was no plan created for this cluster and year, plan board is empty and can be created: manually or automatically. Manual planning is usually used for small clusters. Automatic planning can be used either for bigger clusters, but these plans are not optimized, so maintenance capacities are rapidly changing, what is not effective.

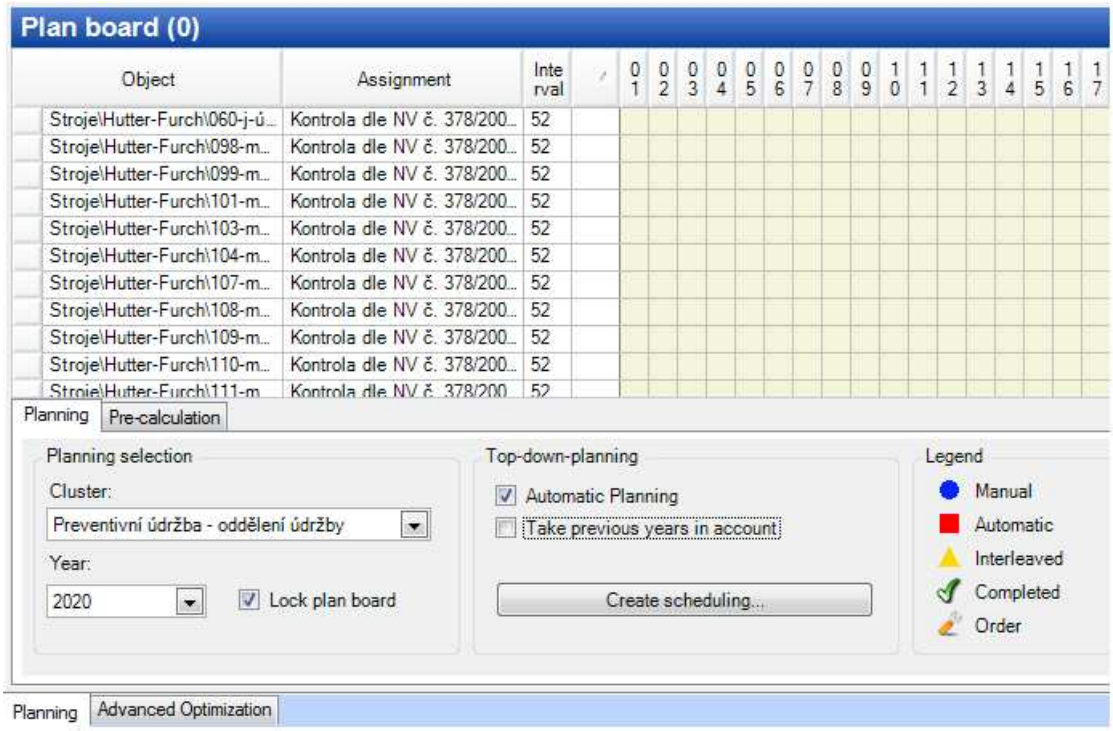

Figure 15: Empty plan board

12.3 Adding Advanced Optimization tab

Optimization module called *Advanced Optimization* will extend functionality of Planned Maintenance. There will be *tab control* added to *Planning* window. First *tab page* contains original controls like plan board, *combo boxes* for cluster and year selection and automatic planning *button*. New tab page contains controls for work with FlexiABC algorithm. There will be two charts for visualization of optimization progress done by algorithm and other controls of optimization process situated in *Options* panel.

Optimization module will be able to create three types of maintenance plans. Standard all-year maintenance plan with even weekly amount of maintenance capacities, standard plan with specified first and last week and also plan for production with regular shutdowns. Type of plan and its input parameters will be set in *Options* panel of optimization module. Standard all-year maintenance plan is generated in two phases: ABC phase and Flexi phase. Flexi phase is used only when check box *Enable flexibility* is checked. Optimization algorithm is started by pressing *Start* button. If user accepts results, they will be transferred to plan board and database after pressing *Accept* button.

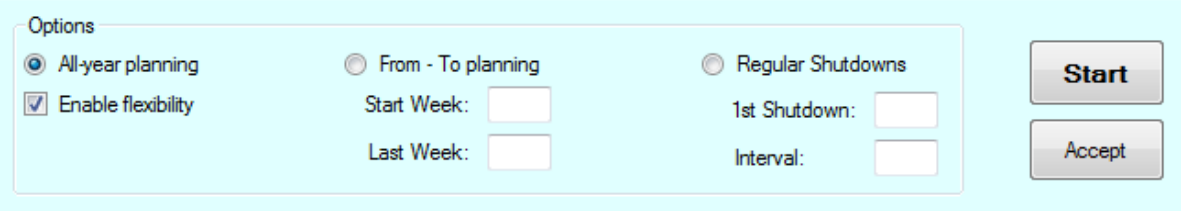

Figure 16: Options panel

Transfer of FlexiABC results to plan board will be done using standard Maintenance Control functions for handling plans in Planned Maintenance module. These functions gets result of optimization and display it in plan board. Result is also saved in MS SQL Server [20] database as all the data in Maintenance Control.

Quality of plan is represented by objective value, which is displayed not only in objective value chart, but also on plan board (figure 15) to help decide, whether is current plan better than previous one or not.

13. Optimization module in practical use

Optimization module called Advanced Optimization became a part of industrial information system Maintenance Control. It extends capabilities of Planned Maintenance module with option of creating maintenance plans with even weekly amount of maintenance capacities and also with option of creating maintenance plans for production with regular shutdowns. In stage of design, optimization algorithms were tested on pseudo-randomly generated data, which should be similar to real data from production environment. In this chapter, optimization module and its algorithm will be tested on real data from Czech customers using Planned Maintenance module of Maintenance Control. Table 6 shows list of selected clusters of these customers and marking in this chapter. *MC data (name of clusters and maintenance rules) are in Czech language, because they are from Czech customers.*

Factory	Cluster	Number of rules	Cluster mark
Norma Group in Hustopeče	Interní oddelení udrzby	525	A1
Norma Group in Hustopeče	Obsluha stroje	9	A2
OP Papírna in Olšany	Odstávky PS1	135	B1

Table 7: Customer's clusters

Optimization module will be tested on real data for three clusters:

- Small cluster – A2 – 9 maintenance rules
- Medium cluster – B1 – 135 maintenance rules
- Large cluster – A1 – 525 maintenance rules

13.1 Influence of Flexi phase

ABC algorithm was selected as best method for creating maintenance plans in previous chapters. It was accompanied by Flexi phase, which represents idea of removing peaks in weekly sum vector of maintenance capacities. Flexi phase should refine solution of ABC phase, but it slightly changes interval value. Flexi phase is set as optional for users, who wants to keep interval value constant. Table 8 compares objective value after ABC phase with objective value after Flexi phase.

	Cluster		
	Small – A2	Medium – B1	Large – A1
Obj. value after ABC phase [h]	65	76	285
Obj. value after Flexi phase [h]	*36*	*39*	*144*

Table 8: Influence of Flexi phase

Objective value shown in table 8 represents average value for twenty runs of algorithm. Influence of Flexi phase is very significant, because it is able to improve ABC phase solution by reducing its objective value approximately by half.

13.1 Planning with even weekly amount of maintenance capacities

Planning with even amount of maintenance capacities is primary planning problem, which produced very good results in phase of testing and implementation of FlexiABC algorithm.

Small cluster A2 contains only 9 maintenance rules. Plan for this cluster can be created manually, but optimization module should be also able to create plans with only few maintenance rules. Automatically created plan for this cluster was shown in problem definition and now, it can be compared with plan created by optimization module (figure 17). Title of *Plan board* displays objective value, which shows difference of current plan from global optimum in hours. Objective value of automatic plan is 682 hours, while objective value of optimized plan is only 38 hours. Optimized plan is much better, because objective value is much lower and weekly sum of maintenance is almost constant.

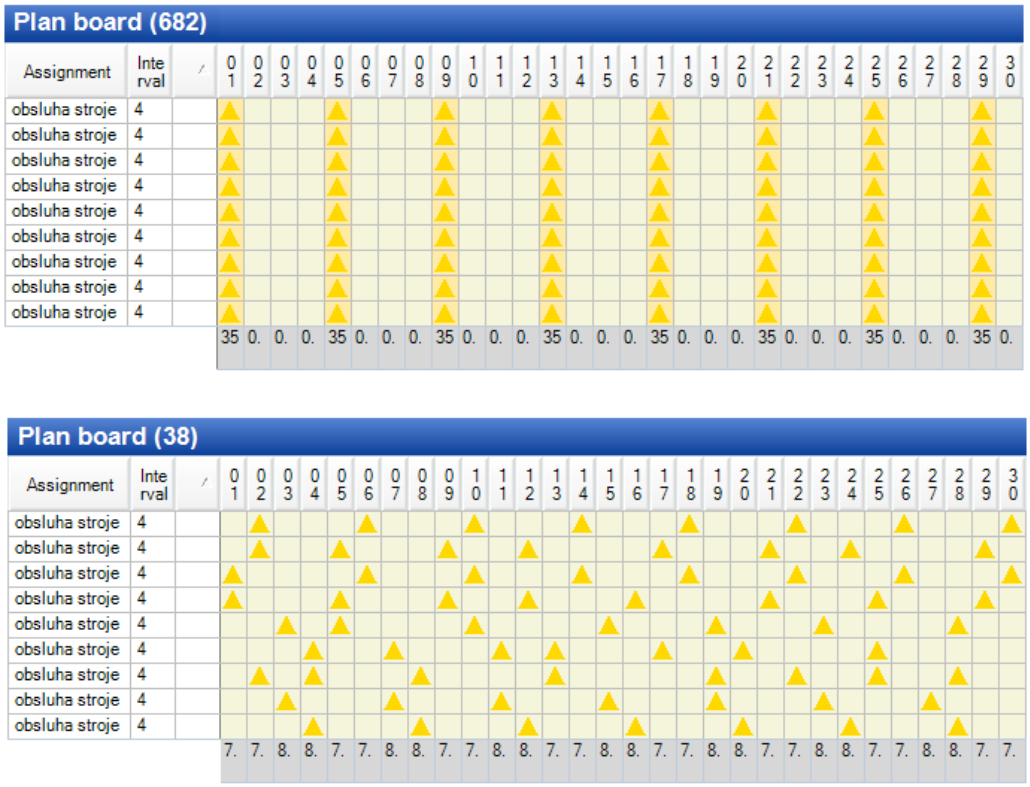

Figure 17: Plans comparison for small cluster (A2)

Medium cluster B1 contains 135 maintenance rules. Plan for this cluster can be created manually very hardly; also quality of manual plan can be bad. Creating plan for medium cluster is typical task for automatic planning. Plan can be created using automatic planning and optimization module (figure 18). Automatic plan has objective value equal to 482 hours, while optimized plan's objective value is only 35 hours. Weekly maintenance capacities are almost constant in optimized plan, but they are rapidly changing in automatic plan. Optimized plan is much better than automatic plan and it can be used in practical use.

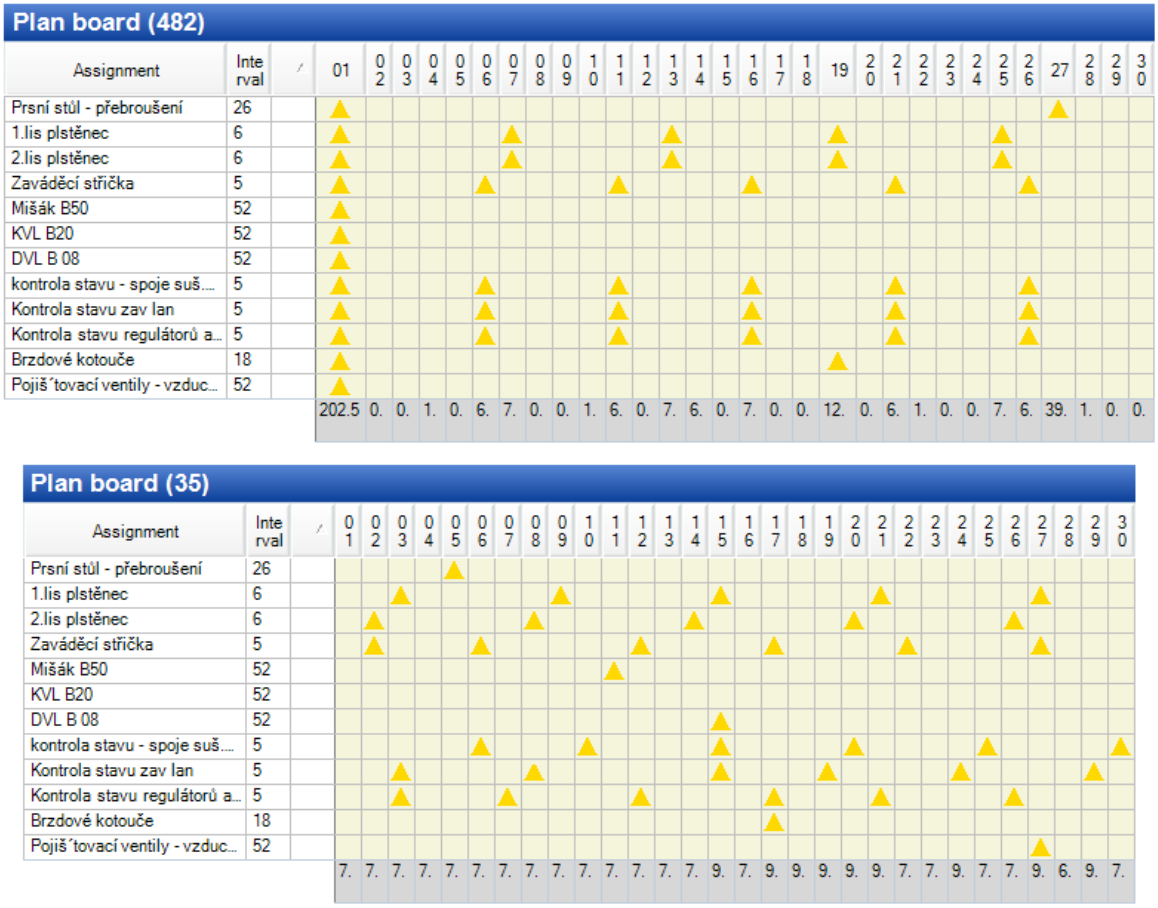

Figure 18: Plans comparison for medium cluster (B1)

Large cluster A1 contains 525 maintenance rules. Plan for cluster cannot be created manually, because of high number of rules. Automatic planning can create plan for this cluster and also optimization module can handle it. Objective value of automatic plan is equal to 2116 hours; objective value for optimized plan is only 139 hours (figure 19). Weekly sum of maintenance capacities is very different; Sum in automatic plan is changing from 0 up to 700 hours weekly. Sum in optimized plan is changing from 22 hours to 30 hours. Optimized plan's

range is very small in compare with range of automatic plan. Optimized plan can be used practically because of almost constant weekly sum of maintenance capacities.

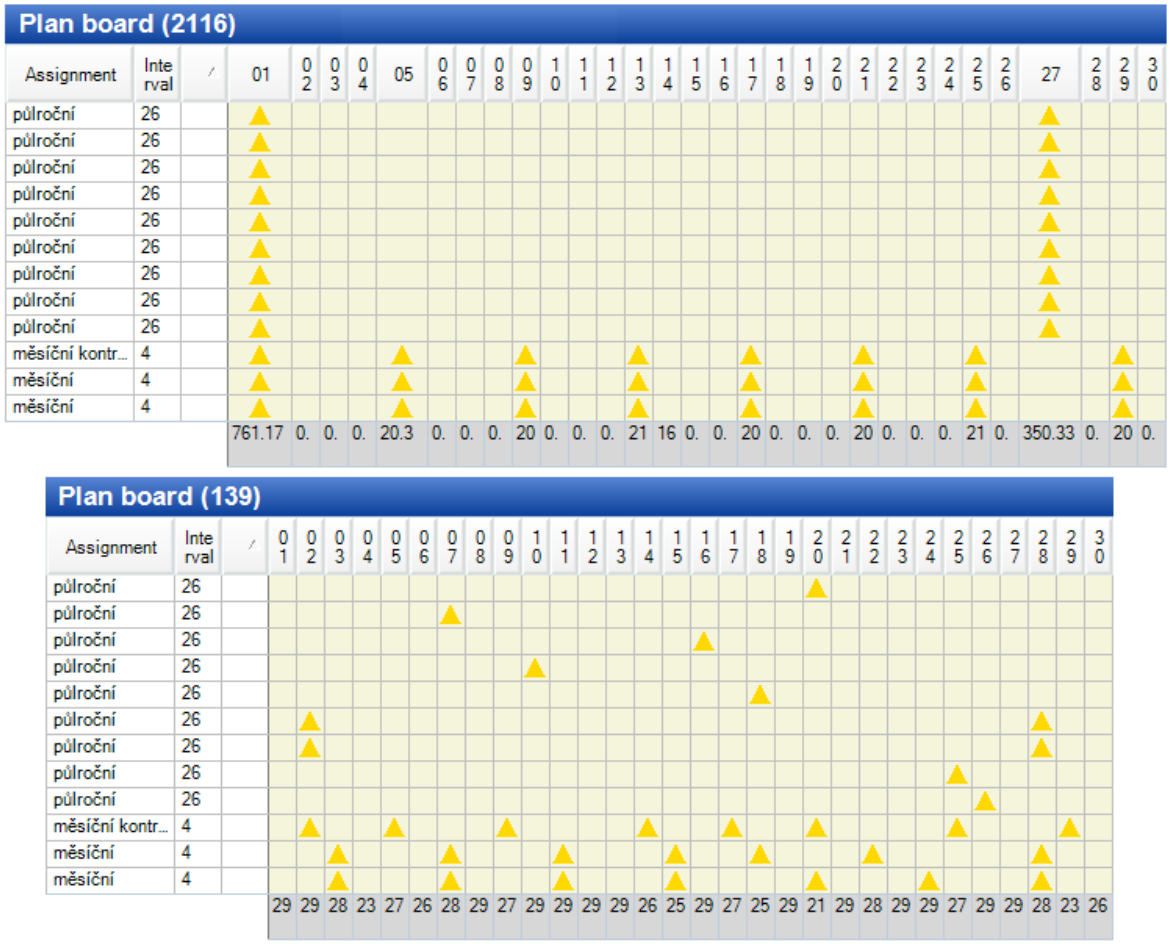

Figure 19: Plans comparison for large cluster (A1)

Optimization module is also able to create partial plan with specified start week and end week. Algorithm works exactly the same, but planning period is not whole year, but only several weeks. Sum of maintenance capacities is equal to zero in weeks, which are out of specified period and even in specified weeks. This option is new, because automatic planning does not have it. Partial plan for cluster B1, starting in 8th week, ending in 37th week, is shown in figure 20.

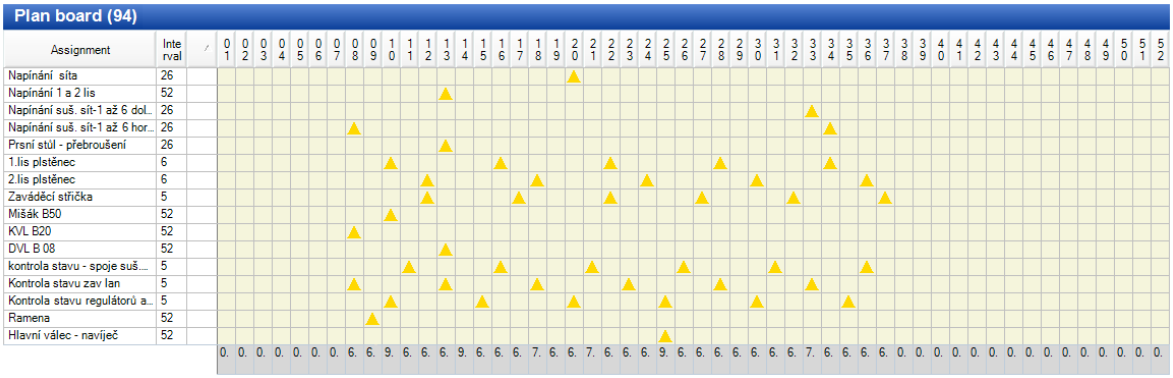

Figure 20: Partial plan (B1)

13.2 Planning with regular shutdowns

Secondary planning problem is planning for production with regular shutdowns. Its solution uses second version of objective function described in chapter 3.5. Weekly sum of maintenance capacities should be even in production weeks and doubled in weeks of shutdown. This option is also new, because automatic planning does not support it. Figure 21 shows plan for production with first shutdown in second week and with interval of ten weeks. Amount of maintenance capacities is almost even in production weeks (6 – 7 hours in average) and it is doubled in weeks of shutdown (2nd week, 12th week …). This plan meets criteria for plan for production with regular shutdowns.

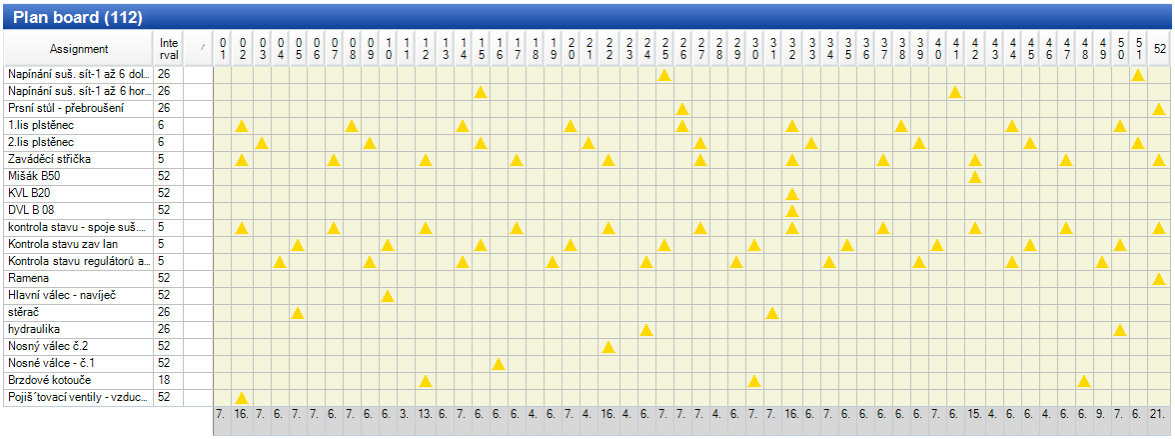

Figure 21: Plan with regular shutdowns (B1)

Planning with regular shutdown hardly depend on rules' parameters in selected cluster. For clusters, where interval value is changing from low values (2 – 6 weeks) up to high value (26 – 52 weeks), quality of plan is very good. For clusters, where interval value is mostly only low or only high, quality of plan is not good, because problem is too constrained.

Figure 22a shows maintenance capacities in cluster with all the interval values lower than 12 weeks. Middle part of plan is generated well, but first weeks and also last weeks are not planned well. Amount of maintenance capacities in week of shutdown is not double in compare with production week as it should be. Cluster contains only low interval values, so problem is too constrained and algorithm is not able to deal with it. After changing interval values for several maintenance rules (20% of rules in cluster) to higher values (26 weeks, 52 weeks) resulting plan shown in figure 22b is much better.

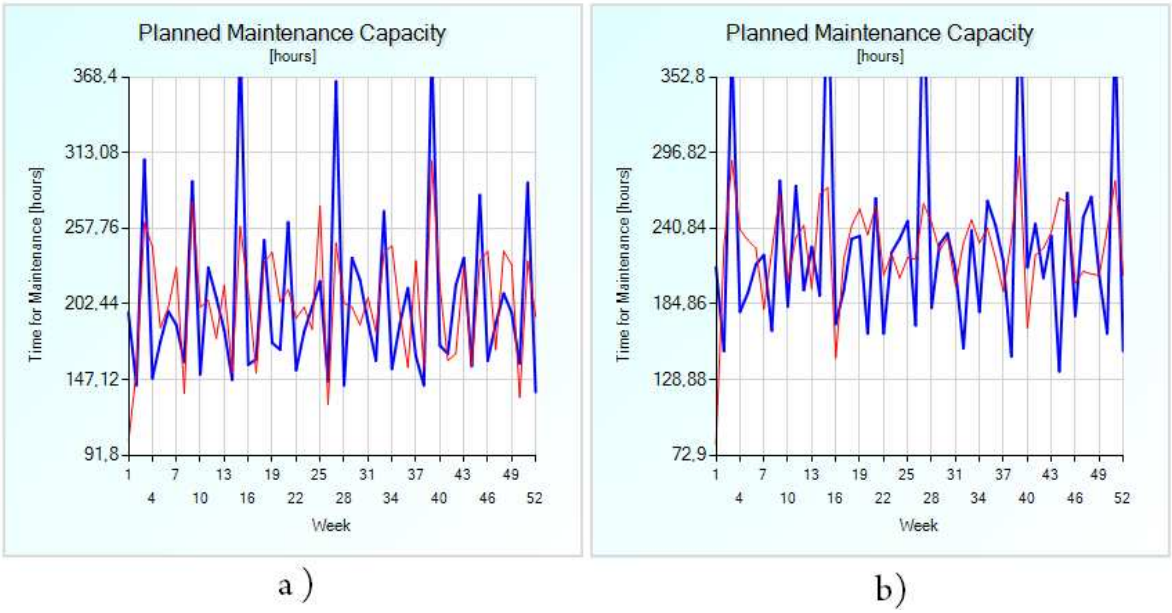

Figure 22: Constraints problem

14. Conclusion

Goal of this master's thesis is design of intelligent optimization module for industrial information system Act-in Maintenance Control. Optimization module should be able to create all-year maintenance plans with even weekly amount of maintenance capacities and plans for production with regular shutdowns. Optimization module takes cluster of maintenance rules as input and based on its algorithm, it creates optimal maintenance plan.

First step of design process is study of optimization methods suitable for planning problems. Optimization methods overview provides wide spectrum of methods. There were deterministic and probabilistic methods introduced. According to advice of thesis supervisor, modern probabilistic methods were prioritized, but deterministic methods were studied too. Three optimization methods were selected for next phase, testing. Non-linear least squares from Matlab Optimization Toolbox as deterministic method and genetic algorithm and artificial bee colony algorithm (ABC) as probabilistic methods were selected for testing on planning problem in Matlab environment.

Before testing could start, planning problem was described with objective function. Objective function decides quality of current plan and marks it by objective value. Selected methods were tested with same pseudo-randomly generated data to make comparison simple. Non-linear least squares turns as not suitable method for this problem, but genetic algorithm and ABC algorithm produced very good results. After testing various input parameters, ABC algorithm was selected as best method, which will be implemented in optimization module. ABC algorithm as metaheuristic method was accompanied by Flexi phase. Flexi phase is simple algorithm, which finds peaks in sum vector of maintenance plan and tries to remove them by updating plan. After testing phase, FlexiABC code was completed in Matlab environment.

In implementation phase FlexiABC code was translated to VB.NET language of Microsoft Visual Studio to be compatible with other MC code. Code was also edited to fulfils requirements of object oriented programming in Microsoft .NET Framework. When algorithm was working separately as standalone application, it was ready to be implemented in Maintenance Control (MC). Input values are loaded from Act-in database using standard MC methods and processed by FlexiABC algorithm. Optimization process is visualized in two charts. First chart represents progress of objective value; second chart shows weekly maintenance capacities. When optimization is finished, user can decide what to do – accept

result; restart optimization or cancel it. If results are accepted, they will be displayed in plan board and saved to database.

Optimization module as member of industrial information system Maintenance Control was finished in April 2012. It is able to create plans with even weekly amount of maintenance capacities, partial plans and also plans for production with regular shutdowns. Optimization module was rated as helpful tool, which saves time and improves quality of maintenance plans. Its simplicity, speed and solution quality were rated as its biggest advantages.

Optimization module was tested on real data from Norma Group factory in Hustopeče and OP Papírna factory in Olšany. Optimized all-year maintenance plans were created and compared with actual plans. Quality of optimized plans, represented by objective value, was much better than quality of automatic plans as shown in chapter 13. Also partial plans and plans with regular shutdowns were created with good quality. Based on this conclusion, optimization module is ready to be deployed to MC customers, so goal of this master's thesis was reached.

Bibliography

[1] Weise T.: Global Optimization Algorithms – Theory and Application, Online, 2012
http://www.it-weise.de

[2] Mathworks Online Documentation: Optimization toolbox description
http://www.mathworks.com/help/toolbox/optim/ug/bs_fzn8.html

[3] Mathworks Online Documentation: *fmincon* function
http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html

[4] Mathworks Online Documentation: *fsolve* function
http://www.mathworks.com/help/toolbox/optim/ug/fsolve.html

[5] Mathworks Online Documentation: *lsqnonlin* function
http://www.mathworks.com/help/toolbox/optim/ug/lsqnonlin.html

[6] Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975

[7] Fogel, D.B.: Evolutionary Computation. New York, IEEE Press, 1995

[8] Darwin, Ch.: The Origin of Species. London, John Murray, 1859

[9] Mendel, G.: The Origin of Genetics – A Source Book. Translated by C. Stern and Eva R. Sherwood. W. H. Freeman, San Francisco – London, 1966

[10] Schaffer, J.D., Eshelman, L.J.: On Crossover as an Evolutionary Viable Strategy. Proceedings of the Fourth International Conference on Genetic Algorithms. R. Belew, Booker L., eds., San Mateo, CA: Morgan Kaufman, 1991, 61-68

[11] Spears, W.M.: Crossover or Mutation? Proceedings of the Second Foundations of Genetic Algorithms Workshop, Whitley D., ed., San Mateo, CA: Morgan Kaufman, 1992, 221-237

[12] Genetic Algorithm for Hello World (available 15.3.2012)
http://www.puremango.co.uk/2010/12/genetic-algorithm-for-hello-world/

[13] James F. Kennedy, Russel C. Eberhart, Yuhui Shi: Swarm Intelligence, Morgan Kaufmann, 2001

[14] Beni, G., Wang, J.: Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30 (1989)

[15] Artificial Bee Colony (ABC) Algorithm Homepage (available on 6.3.2012)
http://mf.erciyes.edu.tr/abc

[16] D. Karaboga, B. Basturk, On The Performance Of Artificial Bee Colony (ABC) Algorithm, Applied Soft Computing,Volume 8, Issue 1, January 2008, Pages 687-697.

[17] Baykasoulu A., Özbakir L., Tapkan P.: Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem, University of Gaziantep, Department of Industrial Engineering, Erciyes University, Department of Industrial Engineering, Turkey

[18] Microsoft Developers Network
http://msdn.microsoft.com

[19] Samples Environment for .NET Framework 4 Chart Controls
http://archive.msdn.microsoft.com/mschart

[20] SQL Server Online Forums
http://social.msdn.microsoft.com/Forums/en/category/sqlserver

[21] Act-in Internal Documentation

List of symbols, physical constants and abbreviations

f	objective function
x	solution candidate
S	solution space
X^*	Global optimum set
G	Inputs
g	input

EA	Evolutionary algorithms
GA	Genetic algorithms
SI	Swarm Intelligence
ABC	Artificial Bee Colony
MC	Maintenance Control