Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's thesis

Evolutionary Metaheuristics for the Nurse Rostering Problem

Michael Rudolf

Supervisor: Ing. Jiří Kubalík, Ph.D.

Study Programme: Open Informatics

Field of Study: Computer and Information Science

May 21, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Michael R u d o l f

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Rozvrhování směn pomocí evolučních metaheuristik

Pokyny pro vypracování:

1. Prostudujte problematiku z oblasti rozvrhování, konkrétně problém Nurse Rostering Problem (NRP). Prostudujte stávající metaheuristické optimalizační metody používané pro řešení NRP. Zaměřte se na přístupy založené na iterovaném lokálním prohledávání.
2. Prostudujte problematiku evolučních algoritmů a navrhněte možnost jejich využití v optimalizačních algoritmech pro řešení NRP.
3. Navrhněte a naimplementujte evoluční algoritmus pro řešení NRP.
4. Experimentálně ověřte funkčnost navrženého algoritmu na vybraných testovacích instancích.
5. Dosažené výsledky vyhodnoťte a srovnejte s jinými metodami.

Seznam odborné literatury:

[1] San Luke: Essentials of Metaheuristics. Lulu, 2013, 2. vydání, volně dostupné na http://cs.gmu.edu/~sean/book/metaheuristics/
[2] E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem: The state of the art of nurse rostering. Journal of Scheduling, 7(6):441-499, 2004.
[3] Edmund K. Burke, Timothy Curtois, Rong Qu, Greet Vanden Berghe: A Time-Predefined Variable Depth Search for Nurse Rostering. Journal on Computing, Volume 25 Issue 3, Summer 2013.

Vedoucí bakalářské práce: Ing. Jiří Kubalík, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Michael R u d o l f

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Evolutionary Metaheuristics for the Rostering Problem

Guidelines:

1. Study an optimization problem known as the Nurse Rostering Problem (NRP). Review metaheuristic approaches used for solving the NRP. Focus on approaches based on an iterated local search.
2. Study evolutionary algorithms and review their potential for solving the NRP.
3. Propose and implement evolutionary-based algorithm for solving the NRP.
4. Design proof-of-concept experiments and experimentally evaluate a performance of the proposed algorithm on standard test instances.
5. Analyse achieved results and compare the proposed algorithm with other existing approaches.

Bibliography/Sources:

[1] San Luke: Essentials of Metaheuristics. Lulu, 2013, 2. vydání, volně dostupné na http://cs.gmu.edu/~sean/book/metaheuristics/
[2] E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem: The state of the art of nurse rostering. Journal of Scheduling, 7(6):441-499, 2004.
[3] Edmund K. Burke, Timothy Curtois, Rong Qu, Greet Vanden Berghe: A Time-Predefined Variable Depth Search for Nurse Rostering. Journal on Computing, Volume 25 Issue 3, Summer 2013.

Bachelor Project Supervisor: Ing. Jiří Kubalík, Ph.D.

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

ABSTRACT

The topic of this bachelor's thesis is the nurse rostering problem (NRP), which is a well known NP-hard problem. An integral part of this thesis is a survey of the rostering problem and its representation. The main goal is to design and implement an iterative algorithm, applying evolutionary and other meta-heuristics for solving instances of NRP. The commonly used solving procedures are introduced and compared with the proposed one. Proposed solution is inspired by heuristics such as Lin-Kernighan heuristic or variable depth search (VDS). Designed algorithm is tested on benchmark instances and compared with best known solutions. On some instances, solutions equal to the best known ones were found.

KEYWORDS

nurse rostering problem, evolutionary algorithm, shift rostering, personnel scheduling, Lin-Kernighan heuristic, variable depth search

ABSTRAKT

Tématem této bakalářské práce je rozvrhování směn zdravotním sestrám (Nurse Rostering Problem - NRP), což je dobře známý NP-těžký problém. Nedílnou součástí práce je uvedení do problému rozvrhování a jeho reprezentace. Hlavním cílem je navrhnout a implementovat iterativní algoritmus, používající evoluční a další metaheuristiky pro řešení instancí problému NRP. V práci jsou představeny běžně používané postupy řešení, které jsou porovnány s postupem navrhovaným. Navrhované řešení je inspirované heuristikami jako Lin-Kernighan heuristika nebo variable depth search (VDS). Implementovaný algoritmus je testován na benchmarkových instancích a porovnáván s nejlepšími dosaženými výsledky. U několika instancí pak dosažené výsledky vyrovnají nejlepší nalezená řešení.

KLÍČOVÁ SLOVA

nurse rostering problem, evoluční algoritmus, rozvrhování směn, rozvrhování pro personál, Lin-Kernighan heuristika, variable depth search

RUDOLF, Michael *Evolutionary Metaheuristics for the Nurse Rostering Problem*: bachelor's thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 10.5.2014. 44 p. Supervised by Ing. Jiří Kubalík, Ph.D.

DECLARATION

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60(1) of the Act.

Prague May 21, 2014

ACKNOWLEDGEMENT

I would like to thank my supervisor Ing. Jiří Kubalík, Ph.D., for his help and much advice he provided me with during the work on my Bachelor's Thesis.

CONTENTS

LIST OF ALGORITHMS

LIST OF TABLES

INTRODUCTION

Rostering nurse shifts in hospitals is a complex and highly constrained problem. It involves creating a roster of shifts for a specific hospital ward for a predefined number of days. It is of course important to efficiently distribute workload among the staff in order to satisfy the hospital and personal preferences. High quality rosters lead to more satisfied and effective employees.

Traditionally, the task of rostering is done by hand. This type of scheduling is usually referred to as *Self-scheduling*. It is a very time-consuming activity for the head nurse or planner. There is no tool for checking whether the constructed schedule is optimal or not. The planners usually use straightforward constrains of work time and idle time for every employee. Some schedule editing features are often used, however, automatic tools that produce optimal solutions are not available due to the problem complexity.

Automation of the scheduling process is, however, a broadly researched field, as can be seen in the overview given in [7]. Automated approaches hold significant potential for improving the timetabling process and the quality of rosters. Mathematical or heuristic approaches can easily produce a number of solutions, they can report upon the quality of schedules and they can try to divide the work evenly among personnel. One of the most significant benefits of automating the personnel scheduling process is a very considerable time saving for the administrative staff involved.

Nowadays the frequent approaches to the problem of rostering use the benefits of heuristic and metaheuristic procedures, mathematical programming or goal programming. Some authors even tackle the problem using the methods artificial intelligence.

In this thesis, a new metaheuristic algorithm will be proposed. It will use an evolutionary algorithm to iteratively improve the current roster in order to find the optimal or near-optimal solutions. The proposed procedure will be tested on broadly used rostering benchmark instances and compared with the best existing solutions, as well as solutions found in [21].

The first chapter of this thesis is dedicated to the definition and representation of the rostering problem. The second chapter is about the state of art in (nurse) rostering. Common approaches used for solving the problem are introduced. The third chapter is dedicated to definition and presentation of metaheuristics. Those most related to this work are described and studied in detail. Chapter 4 explains the chosen approach and describes the used algorithm.

Chapter 5 is focused entirely on evaluation of the proposed algorithm and its modifications on various benchmark instances. The current benchmark datasets are shown as well as their current best known solutions (some of them proven optimal). The results of the proposed algorithm are discussed and compared with the best results and those in [21]. The efficiency and possible improvement of the algorithm is further discussed. The conclusion chapter recapitulates all accomplished results and observed facts.

1 ROSTERING

There are many variants of the problem of scheduling, whether it is from the managers point of view or the planners. For example *staffing* [7] tackles the problem of determination the ideal volume of qualified workforce. There are many factors that has to be taken into account such as skill categories, patient needs, working preferences, budgeting and type of employment contract, which affects the resulting workload. These long term decisions are usually made by the hospital management and it is an independent subject to explore before the rostering process starts.

Over the past 50 years of nurse rostering, some authors focused on *cyclical scheduling* [7], where each person works a cycle of n weeks, where the requirements follow a cyclical pattern. This approach has some serious drawbacks, but it is common if the day is divided in distinct shift types. Cyclical scheduling is also referred as 'fixed', while non-cyclical scheduling is sometimes called 'flexible'.
Other papers deal with the definition and representation of the problem itself. Whether to have specified shift times or allow the moving shifts that can overlap other shifts and depend on number of hours to be at work.

1.1 Representation

In this work, frame of exactly defined shift types with starting and ending times formulated in the instance definition is used. The rostering itself is nowadays understood as short-term timetabling part of the problem, where the shifts are typically assigned to the entire staff for a few weeks. The typical representation (see fig. 1.1) of a work roster is a two-dimensional table with employees as the rows and the scheduling period as the columns:

$$roster[employee, day] = shiftType \tag{1.1}$$

In 1.1 is an example of one cell of the roster with particular *shiftType* assigned to specific *employee* and *day*. The roster is filled with shift types for every day and every employee.

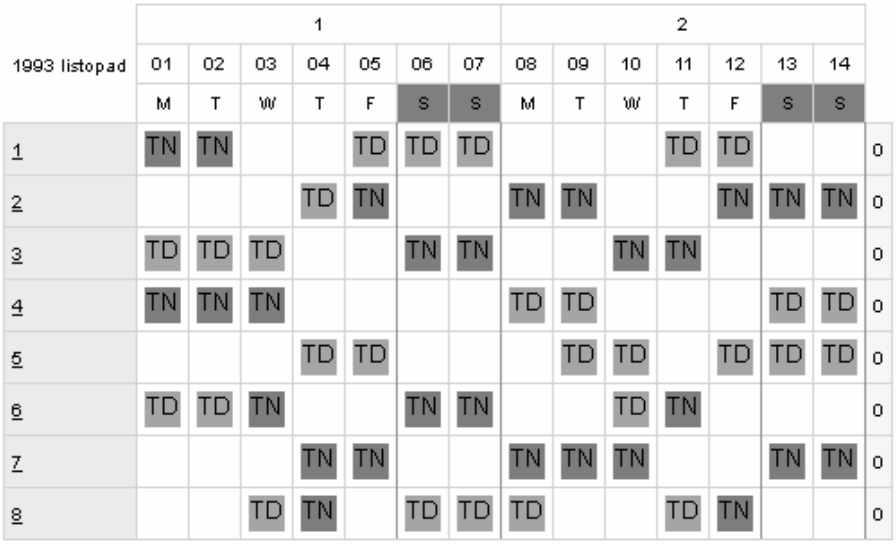

Fig. 1.1: An example of the roster for 8 employees, two weeks period of time and two types of shifts (TN,TD).

1.2 Constraints

For every problem instance there are constraints imposed on the roster that define their feasibility and quality. These constraints usually represents the requirements for the preferred workload during the scheduling period and they reflect the complete set of demands and requirements of the employer, employees and legal relations in the region of the hospital. In the rostering problem the constraints are usually divided into two groups. Soft constraints and hard constraints [6].

1.2.1 Hard Constraints

The hard constraints are those that must be satisfied at all cost. If one or more hard constraints are violated, the generated roster is not feasible and cannot be used as a final solution. Here are three main examples [6] of such constraints:

1. A nurse cannot be assigned more than one of the same shift type per day.
2. Shifts which require certain skills can only be covered by (i.e. assigned to) nurses who have those skills.
3. The shift coverage requirements must be satisfied. For example, if a certain day requires three night shifts then there must be three employees present at that time to work during that shift. Over coverage is not permitted.

1.2.2 Soft Constraints

The soft constraints, as the name suggests, are allowed to be violated at the cost of their penalty. Given that it is sometimes impossible to satisfy all constraints at once, it is better to violate some of the less severe ones. Here are some examples [6] of soft constraints:

- Maximum number of shifts worked during the scheduling period.
- Maximum and minimum number of hours worked during the scheduling period or per week.
- Maximum and minimum number of consecutive working days.
- Maximum and minimum number of consecutive non-working days.
- Maximum number of a specific shift type worked. For example, maximum zero night shifts for the planning period or a maximum of seven early shifts. This constraint can also be specified for each week. For example, a nurse may request no late shifts for a certain week.
- Maximum number of weekends worked in four weeks (a weekend definition is also a user definable parameter i.e. Friday and/or Monday may be considered as part of the weekend).
- Maximum number of consecutive weekends worked.
- No night shifts before a weekend off.
- No split weekends, i.e. shifts on all days of the weekend or no shifts over the weekend.
- Identical shift types over a weekend. For example, if a nurse has a morning shift on Saturday then he/she may prefer to have a morning shift on Sunday also.
- Minimum number of days off after night shifts.
- Valid numbers of consecutive shift types. For example, three or four consecutive early shifts may be valid but two or five consecutive early shifts may not.
- Shift type successions. For example, if shift rotation is allowed, is shift type A allowed to follow B the next day?
- Maximum total number of assignments for all Mondays, Tuesdays, Wednesdays... For example, a nurse may request not to work on Wednesdays or may require to work a maximum of two Tuesdays during the scheduling period.
- Avoid a secondary skill being used by a nurse. Sometimes a nurse may be able to cover a shift which requires a specific skill but they may be reluctant to do so as it is not their preferred duty. An example would be a head nurse not wanting to stand in for a regular nurse.
- Day on/off and shift on/off requests with associated priorities.

1.3 Roster Evaluation

The rostering problem is to find the best possible roster for the staff in a specific institution department or, in the case of the healthcare, a hospital ward. In order to find "the best" roster, we need to find a value representing the rosters quality. This can be done from two points of view, positive and negative. In the positive view we would assign the roster quality points for satisfied constraints and the best roster has naturally more points than the other generated solutions. On the other hand, the negative view gives penalization points for every constraint violation. The second view comes to mind as more natural and common due to precisely formulated set of constraints that are easy to be tested. Nowadays, we compare the rosters via their penalty, which is determined by the number and value of the violated constraints. Function that computes the total penalty of the solution S from the space of all solutions $\mathbb{S}$ is called an *objective function*. It is defined as follows:

$$ObjectiveFunction: \mathbb{S} \to \mathbb{N}$$
$$ObjectiveFunction(S) = \sum_{i=1}^{n} penalty(C_i) \cdot violations(C_i, S) \tag{1.2}$$

It is a sum of penalties of each of the n constraints C over their violation count in the current solution. Considering the size of the instances, number of constraints for each roster and number of changes in the roster to find a feasible solution, it it a crucial point for the objective function 1.2 to be simple and quick for the algorithm to run a reasonable amount of time. Note that even single employee's timetable can be viewed as a partial solution, thus the rosters total penalty is the sum of penalties over all employees timetables.

1.4 Roster variation operators

When modifying a roster, it is for the best if we change shifts without the violation of any constraint. Correcting the previously violated constraints at the cost of breaking the least severe ones is acceptable, too. In order to make feasible solutions the rosters are initialized randomly with respect to the hard constraints. The algorithm can then focus on changes that don't change the feasibility. This usually means that cover constraints are provided exactly as required for every day. Thus, if there has to be 3 day shifts and 2 night shifts every day of the workweek, there indeed are 3 day shifts and 2 night shifts assigned, while randomly distributed among the employees, for every day from Monday to Friday, during the initialization.

The changes in the roster are made not to break the provided cover. Note that deleting a shift or assigning new shifts can make the solution infeasible due to breaking the preferred number of employees at work. On the other hand, swapping a shift or even a block of shifts between two employees does not violate the cover constraints. It changes the roster in a way that it is easy to recalculate. The swap is defined as a roster operator with several arguments:

$$Swap(emlpoyee1, employee2, day, blockSize) \tag{1.3}$$

It swaps a block of shifts of the size of *blockSize* between *employee1* and *employee2* at the specified *day* of the planning period. The objective function just recalculates penalty of the two employees after the swap, then changes the total penalty accordingly.

2 STATE OF ART IN ROSTERING

In this section, I will present some of the main approaches to nurse rostering. For more information, I would recommend a work of Burke et al. [7] , where they took a closer look for all the past and current methods used in nurse rostering in an extensive paper about scheduling, from the various problem definitions over the staffing problem to the rostering itself.

2.1 Mathematical Programming

The optimizing approaches of mathematical programming are suitable for finding optimal solutions. In nurse rostering we often don't know how far from the optimal solution is the current solution. There are ways to determine the minimal value of the objective function. The only information we get from this information is that optimal solution cannot be lower than the said value. Furthermore the methods are simply not robust enough to deal with the enormity and complexity of the search spaces represented in modern rostering problems. They are often restricted to optimizing a single goal or criterion whereas nurse rostering works with relative ranking assigned to various goals. Still a decent number of articles were dedicated to optimizing methods and some smaller instances of the problem with a limited number of requirements were effectively solved [16] [17].

2.2 Goal Programming

Goal programming defines levels of each criterion and their relative priorities to achieve these goals. The method aims at finding a solution which is as close as possible to each of the targets in the order of the priorities given. Most of the papers apply mathematical programming but the latest modern research uses metaheuristics within a multi-objective framework [8].

2.3 Artificial intelligence methods

The problem was also tackled from the field of artificial intelligence as constraint satisfaction problem [11] (*CSP*) and multiple forms of case-based reasoning [18]. The problems are nevertheless very complex and tend to be over-constrained. By merging some constraints and eliminating interchangeable values, thus reducing the domains, was achieved higher quality solutions.

Even better solutions were achieved by forming a hierarchical CSP with a library of various search algorithms and constraint propagation techniques [9]. The case-based features were mostly created for a human planners to have user-interactive, integrated (staffing, rostering) decision support methodologies for the nurse rostering. However, some algorithms combining constraint networks and knowledge-based rules to solve employee timetabling problems were developed along the way [15]. Some algorithms even make use of pattern recognition and machine learning [12]. For example, they use a classifier of promising solutions based on a neural net rather than computing all the penalties, which tends to be computationally difficult.

2.4 Heuristics and Metaheuristics

The size of the rostering problems and the lack of knowledge about the structure of most of them interfere with the use of exact optimization methods. Therefore, many heuristic techniques were developed to obtain high quality feasible solutions within a reasonable time horizon.

There are many heuristics and metaheuristics, which can be further hybridized and combined together, to form better tools for facing complex problems of rostering. In order to not lose any part of already immense search space and because some instances of rostering are over-constrained, some of the heuristics count some hard constraints as soft constraints with great penalization. This is only natural because most of the heuristic methods are error driven and momentary violation of some hard constraint can be, at some points, contributing to the search for best solution.

Nowadays the most popular meta-heuristic algorithms in the field of nurse rostering are hill climbing algorithm (which is probably the most common algorithm to be hybridized with other heuristics), simulated annealing [4], tabu search [5], genetic programming [5], bee-colony optimization [20], ant colony system [2] or harmony search algorithm [3]. One of the most successful current heuristics used in rostering is variable depth search [6]. It is based on thought introduced in 1973 [13]. It was, however, used to tackle a different problem so it wasn't obvious to use this heuristic on the rostering problem, which by the way wasn't even as properly defined as it is today. Approaches that are most related to my work will be described in the following chapter about metaheuristics.

3 METAHEURISTICS

Metaheuristics are stochastic optimization techniques [14]. These algorithms employ some degree of randomness in the process of looking for an optimal or as good as possible solution to otherwise hard problems.
For some problems there is no exact algorithm that could optimally solve the instances of the problem in reasonable time horizon. We have only little heuristic information to go on and the exhaustive brute-force search is expelled due to the search space complexity. However, quality of the solution can be tested and that is the way the progress is measured. The metaheuristics are used for finding high quality solutions without the guarantee of finding the optimal solution. Metaheuristics find their use in combinatorial optimization [14] and tackle variety of large-scale problems such as the *Traveling salesman problem*.

3.1 Single-state algorithms

The single state methods work with one candidate solution at a time. Typically, they search a neighborhood of the working solution for the better one. If such solution is found, it replaces the current working solution.

3.1.1 Random search

The simplest algorithm that use random behavior and quality testing is random search. It randomly samples n solutions and returns the best one. In real life problems this tends to create very bad solutions. The pseudo-code for random search is in algorithm 3.1.

3.1.2 Hill-Climber

Better alternative to the Random search algorithm is Hill-Climbing (algorithm 3.2). It starts with a random solution and iteratively make small random modifications on it. If the modified solution is better than the current one, it is accepted as the new version of the current working solution, otherwise we throw it away and make another random change. This can be done until a feasible solution is found. Hill-climber exploits belief that similar solutions tend to behave similarly, so small changes will result in small well-behaved changes in quality, thus allowing the algorithm to "climb the hill" towards the good solutions. This belief is one of the defining features of metaheuristics. One can say that all single-state metaheuristics are essentially combination of hill-climbing and random search.

Algorithm 3.1 Random search algorithm

1: $S \leftarrow$ random initial solution
2: **repeat**
3: $\quad R \leftarrow GenerateRandomSolution()$
4: $\quad$ **if** $Quality(R) > Quality(S)$ **then**
5: $\quad\quad S \leftarrow R$
6: **until** S is the ideal solution or we ran out of time
7: **return** S

Algorithm 3.2 Hill-Climbing Algorithm

1: $S \leftarrow$ random initial solution
2: **repeat**
3: $\quad R \leftarrow Modify(Copy(S))$
4: $\quad$ **if** $Quality(R) > Quality(S)$ **then**
5: $\quad\quad S \leftarrow R$
6: **until** S is the optimal solution or we ran out of time
7: **return** S

3.1.3 Tabu search

Tabu Search employs a different approach to exploration. It keeps a history of recently considered candidate solutions (known as the tabu list) and refuses to return to those solutions until they are sufficiently far in the past. The simplest approach to Tabu Search is to maintain a tabu list of candidate solutions that has been visited so far. Whenever we adopt a new candidate solution, it goes in the tabu list. If the tabu list is too large, we remove the oldest candidate solution and it is no longer taboo to reconsider. Tabu Search is usually implemented as hill-climbing where the best of n solutions, produced by modification of the current solution, becomes the new current solution (it is as we've had a pool for choosing the 'gradient' direction in which to move). Pseudo-code is in algorithm 3.3

Algorithm 3.3 Tabu search algorithm

```
1: l ←desired maximal tabu list length
2: n ← number of solutions to sample the 'gradient'
3:
4: S ←random initial solution
5: Best ← S
6: L ← {} tabu list                    ▷ implemented as first in, first out queue
7: Enqueue S into L
8: repeat
9:     if Length(L) > l then
10:        Remove the oldest element from L
11:    R ← Modify(Copy(S))
12:    for n-1 times do
13:        W ← Modify(Copy(S))
14:        if W ∉ L and (Quality(W) > Quality(R) or R ∈ L) then
15:            R ← W
16:    if R ∉ L and Quality(R) > Quality(S) then
17:        S ← R
18:        Enqueue R into L
19:    if Quality(S) > Quality(Best) then
20:        Best ← S
21: until Best is the optimal solution or we ran out of time
22: return Best
```

3.1.4 Variable Depth Search

A variable depth search (VDS), also known as variable depth neighborhood search (VDNS), is a generalization of the local search method, which was first successfully applied by Lin and Kernighan [13] to the traveling salesman problem (TSP) and the graph partitioning problem. The main idea is to adaptively change the size of neighborhood so that it can effectively traverse larger search space while keeping the amount of computational time reasonable.

The neighborhood of some solution S is a space of all candidate solutions that can be generated by applying operator o on solution S. Different operators produce neighborhoods of different magnitudes. The 1-exchange neighborhood is a set of solutions where one variable/position is changed. Similarly, the 2-exchange neighborhood has swapped values of two variables/positions. In general, members of k-exchange neighborhood differ from current solution in k variables. Variable depth search (algorithm 3.4) explores neighborhoods of the current solution.

It starts from the smallest neighborhood to the largest one for each candidate solution and whenever a better solution is found it becomes the new candidate solution and the search continues from the smallest neighborhood again. When the 1-exchange neighborhood has no improving solution, algorithm will choose one neighboring solution (typically optimizing one part of our solution), which 1-exchange neighborhood is the deeper neighborhood for the current candidate solution. Whether the better solution was found or the depth reached its maximum value, the algorithm starts new search. The algorithm ends when the desired solution is obtained or the time expired.

This algorithm shows lot of similarities with previously mentioned Tabu-search. Instead of n solutions to sample the direction in which to move the search, it searches through entire one-step neighborhood of the current solution and it also keeps a list of forbidden moves in order to prevent the cyclical returning into previously visited solutions. Unlike tabu search, this procedure does not modify solutions randomly but rather systematically in order to find the best move. The variable in the name means that the method produces sequences of changes of variable length, depending on the contribution to the overall quality.

Algorithm 3.4 Variable Depth Search

```
1: maxDepth ← maximal depth of search
2: depth ← 0
3: Best ← some initial solution
4: S ← Best
5: L ← {S}            ▷ List of visited states for current solution, prevents cycling
6: repeat
7:     depth ← depth + 1
8:     W ← BestSolution({OneStepNeighborhood(S) \ L})
9:     if Quality(W) > Quality(Best) then
10:        Best ← W
11:        depth ← 0
12:        L ← Null                          ▷ Empty the list for another search
13:    else if depth = maxDepth then
14:        depth ← 0
15:        S ← Best
16:    else
17:        S ← W
18:    add W into L
19: until Best is the ideal solution or the time ran out
20: return Best
```

3.2 Evolutionary algorithms

Inspired by the nature and natural selection, evolutionary algorithms adopt the basic thoughts of evolution and apply it on various problems. As in the nature, only the strongest and those who are best adapted to the conditions survive and continue to reproduce, this rule of evolution is also called *survival of the fittest*. The same rule applies in the evolutionary algorithms. Only the best candidate solutions are kept in order to find the result. The greatest difference between evolutionary algorithms and previously mentioned metaheuristic methods is that they keep a sample of candidate solutions rather than just one. We call that set a *population*. First we need to introduce some of the terms used in evolutionary algorithms:

- ***individual*** - the candidate solution
- ***population*** - set of candidate solutions
- ***fitness*** - value representing the quality of an individual
- ***mutation*** - modification of a single individual, also referred to as 'asexual' breeding
- ***selection*** - picking individuals based on their fitness
- ***crossover*** - procedure creating one or two individuals (children) from parts of two other individuals (parents), referred to as 'sexual' breeding
- ***breeding*** - producing children from population of parents using iterated process of selection and crossover/mutation

3.2.1 Basic Evolutionary Algorithm

Basic structure of an evolutionary algorithm is that we generate an initial population of random solutions, then we evaluate their fitness. We then breed a new population from the old one using crossover, mutation and following the *survival of the fittest* rule. New population is again evaluated to determine the fitness of its members. The process continues to breed another generations until a termination criterion is met (either we found suitable solution or we ran out of time or predefined number of generations did not bring any improvement). The pseudo-code is in algorithm 3.5.

The Breed operation usually consist of two phases: *selection* of the parents from the old generation, and creating new population using some degree of *mutation* and *recombination*. The *Replacement()* operation represents the *survival of the fittest* rule. Either it replaces the parents with the children, or keeps the fit parents along with the newly formed individuals.

Algorithm 3.5 General Evolutionary algorithm

1: $P \leftarrow$ Initial population
2: $Best \leftarrow Null$
3: **repeat**
4: $\quad EvaluateFitness(P)$
5: $\quad$ **for all** $P_i \in P$ **do**
6: $\quad\quad$ **if** $Best = Null$ or $Fitness(P_i) > Fitness(Best)$ **then**
7: $\quad\quad\quad Best \leftarrow P_i$
8: $\quad P \leftarrow Replacement(P, Breed(P))$
9: **until** $Best$ is the ideal solution or we ran out of time
10: **return** $Best$

3.2.2 Mutation

As I have already mentioned, mutation is a modification of single individual. It is a useful tool for evolution because it keeps certain degree of diversity in the population. Mutated individual, represented as a string of characters or sequence of numbers (possibly bit vector), differs from the original at few positions, which has been changed. The number of positions determine the type of mutation used. One-point mutation changes only one value, whether k-point mutation changes k different points within an individual. With higher numbers of the parameter k, the mutation basically generates random individuals. That is why only one- or two-point mutations are most common, because it corresponds with the heuristic belief that similar solutions behave similarly, ergo larger interventions tend to spoil the achieved quality.

3.2.3 Crossover

Crossover or recombination takes two individuals and combines them into two children. Again the most general type is k-point crossover, but this time it divides the data vectors at k places into smaller sequences that are later swapped between the parents to form new individuals. One-point crossover divides the data vectors at specified position and swaps the first part of the first parent with the first part of the second individual. Two-point crossover has two division points resulting in 3 subsequences and thus swapping the central sequences. Generally after k-point crossover action, children are consisting of swapped and preserved data blocks. Again one- or two-point crossovers are most common.

3.2.4 Steady-state vs. Generative evolution

Evolutionary algorithms have two different approaches regarding the manipulation with the populations. The traditional generative model works always with old and new generation and then uses the *survival of the fittest* rule to substitute the weaker individuals in the new generation. The steady-state approach keeps one population of individuals and after every single child is bred, the algorithm chooses the individual to be replaced by the newborn, usually it is the individual with the lowest fitness.

Both approaches have their pros and cons, e.g. the steady-state algorithm tends to converge faster than the generative, that means the rate of convergence is higher for the fitness function in time, but at the cost of searching possibly smaller state space and stagnation in local optimum. The generative approach will then search through possibly larger state space at the cost of finding possibly similar solution as its faster sibling.

3.2.5 Selection

Evolutionary algorithm uses different selection of parents for breeding. Picking the n fittest individuals form previous generation is called *truncation selection*. This type of selection is rather simple but has a few shortcomings as it tends to decrease the diversity in the next generation. When using the *roulette selection*, each individual gets picked with the probability proportional to its fitness. The *tournament selection* works in a similar way, but here each parent is chosen from a small group of randomly selected individuals. As it comes naturally in mind, the fittest individual of each group is selected. The pseudo-code is in algorithm 3.6. The method is simple and tunable by the variation of the tournament size. If the size is 1 the tournament selection becomes random search, on the other hand if the size is very large the algorithm will chose the fittest individual each time (also known as *truncation selection*).

Algorithm 3.6 Tournament selection

```
1: P ← population
2: t ← tournament size, t ≥ 1
3: Best ← random individual from P
4: for t times do
5:     Next ← random individual from {P \ Best}
6:     if Fitness(Next) > Fitness(Best) then
7:         Best ← Next
8: return Best
```

4 PROPOSED APPROACH

In this chapter, the algorithm I designed for solving the nurse rostering problem will be introduced. The proposed name of the algorithm is **Evolutionary Lin-Kernighan heuristic** (ELK). All the aspects, such as the motivation for choosing the encountered problem, the original idea behind the realization and which algorithms were considered during the implementation, should be explained below.

4.1 Motivation and Inspiration

In the 4th semester of my studies at The Czech Technical University in Prague I just finished reading John R. Koza's book about genetic programming [10] as a preparation for one of my semester works. I was so enthusiastic about the whole concept of evolutionary algorithms and genetic programming that I decided to search for a topic for my Bachelor's thesis in this field of problem solving. My at that time supervisor-to-be Ing. Jiří Kubalík, Ph.D. proposed many problems that deserved attention, but the most lucrative seemed to be the nurse rostering problem. I was told to read some articles about the topic in order to find a possible use of the evolutionary algorithm. I learned that several studies tried solving the problem using evolution, but the results were of rather unsatisfactory. The best results were achieved through hybridizing the evolutionary metaheuristics with other heuristics such as local search [5]. That was the point where i decided to focus on some type of search that would implement evolution in order to achieve the best solutions.

The main inspiration came from the article about variable depth search algorithm published by Burke et al. in 2013 [6]. As mentioned before, variable depth search applies sequences of modifications of variable length at the current solution. This made me realize that i can randomly generate whole population of such sequences and using the traditional rules of evolution, breed sequences that make better solutions in order to search for the best candidate.

4.2 Algorithm

The algorithm is actually pretty simple. It works with swaps of shifts applied at an feasibly initialized roster. Each swap is performed between two employees at a specified day and it swaps a block of shifts of varying length.
The algorithm is chaining the swaps into sequences of predefined length and each *sequence's fitness* and *best point* is computed. The value of *sequence's fitness* is defined as value of the roster's penalty minus the best value of the roster's penalty during applying the sequence of swaps.

The *best point* of one sequence is the index of swap after which the sequence reached its fitness so the roster's penalty was lowest after the execution of swap at this particular position.

As in any other evolutionary algorithm there is generated a population of these sequences, their fitness is computed and then starts the process of breeding next generation. Some individuals are mutated, some crossbred and some survive to the next generation thanks to the *survival of the fittest* rule. After some number of generations that did not bred any better individual than the current best sequence, the best sequence is applied to the roster (usually not whole but all the swaps till the best point swap in current sequence) and thus starts the new round of evolutionary algorithm to find the next best sequence of swaps that improves the current roster. This happens until the termination criterion is met, for example we ran out of time or solution satisfies the constraints or a defined number of evolutions did not improve the best found solution. The pseudo-code is in algorithm 4.1.

Algorithm 4.1 Evolutionary Lin-Kernighan heuristic

```
1:  Roster ← initialized solution
2:  P ← {}                                  ▷ population of swap sequences
3:  Best ← Null                             ▷ best individual in population
4:  S ← Null                                ▷ current best individual in population
5:  n ← number of non-improving generations
6:  repeat
7:      P ← GenerateSwapSequences()
8:      EvaluateFitness(P)
9:      Best ← BestIndividual(P)
10:     while i < n do
11:         P ← Join(P, Breed(P))
12:         S ← BestIndividual(P)
13:         if Fitness(S) > Fitness(Best) then
14:             Best ← S
15:             i ← 0
16:         else
17:             i ← i + 1
18:     ApplySequenceAtRoster(Best, Roster)
19: until Roster is the ideal solution or the time ran out
20: return Roster
```

4.3 Implementation

Here I will describe the implementation of the proposed algorithm in detail, all the parts of the tool for solving the rostering problem.

4.3.1 Computational framework

First I'd like to mention that this work was designed within the framework designed by Burke et al. [6] and until recently was publicly available [1]. The framework is implemented in modern, general-purpose, object-oriented programming language C#, which is simple and convenient to use. There are many advantages of using this application interface as the manipulation with the rosters, constraints patterns and penalty evaluation is implemented. This prevents the errors that could occur when many researchers would implement the penalty evaluation by themselves. This way even if there is a mistake in fitness computation, all solutions are computed using the same evaluating software so it has no influence on the quality for comparison. One can easily concentrate on forming the actual method for solving the problem rather than implementing a whole framework. The resulting roster can be viewed in the html format and format for the freeware program Roster Booster [19], where the solutions can be viewed as well as modified and where the instances can be solved using implemented variable depth search [6] algorithm.

4.3.2 Initialization

The initial roster is generated from the cover constraints defined in each instance of the nurse rostering problem. First, preferred number of each shift type is detected for each day of the planning period. These shifts are then assigned at random to available employees for each day in the specified quantities. This procedure usually produces roster with total cover penalty equal to 0, but there are instances which use minimal and maximal number of assigned shifts or employee-specific cover constraints, where this type of initialization usually cannot produce satisfactory cover initialization. As the said initialization was satisfactory for majority of the rostering problem instances, I decided to use it for all encountered problems. The influence of the non-zero cover penalty will be discussed in the experiments chapter.

After the basic cover initialization, series of randomly generated swap sequences are evaluated and greedily applied in order to lower the initial roster's penalty. These quick changes save little time for the main algorithm computation and can be viewed as another part of initialization.

4.3.3 Generation of swap sequences

For each roster there was a population of 50 swap sequences generated in each step. The swap sequences had length of 10 swaps as it reached to the ten-exchange neighborhood of each roster. The length of the swap sequence usually determined the rate of convergence in the first parts of the algorithm where the solution is still poor and many swaps can effectively change the roster towards better results, however in the later time the 10-exchange neighborhood was more than enough to provide sufficiently large search space to traverse in every step of the solution method.

The swaps are related in a way that they are connected with each other. For example the first swap interchanged a block of shifts between employee A and B, the second swap then took employee B and swap a block of shifts between him or her and the employee C, the third swap took employee C and so on.

This type of swap execution originates from the self-scheduling where the planners usually make swap to improve one's timetable but the other side of the swap can be handicapped by this procedure so now it's this employee's timetable to be fixed. This thought is more or less the original idea of variable depth search. The creation of each swap (except the first one) is then realized as generating an employee (different from the previous one) with whom to swap shifts and a random number of of shifts that will be swapped. The day in which we swap this block of shifts is then found in order to maximize the fitness of the roster.

I designed three heuristics for choosing the day of each swap. First is to improve most of all the fitness of the first employee and expect the other swaps to make up for the potential worsening of the second participating employee. This is the original idea taken from the rostering practice. The second one is to improve at most both of the employees in total and the third one focus on most improving either one of the involved parties. The effectiveness of these heuristics will be further discussed in the chapter with experiments and results.

4.3.4 Evolutionary algorithm

As mentioned before, the input for the evolutionary algorithm was the population of 50 individuals which were then crossbred and mutated in order to create the next generation of swap sequences. I decided to test generative as well as the steady-state evolutionary approach, to determine the fitter solving procedure . The *survival of the fittest* rule was for the generative approach reduced. Only the best individual from the previous generation 'survived' and that happened only if there was no better individual in the new generation.

Each of the parents was selected via tournament selection of size 4. The parents were then crossbred or mutated with 50 percent chance. I used one-point mutation and one-point crossover in order to preserve the unique qualities of the individuals. There are two types of the one-point crossover applied at the parenting sequences. First version of crossover randomly chooses the position in sequence where the part of the second parent is placed, whereas the second version splits the individual at the best point of this sequence. This comes naturally in mind if we want to improve the best feature of each parent. I did not applied mutation on the children coming out of the crossover because some 'mutations' were made during the recombination to prevent auto-swapping and cyclical swaps in each sequence.

To ensure the progress in solution, at least one swap of each sequence must be executed in order to change the current roster. Some sequences do not change the fitness of the solution but they change the roster's configuration so the new state (and new neighborhood) of the solution is discovered. For the generative approach, if 20 consecutive generations of sequences did not produce better individual than the yet best found individual, the best individual is then applied as another modification of the roster. For the steady-state evolution the experimentally chosen limit for the number of newly created non-improving individuals is 150.

4.3.5 Stopping criterion

I did not establish any time after which the yet best solution has to be returned, because the main goal is to find the solution as good as possible. The stopping criterion for the algorithm was 30 (generative approach) and 50 (steady-state approach) evolution steps without an improvement of the overall fitness or (where it was possible) equalizing the proven optimum for the encountered instance of nurse rostering problem.

5 EXPERIMENTS

All rostering instances involved in the experiments were solved by 12 versions of the algorithm. For each evolutionary approach there are 2 versions of crossover and 3 heuristics for choosing the day in which each individual swap in sequence was executed. As the used methods have stochastic nature and some degree of randomness is involved, each computation was evaluated 10 times to find the best and the average solution. The experiments were conducted on my personal laptop which performance specification is:

- **CPU** - Intel Core 2 Duo T6500 2.10 GHz
- **RAM** - 3.50 GB
- **OS** - Windows XP 32-bit

5.1 Data

The data used for testing the solving algorithm are publicly available at the web of the University of Nottingham [1]. These collected benchmark instances include problems from various sources, including industrial collaborations and scientific publications. For each instance there is the best found solution and its penalty. If the penalty equals the lower bounds for the solution, it is marked as the proven optimum. All encountered problems and their specifications is in the table 5.1.

Tab. 5.1: Automated employee scheduling benchmark instances

Instance	Employees	Scheduling period (days)	Shift types	Best known solution
SINTEF	24	21	5	0*
MILLAR	8	14	2	0*
MILLAR-S	8	14	2	0*
MUSA	11	14	1	175*
LLR	27	7	3	301*
OZKARAHAN	14	7	2	0*
BCV-4.13.1	13	29	4	10*
GPOST	8	28	2	5*
VALOUXIS	16	28	3	20*
BCV-3.46.2	46	26	3	894*
ORTEC01	16	31	4	270*

*** Proven optimal**

5.2 Experimental setup

Here are listed the values of particular parameters used for solving the rostering instances:

- $nonImprovingGenerations_{generative}$: 20
- $nonImprovingSteps_{generative}$: 30
- $nonImprovingGenerations_{steady-state}$: 150
- $nonImprovingSteps_{steady-state}$: 50
- *populationSize*: 50
- *lengthOfSwapSequences*: 10
- *tournamentSize*: 4
- *crossoverMutationProbability*: 0.5

5.3 Experimental results

The following section contains the results accomplished from solving the benchmark instances. Each of the instances has different set of constraints with different weights, therefore some solutions appear much worse than others but its due to different scale of penalty. Some problems don't have satisfying solution due to the nature of the hard constraints which are not affected by the solving procedure. Heuristics that help choose the day of swap are properly named after the way they were used:

- **First best** Choosing day that will most improve the employee1 (1.3)
- **Both best** Choosing day that will improve at most both employees' timetables
- **Either best** Choosing day that will improve at most either employee

5.3.1 Best result comparison

In the tables 5.2 and 5.3 there are the best results that all the algorithms achieved. Note that all the versions achieved the lower bounds for some instances of the rostering problem. Some instances (Musa,Ozkarahan) had larger penalty due to the combination of minimal and preferred cover constraints which were not completely satisfied by the initialization procedure. Some (GPost) had larger penalties due to the different working contracts for different employees. These penalties could be eliminated by solving every skill group and workload group of employees separately. However, this further knowledge is not expected from the algorithm as it incorporates another problem of assigning appropriate number of shifts to each group of employees. The used heuristics for picking the day of each swap in generated sequence were about the same quality, although the *first best* heuristic found most of the algorithm's best solutions.

The other heuristics usually produced solutions of similar quality, as the procedures have stochastic nature and some amount of uncertainty is always present.

Tab. 5.2: Comparison of the best accomplished results for generative approach

Instance	Best solution	Heuristics					
		First best		Both best		Either best	
		r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	0	7	**4**	6	5	**4**	**4**
MILLAR	0	0	0	0	0	0	0
MILLAR-S	0	0	0	0	0	0	0
MUSA	175	**279**	282	304	311	284	294
LLR	301	301	301	301	301	301	301
OZKARAHAN	0	4300	**3900**	4500	4300	4300	4500
BCV-4.13.1	10	10	10	10	10	10	10
GPOST	5	228	**213**	220	426	224	1023
VALOUXIS	20	**240**	300	400	320	380	280
BCV-3.46.2	894	899	897	897	898	**895**	899
ORTEC01	270	**400**	1340	2380	470	1365	1365

r.p./b.p random/best point crossover

Tab. 5.3: Comparison of the best accomplished results for steady-state approach

Instance	Best solution	Heuristics					
		First best		Both best		Either best	
		r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	0	**4**	**4**	7	**4**	6	6
MILLAR	0	0	0	0	0	0	0
MILLAR-S	0	0	0	0	0	0	0
MUSA	175	**262**	272	277	277	274	269
LLR	301	301	301	301	301	301	302
OZKARAHAN	0	4500	4900	5100	5100	**4300**	**4300**
BCV-4.13.1	10	10	10	10	10	10	10
GPOST	5	**229**	447	419	415	697	423
VALOUXIS	20	**180**	300	400	260	300	240
BCV-3.46.2	894	**896**	897	**896**	897	899	899
ORTEC01	270	**370**	1430	1355	1355	1375	1440

r.p./b.p random/best point crossover

5.3.2 Average time comparison

As we can see from tables 5.4 and 5.5, where the average computation times can be found, the *first best* heuristic algorithm is the fastest one as it computes penalty of only one employee for picking a day from the planning period. The other two heuristics has to compute the change of fitness for both employees present at each swap.

As mentioned before, the steady-state evolutionary approach is faster than the generative, simply because it generates fewer solutions. The parameters could be set to generate about as much individuals as the generative approach, but the experimentally chosen parameters were satisfactory and higher values of the parameters meant increasing of the computational time rather than improving the solution.

5.3.3 Average penalty comparison

The average penalty of the found rosters, displayed in tables 5.6 and 5.7, prove that the used algorithms produce solutions of similar qualities and that the best average penalties are almost equally distributed among the proposed procedures over all tested rostering instances.

Tab. 5.4: Comparison of the average computational time for generative approach

Instance	Heuristics					
	First best		Both best		Either best	
	r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	**19m**	20m	28m	33m	34m	25m
MILLAR	2m 43s	**1m 47s**	2m 50s	3m 40s	4m 41s	2m 33s
MILLAR-S	43s	46s	**36s**	1m 3s	1m 27s	1m 16s
MUSA	**2m 20s**	2m 28s	3m 16s	3m 25s	4m 15s	3m 4s
LLR	**1m42s**	1m 51s	2m 49s	2m 58s	2m 28s	2m 29s
OZKARAHAN	**31s**	**31s**	40s	38s	39s	45s
BCV-4.13.1	**20m**	32m	37m	1h 4m	51m	40m
GPOST	45m	**38m**	1h 7m	53m	1h 23m	58m
VALOUXIS	**32m**	33m	1h 16m	59m	44m	53m
BCV-3.46.2	**1h 4m**	1h 8m	1h 18m	1h 36m	1h 57m	1h 50m
ORTEC01	**1h 25m**	1h 33m	2h 14m	2h 17m	2h 55m	3h 16m

r.p./b.p random/best point crossover

Tab. 5.5: Comparison of the average computational time for steady-state approach

Instance	Heuristics					
	First best		Both best		Either best	
	r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	8m 18s	**6m 26s**	12m 52s	7m 56s	10m 39s	11m 22s
MILLAR	1m 3s	50s	1m 28s	**26s**	1m 14s	38s
MILLAR-S	26s	**17s**	26s	13s	32s	36s
MUSA	1m 11s	**58s**	1m 15s	1m 14s	1m 54s	1m 34s
LLR	**43s**	47s	1m 10s	53s	1m	1m 9s
OZKARAHAN	**12s**	**12s**	14s	16s	18s	19s
BCV-4.13.1	9m 56s	**8m 32s**	18m	17m	23m	22m
GPOST	**15m**	26m	23m	21m	31m	27m
VALOUXIS	**14m**	17m	20m	26m	29m	21m
BCV-3.46.2	19m	**18m**	31m	23m	35m	43m
ORTEC01	**35m**	38m	1h 3m	58m	1h 14m	1h 3m

r.p./b.p random/best point crossover

Tab. 5.6: Comparison of the average penalty for generative approach

Instance	Best solution	Heuristics					
		First best		Both best		Either best	
		r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	0	10	**6**	8	7	7	7
MILLAR	0	50	34	62	**25**	60	17
MILLAR-S	0	0	0	0	0	0	0
MUSA	175	**301**	317	319	325	313	324
LLR	301	304	303	**302**	**302**	303	303
OZKARAHAN	0	5150	5053	5500	5100	**4977**	5575
BCV-4.13.1	10	**10**	**10**	12	11	11	11
GPOST	5	**767**	915	1052	1153	875	1673
VALOUXIS	20	428	**415**	514	380	511	**415**
BCV-3.46.2	894	901	900	902	**899**	901	901
ORTEC01	270	2085	2113	3151	**1411**	2016	2014

r.p./b.p random/best point crossover

Tab. 5.7: Comparison of the average penalty for steady-state approach

Instance	Best solution	Heuristics					
		First best		Both best		Either best	
		r. p.	b. p.	r. p.	b. p.	r. p.	b. p.
SINTEF	0	**8**	**8**	9	9	9	9
MILLAR	0	50	**11**	25	38	13	67
MILLAR-S	0	0	0	0	0	0	0
MUSA	175	308	318	322	308	**307**	308
LLR	301	303	303	**302**	**302**	304	303
OZKARAHAN	0	**5120**	5667	5575	5800	5440	5150
BCV-4.13.1	10	11	**10**	12	11	11	13
GPOST	5	1350	1685	1568	**1023**	1131	1289
VALOUXIS	20	418	414	483	417	**398**	400
BCV-3.46.2	894	902	901	901	901	901	**900**
ORTEC01	270	**2202**	2536	2254	2393	2272	2428

r.p./b.p random/best point crossover

5.3.4 Comparison with other methods

The results were compared with the best solutions ever found and with solutions found in [21]. In [21], several methods for solving the rostering problems were published, but the only instances I could convincingly compare were *Valouxis*, *Millar*, *Millar-s* and *Sintef*, as the other instances were modified and did not have the same set of constraints as they have now in the Nottingham's university framework [1]. In the tables 5.8, 5.9 and 5.10 you can see results of the ELK algorithm and those introduced in [21] and [6]. The proposed algorithms produced solutions of similar quality but the scatter search solved all the compared instances best. In the *Valouxis* example, which is the best instance for quality comparison among these instances, only scatter search method was better than here proposed algorithm. In the scatter search example, best solution is found at the expense of extremely high computational time. As we can see from the table 5.9, the proposed algorithm is rather slow in comparison to the other methods, as it is more complicated and requires more system resources for computation.

Tab. 5.8: Comparison of the best results for different methods

Instance	Best solution	Heuristics					
		ELK algorithm	Hill* Climber	Tabu* Search	TPVDS*[1]	Scatter* Search	Simulated* Annealing
SINTEF	0	4	**0**	10	**0**	**0**	21
MILLAR	0	0	0	0	0	0	0
MILLAR-S	0	0	0	0	0	0	0
VALOUXIS	20	180	260	640	220	**160**	620

[1] Time-predefined Variable Depth Search [6]

***** Algorithm used in [21]

Tab. 5.9: Average Time comparison for different methods

Instance	Heuristics						
	ELK		Hill* Climber	Tabu* Search	TPVDS*[1]	Scatter* Search	Simulated* Annealing
	steady st.	generative					
SINTEF	10m	26m	2m 36s	12m 50s	3m 25s	10m 52	**13s**
MILLAR	56s	3m 2s	5s	6s	3s	28s	**2s**
MILLAR-S	25s	58s	**1s**	**1s**	2s	**1s**	**1s**
VALOUXIS	21m	50m	3m 16s	9m 47s	20m	2h 26m	**9s**

[1] Time-predefined Variable Depth Search [6]

***** Algorithm used in [21]

Tab. 5.10: Average penalty comparison for different methods

Instance	Heuristics						
	ELK		Hill* Climber	Tabu* Search	TPVDS*[1]	Scatter* Search	Simulated* Annealing
	steady st.	generative					
SINTEF	8^{2}	6^{2}	19	14	8	**0**	60
MILLAR	11^{2}	25^{3}	265	315	388	**0**	218
MILLAR-S	**0**	**0**	73	5	58	**0**	22
VALOUXIS	398^{4}	415^{24}	401	3233	322	**225**	1857

[1] Time-predefined Variable Depth Search [6]

***** Algorithm used in [21]

[2] [3] [4] *First best / Both best / Either best* heuristic

5.4 Possible improvements

Some solutions might be better if the instance was divided and solved for each skill-type individually. Similar improvement would be achieved by individual solving of employees with full-time and part-time contracts. Another point in which the algorithm could be improved is the initialization of the rosters. Some instances were initialized with some cover constraints violated as the initializing procedure only assigned the preferred number of shifts for each day in random order. In those cases, an effective exploiting of the structure of the cover and other types of hard constraints would help divide the problem into simpler problems, removing some penalized constraint violations and thus producing solutions of higher quality.

6 CONCLUSION

The main purpose of this thesis was to propose, implement and test a new algorithm that uses evolutionary and other metaheuristics for solving rostering problems. To fulfill this goal, an algorithm that iteratively tries finding better solution was implemented. In each iteration, an evolutionary algorithm is triggered. The evolutionary algorithm searches for the most improving sequence of swaps (eq. 1.3), that will modify the current roster.

Several modifications of the algorithm were made that differ in the swap selecting (see sec.5.3) and population replacement strategy (see sec. 3.2).

The algorithm and its modifications were tested on benchmark instances [1]. The results were evaluated and compared with best known solutions and results found in [21]. The achieved results were satisfactory as they achieved proven optimum or near optimum fitness values for several instances. However, some instances could be solved better if the initialization was made more sophisticated and the problems were divided into subproblems. The computation times were usually higher than they were in the case of [21]. This can be explained by the complexity of evolutionary algorithm and its maintenance.

For the results to be even better, several ideas for possible improvement of the algorithm were proposed.

7 BIBLIOGRAPHY

[1] Automated employee scheduling benchmark instances. http://www.cs.nott.ac.uk/ tec/NRP/.

[2] Uwe Aickelin, Edmund K Burke, and Jingpeng Li. An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58(12):1574–1585, 2007.

[3] Mohammed A Awadallah, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, and Asaju La'aro Bolaji. Nurse rostering using modified harmony search algorithm. In *Swarm, Evolutionary, and Memetic Computing*, pages 27–37. Springer, 2011.

[4] Michael J Brusco and Larry W Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organizations. *European Journal of Operational Research*, 86(2):249–261, 1995.

[5] Edmund Burke, Peter Cowling, Patrick De Causmaecker, and GreetVanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, 2001.

[6] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, 25(3):411–419, 2013.

[7] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. of Scheduling*, 7(6):441–499, November 2004.

[8] Edmund K Burke, Patrick De Causmaecker, Sanja Petrovic, and G Vanden Berghe. A multi criteria meta-heuristic approach to nurse rostering. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1197–1202, 2002.

[9] H Meyerauf'm Hofe. Conplan/siedaplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *Conference Proceeding, PACT*, volume 97, pages 257–271, 1997.

[10] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[11] Haibing Li, Andrew Lim, and Brian Rodrigues. A hybrid ai approach for nurse rostering problem. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 730–735. ACM, 2003.

[12] Jingpeng Li, Edmund K Burke, and Rong Qu. A pattern recognition based intelligent search method and two assignment problem case studies. *Applied Intelligence*, 36(2):442–453, 2012.

[13] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[14] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[15] Amnon Meisels, Ehud Gudes, and Gadi Solotorevsky. Employee timetabling, constraint networks and knowledge-based rules: A mixed approach. In *Practice and Theory of Automated Timetabling*, pages 91–105. Springer, 1996.

[16] Harvey H Millar and Mona Kiragu. Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European journal of operational research*, 104(3):582–592, 1998.

[17] Margarida Moz and Margarida Vaz Pato. Solving the problem of rerostering nurse schedules with hard constraints: new multicommodity flow models. *Annals of Operations Research*, 128(1-4):179–197, 2004.

[18] Sanja Petrovic, Gareth Beddoe, and Greet Vanden Berghe. Storing and adapting repair experiences in employee rostering. In *Practice and Theory of Automated Timetabling IV*, pages 148–165. Springer, 2003.

[19] Staff Roster Solutions. Roster booster, http://www.staffrostersolutions.com/downloads.php.

[20] Nikola Todorovic and Sanja Petrovic. Bee colony optimization algorithm for nurse rostering. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 43(2):467–473, 2013.

[21] Roman Václavík. Algorithms for the rostering problems. Master's thesis, Czech Technical University in Prague, 2011.

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

NRP nurse rostering problem

ELK evolutionary Lin-Kernighan heuristic algorithm

TPVDS time-predefined variable depth search

$\mathbb{N}$ set of Natural numbers

CPU central processing unit, processor

RAM random access memory

OS operating system

CSP constraint satisfaction problem

CD CONTENT

- Documents: directory with Bachelor's thesis
- nrp-example: directory with computational framework,source codes and solutions
- nrp-example/Solvers/CVUTSolver: source codes of proposed algorithm
- nrp-example/html/data: directory with benchmark instances and their solutions