Parallel algorithms of timetable generation

Łukasz Antkowiak

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to XX weeks of full time studies.

Contact Information:
Author(s):
Łukasz Antkowiak 890506-P571
E-mail: luan12@student.bth.se

University advisor(s):
Prof. Bengt Carlsson
dr. inż Henryk Maciejewski
School of Computing
Faculty of Electronics

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Faculty of Electronics
Wrocław University of Technology
ul. Janiszewskiego 11/17
50-372 Wrocław
Poland
Internet : www.weka.pwr.wroc.pl
Phone : +48 71 320 26 00
Abstract

**Context.** Most of the problem of generating timetable for a school belongs to the class of NP-hard problems. Complexity and practical value makes this kind of problem interesting for parallel computing.

**Objectives.** This paper focuses on Class-Teacher problem with weighted time slots and proofs that it is NP-complete problem. Branch and bound scheme and two approaches to distribute the simulated annealing are proposed. Empirical evaluation of described methods is conducted in an elementary school computer laboratory.

**Methods.** Simulated annealing implementation described in literature are adapted for the problem, and prepared for execution in distributed systems. Empirical evaluation is conducted with the real data from Polish elementary school.

**Results.** Proposed branch and bound scheme scales nearly logarithmically with the number of nodes in computing cluster. Proposed parallel simulated annealing models tends to increase solution quality.

**Conclusions.** Despite a significant increase in computing power, computer laboratories are still unprepared for heavy computation. Proposed branch and bound method is infeasible with the real instances. Parallel Moves approach tends to provide better solution at the beginning of execution, but the Multiple Independent Runs approach outruns it after some time.

**Keywords:** school timetable, branch and bound, simulated annealing, parallel algorithms.
## Contents

Abstract i

1 Introduction 1
   1.1 School timetable problem . . . . . . . . . . . . . . . . . . . . . . 1
   1.2 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.3 Research question . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.4 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Problem description 4
   2.1 The Polish system of education . . . . . . . . . . . . . . . . . . 4
   2.2 Case study . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2.3 Problem definition . . . . . . . . . . . . . . . . . . . . . . . . . 6
   2.4 Problem NP-completeness . . . . . . . . . . . . . . . . . . . . . . 7
      2.4.1 NP set membership . . . . . . . . . . . . . . . . . . . . . . . 8
      2.4.2 Decisional version of DTTP . . . . . . . . . . . . . . . . . . 12
      2.4.3 Problem known to be NP-hard . . . . . . . . . . . . . . . . 12
      2.4.4 The instance transformation . . . . . . . . . . . . . . . . . 13
      2.4.5 Reduction validity . . . . . . . . . . . . . . . . . . . . . . . 14

3 Algorithms used with the Class-Teacher Problem 16
   3.1 Greedy randomized constructive procedure . . . . . . . . . . . . . 17
   3.2 Simulated Annealing . . . . . . . . . . . . . . . . . . . . . . . . 18
      3.2.1 Local search algorithm . . . . . . . . . . . . . . . . . . . . . 18
      3.2.2 Simulated Annealing concept . . . . . . . . . . . . . . . . . 18
      3.2.3 Class-Teacher Problem implementation . . . . . . . . . . . . 19
      3.2.4 Multiple independent runs . . . . . . . . . . . . . . . . . . 19
      3.2.5 Parallel Moves . . . . . . . . . . . . . . . . . . . . . . . . . 19
   3.3 Tabu search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   3.4 Genetic algorithms . . . . . . . . . . . . . . . . . . . . . . . . . 20
      3.4.1 Case from literature . . . . . . . . . . . . . . . . . . . . . . . 21
      3.4.2 Multiple independent runs . . . . . . . . . . . . . . . . . . . 22
      3.4.3 Master-slave model . . . . . . . . . . . . . . . . . . . . . . . 22
      3.4.4 Distributed model . . . . . . . . . . . . . . . . . . . . . . . . 22
   3.5 Branch & Bound . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
Chapter 1

Introduction

1.1 School timetable problem

Management of almost every school have to handle three types of resources: students, teachers and rooms. Upper regulations defines which students should be taught what subject and how much time should be used to do so, which means each student has a set of lessons to be provided for them. Beside students, every lesson also need one teacher and a room to take place. It is impossible for lessons to share a teacher or a room. Additionally it is highly unlikely for a teacher to be able to conduct all types of lessons.

The problem how to assign schedules for each student, teacher and room is called High school Timetable Problem (HSTP) [12]. In literature a simplified problem, where no rooms are taken into account during scheduling are named Class-Teacher Timetable Problem (CTTP) [15].

HSTP is very vague in its nature, because of the diversity of education systems. Changes usually affects the way that schedules are evaluated and compared. It is even possible that in one country a schedule may me concerned unfeasible while in other it would be a very good one.

Sometimes changes are even bigger, some countries use the idea of compulsory and optional subjects while in others all subject should be attended to. The other very important difference affects the concept of student groups. Students are the most numerous resource type that school needs to maintain. To simplify this task, students with the similar educational needs are grouped. Such a collections of students are called classes. In some countries, students in one class should have identical schedule, that means there is no possibility to split class members across different lessons. Although in majority of countries, classes are more flexible, there may be possibility to split members across two different lessons or event allocate the students freely between the different ones.

Due to the fact that the problem of timetable generation may be different in most of the countries a number of researchers have made attempts to gather such a differences. To name a few: Michael Marte described German school system in his paper [9] and Gerhard Post et al. [12] described scholar systems in Australia, England, Finland, Greece and The Netherlands.
Chapter 1. Introduction

This paper mainly focuses on Polish system of education that is very similar to the one used in Finland. For more elaborate description please refer to section 2.1

1.2 Background

The idea of automatic generation of timetable for a school is not a new one. There were several decades since it became interesting for researchers. Gotlieb’s article [6] from 1963 may be used as an example.

During these years a great deal of researches tried to solve that problem. Some papers aims only to prove that CTTP is NP-hard/complete[4, 15]. These pieces of evidence allowed researchers not to look for the optimal algorithms that works in a polynomial time and to design a heuristic algorithms. Although there are a papers describing rather simple algorithm based on greedy heuristics [16, 14], most of the researchers uses meta-heuristics to solve a CTTP problem. The evolutionary meta-heuristic are most popular[5, 17, 2], but a great deal of local search meta-heuristics were designed and evaluated too[1, 10].

Despite all the contribution mentioned above, the problem of automatic generation of an optimal timetable is still considered unsolved, mainly because of the need of conducting a lot of heavy computations. To speed calculations parallel algorithms were designed. Abramson designed two different parallel algorithms for this problem: genetic algorithm[2] and one based on simulated annealing[1].

Parallelization of meta-heuristics is a topic interesting not only for researchers working with CTTP. In his book[3], Enrique Alba gathered ways to introduce concurrency in classical meta-heuristics that were used in literature.

Unfortunately, most of algorithms proposed on literature were aimed to run on multi core shared memory systems, what makes their practical value very limited. Most schools can not afford such a machine for the issue that must be taken care of a few times a year. Especially when they already have quite a big IT infrastructure used to conduct casual computer classes.

1.3 Research question

The main goal of this paper is to investigate the potential to use a network based distributed systems to improve the final quality of timetables. To achieve it, the attempt to answer the following research question was made.

Research Question 1. Is a network based distributed system suitable for a parallel timetable constructing meta-heuristics in context of a polish elementary school?
1.4 Outline

The chapter 2 contains more elaborate description of the problem that the algorithms described in this paper solves. The practical information from the educational branch were provided such as the law regulations, the scale of the average school. In section 2.3 the formal definition of the problem is provided. In the last section of this chapter the problem are proven to be NP-complete.

In chapter 3 approaches found in literature are listed and described.

In chapter 4 the developed algorithms and the parallelization approaches are described.

In chapter 5 the results of an empirical experiment are analyzed. The conducted tests tries to help answer the question about proposed approaches scalability.
2.1 The Polish system of education

After comparing the description of the European education systems described by Gerhard Post et al.[12] and the author’s personal experience, the conclusion was made that the Finnish educational system is the most similar to the Polish one. In both, the main goal is to build a one-week schedule and use it throughout one semester. A valid lesson must have assigned a class, a teacher and a room allocated to it. Teachers are preassigned to most lessons, and rooms are assigned to lesson bearing teacher preference in mind.

In Poland, the concept of a teaching hour is introduced as a fixed period of 45 minutes. Although there is possibility for a lesson to last two subsequent teaching hours, the preference is for those single ones.

The concept of a student group is stronger than in Finland because in Poland the concept of optional courses is not used. There is still a possibility to split a student group into subgroups, especially when a subject requires less numerous groups i.e. a foreign language class.

Basically, there are two types of requirements. The first kind is commonly known as the hard constraints, according to which each schedule must abide by those requirements, otherwise it is illegal to use it. The second set contains the rules that are described in legal documents, but which do not have to be followed if deemed infeasible.

One of the most important is the regulation of the Minister of National Education[11], which states that the students’ workload should be balanced across week days.

The more accurate set of restrictions can be found in the internal regulation of Chief Sanitary Inspectorate[13]. Which states as follows:

1. Student’s workload

   (a) Student’s courses should start at the same time every day. If necessary, the difference should not exceed a two-hour limit.

   (b) A fixed number of classes a day ought not to be exceeded. This limit differs across classes.
(c) Student’s workload’s difference between two subsequent days should not exceed the limit of one hour.

2. Courses diversity

(a) If possible, every day there should be at least one course that requires physical activity.

(b) On the day of maximum workload, there should be a course that requires physical activity.

3. Multiple-period courses

(a) Two subsequent classes of one subject should not be scheduled; if necessary, only once a week.

(b) On Monday and Friday, two subsequent classes of one subject are allowed but only once a day.

2.2 Case study

By courtesy of one elementary school, a real instance of a school timetabling problem is available for an analysis. It is a perfect occasion to describe the needs of an average Polish school more completely.

In this rather small school, 12 groups are taught by 28 teachers in 12 multi-purpose and 6 dedicated rooms. Lessons take place five days a week and there are eight slots for classes a day.

Every room is available all the time throughout the week and so are the students. Two out of 28 teachers have restricted availability. What is more, one of those two teachers has only one hour a week marked as taken. There are no restrictions concerning the subjects: each lesson can take place at any time throughout a week. Although the majority of subjects need not preferred room, some of the subjects need dedicated classrooms, i.e. IT classes.

A group of students falls into subgroups for lessons that need less numerous attendance. There are exceptional cases when subgroups from different groups are combined.

There are 15 different subjects five of which need two subsequent classes once a week. Each group has its own subset of subjects that have to be taught. Because of the absence of the subject that needs more than one longer lesson, requirement 3a is met without any additional effort. Requirement 3b is not taken into account as the timetable used by the school violates it.

Additionally there is the restriction that no subsequent lessons may have similar subjects, i.e. different foreign languages. It helps to meet the requirement 2 from previous section.
Chapter 2. Problem description

2.3 Problem definition

For the scope of this paper a Class-Teacher Timetabling Problem will be used. It is assumed that there will be two types of resources to be taken into account in scheduling: teachers and students, which provides a synthetic reality of the school where there is always a room available to host the lesson. One may note from previous section that the number of classes is not bigger than the number of rooms, thus it is not a very unrealistic assumption.

Definition 2.3.1. Weighted time slots Class-Teacher problem with unavailabilitys - WTTTP Having

- set of teachers
  \[ T = \{t_1, \ldots, t_n\} \] (2.1)
- set of classes
  \[ C = \{c_1, \ldots, c_m\} \] (2.2)
- number of days
  \[ d_n \] (2.3)
- number of periods a day
  \[ p_n \] (2.4)
- set of periods
  \[ P = \{(d_i, p_j), d_i \in \{1, \ldots, d_n\}, p_j \in \{1, \ldots, p_n\}\} \] (2.5)
- function storing information whether the teacher \( T \) is available in period \( P \)
  \[ A_t : (T, P) \rightarrow \{0, 1\} \] (2.6)
- function storing information whether the class \( C \) is available in period \( C \)
  \[ A_c : (C, P) \rightarrow \{0, 1\} \] (2.7)
- function storing information about preference of time slot. The lower the value, the more preferred the time slot.
  \[ W : P \rightarrow \mathbb{R}^+ \] (2.8)
- function storing information about the number of lessons a particular teacher should give to a particular class
  \[ N : (T, C) \rightarrow \mathbb{N} \cup \{0\} \] (2.9)
The goal is to generate the most convenient timetable, that is function:

\[ S : (T, C, P) \rightarrow \{0, 1\} \quad (2.10) \]

That minimizes the objective:

\[ O = \sum_{p \in P} \sum_{t \in T} \sum_{c \in C} W(P) \cdot S(t, c, p) \quad (2.11) \]

Keeping following constraints:

**Constraint 2.3.1.** No teacher has more than one lesson in one time period

\[ \sum_{c \in C} S(t, c, p) \leq 1, \forall t \in T \forall p \in P \quad (2.12) \]

**Constraint 2.3.2.** No class has more than one lesson in one time period

\[ \sum_{t \in T} S(t, c, p) \leq 1, \forall c \in C \forall p \in P \quad (2.13) \]

**Constraint 2.3.3.** No teacher has lessons assigned during unavailability

\[ A_t(t, p) = 0 \Rightarrow \sum_{c \in C} S(t, c, p) = 0, \forall t \in T \forall p \in P \quad (2.14) \]

**Constraint 2.3.4.** No class has lessons assigned during unavailability

\[ A_c(c, p) = 0 \Rightarrow \sum_{t \in T} S(t, c, p) = 0, \forall c \in C \forall p \in P \quad (2.15) \]

**Constraint 2.3.5.** All lessons stated in matrix \( N \) are scheduled

\[ \sum_{p \in P} S(t, c, p) = N(t, c), \forall t \in T \forall c \in C \quad (2.16) \]

**Constraint 2.3.6.** Class cannot have free slots in their schedule

\[ (\exists t_k \in T S(t_k, c, (d_z, p_i)) = 1) \land (\exists t_k \in T S(t_k, c, (d_z, p_j)) = 1) \]

\[ \Rightarrow \forall p_k \in \{p_i, ..., p_j\} \quad (\exists \forall t_k \in T S(t_k, c, (d_z, p_k)) = 1) \]

\[ \forall c \in C \forall d_z \in \{1, ..., d_n\} \forall p_i \in \{1, ..., p_n - 1\} \forall p_j \in \{p_i + 1, ..., p_n\} \quad (2.17) \]

### 2.4 Problem NP-completeness

Because the proposed \( \text{WTTP} \) is a new problem, it has not been proven to be NP-complete yet. In this section the proof is presented. It is an essential step because it does not necessitate searching for polynomial optimal algorithms for this problem.

The proof will be conducted in two stages. In the first stage, \( \text{WTTP} \) will be shown to be an NP problem. In the second stage, the problem will be reduced to a simpler timetabling problem which is known to be NP-hard, in this case the Class-Teacher with teachers unavailabilities problem (see definition 2.4.2), which was proven to be NP-hard by Even S. in 1975[4].
2.4.1 NP set membership

To show that problem is in NP, the algorithm of evaluating and validating the already prepared schedule needs to be presented. The algorithm must check all the six constraints and evaluate the solution in polynomial time.

For clarity, it is assumed in this section that:

- \( d \) is the number of days,
- \( n \) is the number of periods in one day,
- \( t \) is the number of teachers,
- \( c \) is the number of classes

Additionally, \( p \) is the number of periods that may be computed by the multiplication of the number of days by the number of periods in one day;

Listing 2.1: Checking constraint 2.3.1

```python
for teacher in Teachers:
    for period in Periods:
        number = 0
        for class in Classes:
            if S(teacher, class, period):
                number += 1
        if number > 1:
            return False
    return True
```

Listing 2.2: Checking constraint 2.3.2

```python
for class in Classes:
    for period in Periods:
        number = 0
        for teacher in Teachers:
            if S(teacher, class, period):
                number += 1
        if number > 1:
            return False
    return True
```

The algorithms presented in listings 2.1 and 2.2 are very similar because of the similarity of constraints 2.3.1 and 2.3.2. Only the algorithm from listing 2.1 will be described.

This algorithm iterates through all possible combinations of one teacher and one period and checks whether constraint 2.3.1 is violated. If the requirement is not met at least in one combination of a teacher and a period, the algorithm
returns `False` as it has found the solution is not feasible. After checking all the possibilities the algorithm returns `True`, signaling the feasibility of the solution.

Checking whether a constraint is violated or not is achieved by iterating through all the possible classes and counting the lectures that have been assigned for a teacher in a given period. If the number of collected lectures is greater than 1, the constraint is considered violated. With the assumption that \( c \) is the number of classes that must be checked, that part of algorithm has complexity \( O(c) \).

Because the algorithm iterates through the sets of teachers in outer loop and the set of periods in inner loop, with the assumption that \( t \) is the number of teachers and \( p \) is the number of periods, the algorithm complexity is equal \( O(tpc) \). Due to their similarity, the complexity of both algorithms is equal.

*Listing 2.3:* Checking constraint 2.3.3

```python
for teacher in Teachers:
    for period in Periods:
        if not isAvailable(teacher, period):
            for class in Classes:
                if isLectureAssigned(teacher, class, period):
                    return False
    return True
```

*Listing 2.4:* Checking constraint 2.3.4

```python
for class in Classes:
    for period in Periods:
        if not isAvailable(class, period):
            for teacher in Teachers:
                if isLectureAssigned(teacher, class, period):
                    return False
    return True
```

The algorithms presented in listing 2.3 and 2.4 check if there are some lectures assigned to teachers or classes in periods when they have been considered unavailable. Because of the similarity of the solved problems, only the first algorithm will be analyzed.

To properly validate the solution according to constraint 2.3.3, the algorithm iterates through all the possible combinations of a teacher and a period. If the teacher is marked as unavailable, it checks whether the teacher has been assigned lessons in a forbidden time slot. Checking requests iteration through all the possible classes so it can be done with the complexity of \( O(c) \). As such a process needs to be done for all the teachers and periods of time, the complexity of this algorithm may be described as \( O(tpc) \). Due to the similarity, the second algorithm has the same complexity.
Chapter 2. Problem description

Listing 2.5: Checking constraint 2.3.5

```python
for class in Classes:
    for teacher in Teachers:
        number = 0
        for period in Periods:
            if S(teacher, class, period):
                number += 1
        if number != N(class, teacher):
            return False
return True
```

The algorithm presented on listing 5 validates the solution with respect to constraint 2.3.5, that is to assure that the schedule contains all the needed lectures. The method iterates through each combination of a teacher and a class and tries to count the lessons that are allocated to them. If the number of allocated lessons differs from the number stated in instance of the problem, the algorithm returns False.

The process of counting allocated lessons performs an iteration through all the periods, thus it has complexity $O(p)$. Such computation must be done for each combination of a teacher and a class, so the overall complexity of the method is equal $O(tcp)$.

Listing 2.6: Checking constraint 2.3.6

```python
def isScheduled(class, period):
    for teacher in Teachers:
        if S(teacher, class, period):
            return True
    return False

for class in Classes:
    for day in range(1,d_n):
        first_period = 0
        for period in period(1,p_n):
            if isScheduled(class,(day,period)):
                first_period = period
                break
        if first_period = 0:
            break
        last_period = 0
        for period in period(d_n,first_period):
            if isScheduled(class,(day,period)):
                last_period = period
                break
```
for period in period(first_period, last_period):
    if not isScheduled(class, (day, period)):
        return False
return True

Constraint 2.3.6 is the most complex one. To keep the method clear, an auxiliary sub-procedure was proposed isScheduled. The procedure takes a class and a period and checks whether a class has a lesson assigned in a provided period of time. To achieve this, an iteration through the set of all the teachers is executed. The complexity of this auxiliary function is equal $O(t)$

The proposed method iterates through the set of all classes and checks if the constraint is kept for each possible day. At the beginning, the algorithm search for the first assigned time slot, if there is no such one the iteration may continue. If a class has an allocated time slots in the analyzed day algorithm looks for the last used period. At the end of the iteration algorithm checks whether all time slots between the first and the last used are used too. If it is not the case, the algorithm ends immediately marking the solution unfeasible.

All from these three steps of the outer iteration needs to check all possible periods in a day. Because checking each period needs a helper sub-procedure to be executed, the complexity of one iteration is equal $O(nt)$

The iteration is executed for all combinations of classes and days so overall complexity of the algorithm is equal $O(cdnt) = O(cpt)$

**Listing 2.7:** Checking objective

```python
objective = 0
for period in Periods:
    number = 0
    for teacher in Teachers:
        for class in Classes:
            if S(teacher, class, period):
                number += 1
        objective += number * W(period)
return objective
```

The method shown in listing 2.7 evaluates the objective value for the provided solution. The algorithm iterates through the set of all periods. In each iteration it iterates through the every combination of a teacher and a class.

In inner loop only checking whether lesson is assigned and incrementation is executed so the complexity of this part of the algorithm may be shown as $O(1)$. Because of the nested iteration its complexity is equal $O(tc)$. The outer loop is executed for each period so overall complexity of the algorithm is equal $O(tcp)$

Despite the fact that each one from the set of the proposed solution must be executed in order to validate and evaluate solution, every one of them have the same complexity $O(cpt)$. As this is a polynomial complexity it is shown that the
Chapter 2. Problem description

2.4.2 Decisional version of DTTP

In the second stage of the NP-hard problem proof a reduction of the problem to other one already known to be NP-hard should be described. For the need of this section the objective function for the problem will be relaxed.

Definition 2.4.1. Decisional version of weighted time slots Class-Teacher problem with unavailabilities - DWTTP

Having all the data from the definition 2.3.1 With the additional value

\[ K \in \mathbb{R} \]

The goal is to determine whether there exist a schedule:

\[ S : (T, C, P) \rightarrow \{0, 1\} \]

That keeps all the constraint from definition 2.3.1, with the additional one:

Constraint 2.4.1.

\[ \sum_{p \in P} \sum_{t \in T} \sum_{c \in C} W(P) \cdot S(t, c, p) \leq K \]  

2.4.3 Problem known to be NP-hard

Definition 2.4.2. Class-Teacher with teachers unavailabilities problem - TTP

Having:

- a finite set of periods in the week

\[ \hat{H} \]

- A collection of teachers’ availabilities.

\[ \hat{T} = \{\hat{T}_1, \ldots, \hat{T}_n\}, \hat{T}_i \subseteq \hat{H} \]

- A collection of classes’ availabilities.

\[ \hat{C} = \{\hat{C}_1, \ldots, \hat{C}_m\}, \hat{C}_i \subseteq \hat{H} \]

- A matrix storing non-negative integers, where \( r_{ij} \) represents the number of lectures that have to be conducted by the teacher \( i \) to the class \( j \).

\[ R_{n \times m} = r_{ij}, r_{ij} \in \mathbb{N} \]
The goal is to determine whether there exists a schedule presented as a meeting function
\[ f(i, j, h) : \{1, \ldots, n\} \times \{1, \ldots, m\} \times \hat{H} \rightarrow \{0, 1\} \] (2.25)
where \( f(i, j, h) = 1 \) if and only if teacher \( i \) and class \( j \) have an assigned lecture during the period \( h \).

Keeping the following constraint:

**Constraint 2.4.2.** Lectures may take place only when both teacher and class are available
\[ f(i, j, h) = 1 \Rightarrow h \in \hat{T}_i \cap \hat{C}_j \] (2.26)

**Constraint 2.4.3.** The number of lectures given by teacher \( i \) to the class \( j \) must be equal to the number given in \( \hat{R}_{ij} \)
\[ \sum_{h \in \hat{H}} \hat{f}(i, j, h) = r_{ij}, \forall 1 \leq i \leq n \forall 1 \leq j \leq m \] (2.27)

**Constraint 2.4.4.** The class can not have more than one lecture assigned in one period of time
\[ \sum_{i=1}^{n} \hat{f}(i, j, h) \leq 1, \forall 1 \leq j \leq m \forall h \in \hat{H} \] (2.28)

**Constraint 2.4.5.** The teacher can not have more than one lecture assigned in one period of time
\[ \sum_{j=1}^{m} \hat{f}(i, j, h) \leq 1, \forall 1 \leq i \leq n \forall h \in \hat{H} \] (2.29)

The TTP was shown to be NP-complete [4].

### 2.4.4 The instance transformation

**Definition 2.4.3.** Transformation from TTP to the DWTP

The instance of the problem that is to be proven may be create from a TTP instance with the following procedure:

\[ T = \{t_1, \ldots, t_n\}, n = |\hat{T}| \] (2.30)

\[ C = \{c_1, \ldots, c_m\}, m = |\hat{C}| \] (2.31)

\[ d_n = |\hat{H}| \] (2.32)

\[ p_n = 1 \] (2.33)

\[ P = \{(d_i, p_j), d_i \in \{1, \ldots, d_n\}, p_j \in \{1, \ldots, p_n\}\} \] (2.34)
Chapter 2. Problem description

\[ h \in \hat{T}_i \iff A_t(t_i, (h, 1)) = 1, \forall 1 \leq h \leq |\hat{H}|, \forall 1 \leq i \leq n \]
(2.35)

\[ h \in \hat{C}_j \iff A_c(c_j, (h, 1)) = 1, \forall 1 \leq h \leq |\hat{H}|, \forall 1 \leq j \leq m \]
(2.36)

\[ W(p) = 0, \forall p \in P \]
(2.37)

\[ N(T_i, C_j) = \hat{R}_{ij}, \forall 1 \leq i \leq n, \forall 1 \leq j \leq m \]
(2.38)

\[ K = 0 \]
(2.39)

The result may be translated with the following procedure:

\[ S(t_i, c_j, (h, 1)) \iff f(i, j, h), \forall 1 \leq i \leq n, \forall 1 \leq j \leq m, \forall 1 \leq h \leq |H| \]
(2.40)

Additional comment may be useful for steps presented in equations 2.32 and 2.33. The TTP does not use the concept of time slots splitted by days, and none of constraints uses the relations between any time slots. The approach to allocate each period on different day in a DWTTP instance aims to relax the constraint 2.3.6 that is not used in TTP.

As the transformation contains only assignments it is clear that it can be conducted in polynomial time.

2.4.5 Reduction validity

It is crucial to show that if \( I \) is an instance of TTP and \( \hat{I} \) is \( I \) transformed to an instance of DWTTP, both algorithms provide the same answers:

\[ TTP(I) \iff DWTTP(\hat{I}) \]
(2.41)

Both the TTP and the DWTTP will return True only and only if all constraint are satisfied. Due to the similarity of the problem it is trivial to show that:

- Constraint 2.4.2 is equivalent to the constraints 2.3.3 and 2.3.4.
- Constraint 2.4.4 is equivalent to the constraint 2.3.1
- Constraint 2.4.5 is equivalent to the constraint 2.3.2
- Constraint 2.4.3 is equivalent to the constraints 2.3.5

Due to this fact, if at least one of the constraint of TTP is not satisfied, there will be at least one unsatisfied constraint for DWTTP problem. Thus the following implication is always satisfied:

\[ \neg TTP(I) \Rightarrow \neg DWTTP(\hat{I}) \]
(2.42)

If all constraints of TTP problem are satisfied, constraints 2.3.3, 2.3.4, 2.3.1, 2.3.2 and 2.3.5 will be satisfied too. The two additional constraints will be satisfied for each transformed instance, what is shown in theorems 2.4.1 and 2.4.2. Thus the following equation is always satisfied:

\[ TTP(I) \Rightarrow DWTTP(\hat{I}) \]
(2.43)
Theorem 2.4.1. The constraint 2.3.6 is satisfied for each instance transformed with the use of transformation from definition 2.4.3.

In equation 2.33 the periods number in a day is set to 1. Thus even if the first part of the antecedent is satisfied there is no next period in a day to satisfy the second part. That is the antecedent is always unsatisfied. There is no need for checking the consequent in such a case. The constraint is always satisfied.

Theorem 2.4.2. The constraint 2.4.1 is satisfied for each instance transformed with the use of transformation from definition 2.4.3.

In equation 2.37 the weights of all periods are set to 0. The multiplication of 0 will always return 0 and the sum of those multiplications will be equal 0 too. Because the objective value is set to 0 (see equation 2.39) the constraint will be always satisfied.

As with the use of proposed instance transformation TTP can be solved with the use of DWTTP, the second problem is at least as hard as the first one. Because the TTP is proven to be NP-hard, DWTTP is NP-hard as well. In the first stage the polynomial algorithm of checking validity of provided DWTTP solution were proposed, what means that it belongs to the set of NP problems. This makes the DWTTP a NP-complete problem.
Chapter 3

Algorithms used with the Class-Teacher Problem

As was mentioned, the researchers have been trying to solve the CTTP for a number of decades, putting forward a number of ideas in literature. In this part of the paper, a short list of algorithms used to solve the CTTP is presented.

Despite the fact that the algorithms presented in this chapter have completely different natures, almost every implementation from literature uses the same concept of storing the solution in memory. The most common way is to maintain an array of lecture lists (one list of lectures for each time period). The lectures stored in lists are usually represented by some kind of tuples that store information about the type of lecture and needed resources, such as a teacher running a lecture or a teaching class. The concept of this type of solution storage is shown in figure 3.1.

Figure 3.1: Storage of the solution concept
3.1 Greedy randomized constructive procedure

The GRCP was used to create initial solution in [14] for a tabu-search heuristic. In this section, the concept of GRPC will be briefly introduced. For more elaborate description, please refer to [8].

This heuristic is usually used in a Greedy randomized adaptive search procedure - GRASP, as a procedure that creates an initial solution for more complex local search heuristic. Both algorithms are executed multiple times and the best found solution is used. The algorithms are executed multiple times to reduce the risk of the simulated annealing to be trapped in a local optimum. To avoid it, GRPC must not be deterministic or it must be possible to provide some way to change the result between multiple executions. Usually the result of this algorithm must be a feasible solution, because most local-search algorithms expect that from the initial solution.

The GRPC is usually implemented with the use of dynamic algorithm, which builds the solution from scratch step by step trying to keep the partial solution feasible. At each iteration, it builds the candidate list, storing elements that may be used in the solution, and thus keeps it feasible. The list may be additionally restricted depending on the implementation. One element from the list is randomly chosen and used to build the solution.

**Listing 3.1:** High level GRCP implementation

```python
1 Solution = None
2 while isComplete(Solution):
3     candidateList = buildCandidateList()
4     restrictedCandidateList = restrictList(candidateList)
5     candidate = pickRandomly(restrictedCandidateList)
6     Solution = Solution + candidate
7 return Solution
```

The listing 3.1 shows the model of the GRCP.

In Santos’s paper [14], the candidate list stores all the lessons that can be assigned to the solution without losing feasibility. For each candidate, an urgency indicator is computed by evaluating the number of possible time slots that each lecture can be assigned to. The lower the computed value, the more urgent the candidate.

In the second step, the candidate list is restricted with the use of the parameter $p \in (0, 1)$. This parameter defines the length of a restricted list. For 0, list contains only one lesson, for 1, no restriction will be executed at all. Consequently, for $p = 0$ whole algorithm degrades to the greedy heuristic and for $p = 1$ the list is not restricted at all and the algorithm may pick from the set of all possible lectures.

In Santos’ work, it is assumed that the algorithm is not only used to create initial solution but also create neighbor solution. Please refer to section 3.3.
3.2 Simulated Annealing

Simulated annealing is a local-search meta-heuristics that takes inspiration from annealing in metallurgy. The annealing is a process of heating and controlled cooling of a material to improve its quality.

3.2.1 Local search algorithm

As a local-search algorithm, it needs an initial solution to be provided to begin. It iterates number of times using a neighbor generation method that creates a solution basing on the already existing one. The simplest local-search algorithm simply checks whether the new solution is better than the already known one and uses the better one. This type of algorithm tends to get stuck in local optimum. A set of meta-heuristic is known for changing this algorithm to avoid this disadvantage (simulated annealing is one of the examples).

A number of ending criteria types are known for this type of algorithms. To name a few:

- Algorithm iterates a fixed number of time
- Solution did not improved during last \( n \) iterations
- After a number of successful iteration
- The quality of known solution satisfies the needs

3.2.2 Simulated Annealing concept

The simulated annealing allows the standard local-search algorithm to pick a worse solution. This approach allows the algorithm to escape from the local optimum. The possibility that a worse solution is accepted decreases with the time of the algorithm. This is exactly the main idea of annealing in metallurgy.

High temperature give the atoms possibility to change their positions. As the temperature decreases, these changes are less likely to happen. It was empirically tested that atoms with the ability to change position find the equilibrium which increases the features of material.

The local-search algorithm begins with an initial temperature that is gradually decreased. In most implementation there are a number of levels and once some criteria are satisfied, the temperature is changed. The criteria that change the temperature level are similar to the ending criteria of plain local-search algorithm. The simulated annealing usually ends when it is no longer possible to decrease the temperature.
3.2.3 Class-Teacher Problem implementation

The simulated annealing was chosen by Abramson [1] to solve the TTP\(^1\). His implementation always picks the new solution if its quality is better than current one. If the solution is worse, the probability of picking equals:

\[
P(\Delta c) = e^{(-\Delta c/T)}
\]  \hspace{1cm} (3.1)

The temperature level is changed after a fixed number of successful swaps. The solution is stored in a set of lesson lists (one list for each time period).

The neighbor is generated by picking a lesson from one period and moving it to another one. Both periods are picked randomly. Only the difference in schedule cost is evaluated by removing the cost of the element removed and adding the cost of added element.

The authors emphasized the importance of not swapping two lectures across two periods but simply moving one. Unlike the first approach, this one allows to change the number of lectures in particular period.

3.2.4 Multiple independent runs

One of the ways to distribute the SA is called Multiple Independent Runs MIR\(^2\). Its biggest advantage is simplicity. It does not try to reduce the amount of time taken by one iteration. Instead, increases the number of iteration that can be executed simultaneously. Each node in a computing cluster executes an independent instance of Simulated Annealing. Once all nodes have finished, one master node collects the results and picks the best one.

It is a good idea to provide different initial solutions to each algorithm instance. Stochastic nature of the GRASP\(^2\) make it a perfect choice for this task. But due to the fact that simulated annealing is not deterministic, such an approach is redundant. Even when all instances use the same initial solution, one may expect improved quality after the same execution time.

3.2.5 Parallel Moves

Parallel Moves is the second way to distribute the SA attempts to reduce the time taken by one iteration.\(^3\) As the process of generating and evaluating new solution usually takes most of the time, that part of the algorithm is usually distributed. This parallelization approach is much more complicated, thus it is not used in literature as often as the MIR.

Figure 3.5 shows the basic idea of parallel moves. The main instance of Simulated Annealing is executed only on the master node. When there is the need

\(^1\)see definition 2.4.2  
\(^2\)see section 3.1 for additional information
to generate or/and evaluate the solution, a distributed sub procedure is executed with the use of the whole cluster.

### 3.3 Tabu search

Tabu search is another type of local-search meta-heuristic\(^3\)[8]. It improves the standard local-search to reduce the chance of being stuck in a local optimum.

Unlike the Simulated Annealing, Tabu Search does not use probability so extensively. It evaluates every solution in neighborhood and always picks the best one. The algorithm keeps a tabu list (a list of moves that are not permitted), to minimize the time spent in a local minimum. After each move used, it is added to the tabu list to prevent the algorithm from going back. The tabu list is a short-term memory. After spending a tenure (the amount of time provided as the execution parameter) the move is removed from tabu list.

To assure that the optimal solution will not be prevented by the tabu list, the concept of aspiration criteria is introduced. If the solution satisfies the aspiration criteria, the solution can be used even if it is created by a prohibited move.

Usually all the short time memory is not enough and algorithms may get stuck around local optimum. A diversification strategy is needed to allow the algorithm to escape from a local minimum. The long time memory is often maintained in order to be used for such a case.

In order to get familiar with the Tabu-search implementation for the Timetabling problem, one may refer to [14]

### 3.4 Genetic algorithms

Genetic algorithms are inspired by the way that nature searches for better solutions. The algorithm maintains the population that is a relatively large set of feasible solutions and introduces an iterative evolution process. In each step, the whole population is changed in such a way that tries to simulate the real file generations.

The already known solutions acts as parents to the new ones. After each iteration, some part of population must be dismissed. The better the solution, the bigger the probability it will survive. The average quality of the elements in next population should have the tendency to increase.

Usually the genetic algorithms use two kinds of methods to generate new solutions from the already known ones.

- Crossover

  This set collects methods that take at least two solutions and mixes them to get the new one.

\(^3\)see section 3.2.1
Chapter 3. Algorithms used with the Class-Teacher Problem

• Mutation

This type of methods aims to assure that algorithm does not get stuck in some part of the search space. Such a situation might happen if elements in population are identical or very similar.

Additionally, it is important to define how the algorithm that decides which elements of population should survive the iteration and which not works. Usually, the probability is computed with the use of the difference between the best already known solution or the best in a solution. Sometimes small differences are introduced. One example of this approach might be assuring that the best element of the population survives to the next generation.

3.4.1 Case from literature

Genetic algorithms are challenging for the scheduling problems. The most demanding part of the algorithm is the way the crossover works. Usually, there is no simple way to mix two solutions. There are a lot of dependencies between different parts of the solutions, and naive merging will most likely result in constructing an unfeasible one. This section describes an example of genetic algorithm design for the Timetabling problem used by Ambrason and Abela[2].

The solution is represented as a set of lists, each of which stores lectures allocated for a particular period of time. This approach allows to compute the weight of the solution and reduce the need to analyze the whole solution each time it is modified.

Crossover works for every period independently. Algorithms split the lectures of the first parent into two sets. The first set is added to the result. In the next step, a number of the second parent’s lectures is dropped. Authors used the number of lectures from the first parent. That reduced list is added to the final result. In situation where there are more elements from the first parent than from the second one, only the lecture from the first parent is used as a result. One may note that the solution still may be changed (the number of lectures may be reduced).

This approach has two major drawbacks.

• Naive joining two sets of lectures may result in lectures duplication, which makes the solution unfeasible.

• Reduction in the number of elements without assigning them to other parts of solution makes solution incomplete, thus resulting in its inability to be used as a full result of algorithm.

The authors proposed, a solution-fixing method, dropping duplicated lectures and randomly assigning those not yet allocated.
3.4.2 Multiple independent runs

Due to the probabilistic nature, the quality of the genetic algorithm results may differ between two independent runs on the same input. This allows to use the Multiple Independent Runs technique to increase the quality of a solution without an increase in the amount of needed time. Please refer to section 3.2.4 for description of this approach to simulated annealing.

3.4.3 Master-slave model

The main idea of the genetic concept allows to conduct a big amount of computations in parallel. The process of construction and evaluation of each solution from the new population may be done independently of other items[3].

This approach assumes that if one of the nodes manages the whole process, it is called a master node. Other nodes, slaves, waits for the task from the master.

The implementation of the genetic algorithm in the master-slave model is achieved by executing only one instance of the genetic algorithm on master. The master takes care of the flow of the algorithm, scheduling tasks to other nodes in a cluster.

Whether generation of the initial solution is time consuming or not, the workers may be asked to carry out this computation or the master may conduct it at the beginning of the algorithm. The master picks the solutions that are meant to be crossover and send them to a slave. This process is multiplied to create as many new solution as possible to construct a new population. Some kind of load balancing between the nodes may be introduced. The main node makes a decision about which elements are placed in a new population and checks if the algorithm may end the iteration.

This approach has a few flaws. One of them is the fact that the cluster must be completely serialized after each iteration. This may make the algorithm sensible for the difference in speed of the computation between the nodes if there is no load balancing used while task are ordered.

3.4.4 Distributed model

Distributed model executes one instance of the genetic algorithm on each node. Despite its similarity to the Multiple Independent Runs technique, the nodes communicate with themselves with the use of the migration concept. An individual from one population may occasionally be sent to the population on another node. Sizes of populations are usually smaller than sizes of populations used in sequential algorithms. Figure 3.8 presents this concept.

There are a number of parameters that affect this approach:

- Migration Gap
Chapter 3. Algorithms used with the Class-Teacher Problem

3.5 Branch & Bound

All discrete optimization problems can be solved with the use of the group of relatively simple algorithm, that is enumerative schemes. In general those algorithms try to check and evaluate a possible solution. Unfortunately, in most cases the number of solution is too big to conduct this kind of computations in a reasonable time. More advanced enumerative schemes provide some kind of optimizations that reduce the amount of solutions that need to be processed. One of such methods is Branch and bound algorithm.

As the very name suggests, the concept of B&B includes:

- Branching, using this scheme, the computer tries to group solutions in sets, in a way that will provide possibility to analyze the whole collection at once

- Bounding, the process of estimation of the lower or upper bound of the objective function value.

Let it be assumed the B&B problem is a minimum optimization one. If a lower bound of a set of solutions is bigger than the weight of the already known solution, the whole collection may be dismissed. Otherwise, the set is split into smaller ones and bounding for them takes place once again. Due to the fact that the lower bound is not be bigger than the already known solution if the set
contains the optimal one, such a set will be split until the set contains only the optimal solution. For the maximum optimization problem the upper bounding is used.

Despite the simplicity of this method, it is not trivial to write an algorithm that uses it. For most of the practical problems both branching and bounding are hard to implement, especially if the quality of this method affects both performance and quality of the algorithm.
Figure 3.2: Local Search Algorithm Block Diagram
Figure 3.3: Simulated Annealing Block Diagram

Figure 3.4: Multiple Independent Runs Concept
Chapter 3. Algorithms used with the Class-Teacher Problem

Figure 3.5: Parallel Moves Concept

Figure 3.6: Genetic Algorithm Concept
Figure 3.7: Genetic Master-Slave Model

Figure 3.8: Genetic Distributed Model Concept
Chapter 4

Developed algorithms

From the set of algorithms described in chapter 3 three approaches were picked: a greedy randomized constructive procedure, a branch & bound and a simulated annealing. The constructive procedure was developed from Santos’s idea[14]. The simulated annealing was developed on the basis of Abramson’s design[1]. As the author of this paper did not manage to find design of B&B algorithm used for any Class Teacher in literature, it was designed and developed from scratch.

Section 4.1 describes the common aspects of algorithms such as the way the solution is encoded or the complexity of common sub-procedures.

The complexity of algorithms are going to be estimated. Formal considerations are going to use following aliases:

- $d$ - the number of days in a week
- $p$ - the number of periods per day
- $t$ - the number of teachers
- $c$ - the number of classes
- $l$ - the number of lecture types
- $n_l$ - the number of lectures to be scheduled

4.1 Common aspects

4.1.1 Instance encoding

The algorithms use a binary file format that stores an instance. The format was designed to allow the algorithms to read data without the overhead of complex processing.

The header contains data describing the size of the instance. That is:

- a number of days in a week,
- a number of periods per days,
Chapter 4. Developed algorithms

• a number of teachers,
• a number of classes,
• a number of lectures

All those numbers are stored in a memory and algorithms may reach their values with the complexity $O(1)$.

With the knowledge of the size of the instance, the algorithm may read the following data:

• the availability of the teachers
• the availability of the classes
• the list of lecture types

The availability is stored as an array of the size $d \cdot p$ written in a day-major order. Each element of such an array is a byte that may get value 0 if it indicates availability or 1 in the opposite case. Because arrays are used, it takes $O(1)$ to read the availability in a particular period.

The list of lectures is the array of 3-tuples. The first element of the tuple stores the id of the teacher, the second stores the id of the class, and the third stores the number of lectures that ought to be conducted. The id of the teacher or class is the number of the array that describes the teacher or class availability. With the knowledge of the number of teachers the algorithm may calculate the offset to the availability table with the complexity $O(1)$.

Weights of periods are stored right after the last lecture type description. The weights are stored in the same manner as the availability, but the item is a double value according to the IEEE 754. This should provide additional flexibility to the way the user wants to provide the weights. Due to the similarity to the availabilities, it is trivial that the weight of particular period may be obtained with the complexity $O(1)$.

While reading the instance to memory the algorithms have only to compute the size of arrays which may be done in $O(1)$. The complexity of copying the availability table is equal $O(dp)$. Because this process must be conducted for each teacher and class, the complexity of this task is $O(dp(t + c))$

4.1.2 Solution encoding

The solution is stored as an array of lists storing assigned lectures. The lecture is defined by pointing to a particular lesson definition. There is one list for each period of time.

The lists are stored in a day major order. This approach allows to reach the container of a particular time slot with constant complexity. As the list used to
implement the solution is taken from the C++ Standard Library, the complexity of basic operations may be taken from the specification[7].

Used operations and its complexities are listed below:

- to add the element at the end of the list - $O(1)$,
- to find the element in a container - $O(n)$,
- to remove the element from a container - $O(n)$, $O(1)$ if the element has been already found,

4.1.3 Solution validation

All the algorithms build solution by assigning the lectures to the time slots one after another. After each step the solution is checked if new lecture did not break it. The process of validation is designed to take advantage of this technique.

The concept of a validator is introduced. It is an object that keeps track of the state of the solution and determines the assignment is allowed before the solution is even changed.

To prevent the close coupling between the Solution and Validator, the communication between the two is reduced to the minimum. As the state of the solution before the Validator is used is important for this process, the Validator must be synchronized with the solution before it can be used.

Check constraints 2.3.1 and 2.3.2, all the already assigned lectures for the period should be checked if they use a class or a teacher associated with the new lecture. This process is going to have the complexity $O(l)$. If any lecture is confirmed to be assigned, the Validator change the availability state of the class and the teacher for affected period of time. Thus, if any lecture is confirmed to be assigned, the validator changes an availability state of the class and the teacher for affected period. This process is executed with the constant complexity. When the Validator is asked whether the new assignment violates the constraint, it simply checks in cache and answers with the complexity $O(1)$.

The checks of constraints 2.3.3 and 2.3.4 are implemented with the use of the previous method. When a solution is synchronized with a validator, initial availability of teachers and classes are read and stored in cache. Thus if a class or a teacher should not be available in a time slot, this fact is already marked in the Validator cache. Despite the advantage of validation of constraint without any overhead, it makes the process of synchronization with the Solution more complex. Because the availabilities must be copied for each teacher and class, and there are $d \cdot p$ availabilities in a week the additional complexity equals $O(dp(t + l))$.

To allow quick checks of constraint 2.3.6 a solution must be built in a specific way. For each class, the lectures on a given day must be assigned to the periods in a non descending order. That is, if a lesson is scheduled for the third period
on Friday, the algorithm can not assign the lesson on the second period without removing following lessons. The algorithms are designed to work with this requirement.

Validator stores information whether any lecture is already assigned in affected day, in every subsequent time slots. If the Validator is asked whether it is allowed to assign a lecture to the marked time slot, it checks whether the previous period of time is used or not. If it is used, the assignment is allowed. Otherwise, it is not. To properly recognize whether the class is unavailable due to the initial unavailability or the assigned lecture, the bit map is used. This approach allows to store the information needed for handling this constraint without an increase in a memory footprint. Due to the caching, the process have constant complexity, but the confirmation of the lecture assignment is complicated because of the need of marking all subsequent time slots. This process has the complexity $O(p)$.

Validator allows to revert lecture assignment. This feature does not affect constraints 2.3.3 and 2.3.4. For constraints 2.3.1 and 2.3.2, handling is straightforward, because neither a class nor teacher can be double booked. Simply clearing the information about the class or teacher unavailability reverts the changes.

Reverting is, according to constraint 2.3.6, more complicated. If the lecture is reverted on the period that has the day usage marked, no further action should be conducted. If the affected period does not have the usage marked, all subsequent periods must have the usage marker cleared. This can be achieved with complexity $O(p)$.

All constraints must be checked each time the Validator allows the assignment. Due to the fact that every constraint may be checked in constant time, the assignment checking may be conducted with the complexity $O(1)$.

Each time the lecture assignment is confirmed, precomputation for each constraint must be executed, hence this process has the complexity $O(p)$.

As the lecture assignments is reverted, the most complex handling is for constraint 2.3.6, hence this process has the complexity $O(p)$.

Process of a synchronization is conducted in two stages. In the first one, the initial availabilities are copied to the cache. In the second one, the lecture assignment is confirmed for every lecture already stored in solution, hence the complexity of synchronization is $O(dp(t + l) + pn)$.

### 4.1.4 Solution evaluation

The process of solution evaluation may work in a similar way to the validation process. The concept of the Evaluator is introduced. It is an object that is able to compute the objective value of the solution in any state (partial or completed).

Before the evaluator can be used, it must be synchronized with the solution that needs to be evaluated. This is achieved by informing the evaluator about every lecture already assigned to the solution.
Each time the Evaluator is informed about a newly assigned lecture, it checks the weight of the affected time slot and adds it to the cached value. Assuming that checking the weight for a period has a constant complexity, this process does not depend on any parameter of the instance, it has complexity $O(1)$.

Each time the Evaluator is informed about the reversion of the lecture assignment, it checks the weight of the affected time slot and deletes it from the cached value. Again, the process has the same complexity as checking the weight of a particular period of time, $O(1)$ in this case.

As the solution objective value is cached inside evaluator, one can learn the actual weight with the constant complexity. Because the process of synchronization needs each lesson to be informed, it has the complexity $O(n_l)$, hence the complexity of getting the objective value of Solution without associated evaluator equals $O(n_l)$.

### 4.2 Constructive procedure

The constructive procedure is meant to be used to construct initial timetables for a more complex algorithms. It is basically a greedy heuristics that assigns lectures in a specific way.

It begins with allocating the lectures to the first periods of each day. When all possible lectures are allocated, it moves to the second time slots. After that, the third time slots are used. This process is executed until all the lectures have been assigned or all the periods analyzed. If all periods have been analyzed and not all lectures are assigned, the method has not been able to construct the initial solution, therefore the result should not be used.

The algorithm assumes that the lectures are going to be assigned in an order from the most to the least urgent lessons. Their urgency may be calculated by counting up, the time slots where each lesson may be conducted, that is the time slots in which both the class and the teacher are available, and divide it by the number of lectures that should be assigned. The smaller the computed value, the more urgent the lesson.

The process of evaluating the lectures needs checking the availability in all the time slots of a given class and teacher. Even with the caching provided by the Validator, it has the complexity $O(dp)$.

Because sorting the lessons by urgency has the complexity $O(l \cdot \log l + l(dp))$, and for each lecture the possibility of an assignment should be checked $O(l)$. Additionally each assigned lesson must be confirmed to the validator $O(lp)$. Thus the complexity of one iteration equals

$$O(l \cdot \log l + l(dp) + l + lp) = O(l(\log l + dp + 1 + p)) = O(l(\log l + dp))$$

There are $d \cdot p$ iterations, therefore the complexity of this method equals:

$$O(dp \cdot l(\log l + dp))$$
Listing 4.1: Constructive procedure

```python
for period in range(periodPerDay):
    for day in Days:
        lectureTypes = sortLecturesByUrgency()
        for lecture in lectureTypes:
            if lecture may be assigned:
                assign lecture
```

4.3 Branch & Bound

4.3.1 Main Concept

The implemented Branch & Bound algorithm uses dynamic programming to construct a solution. The algorithm analyzes each period and tries to find every possible lecture that can be assigned in this particular time slot. A given lecture can be used if it does not make the solution infeasible according to all the constraints except 2.3.5, with the assumption that a number of allocated lectures may not be greater than a number of needed lectures. It is worth noting that it is possible for a time slot to be given no lecture.

With this approach, solutions are branched by the combination of already assigned lectures. Each possibility that is found by the algorithm is bounded and the decision is made whether it is possible that further analysis is promising or not. If the bounding has not dismissed the possibility, the algorithm is re-executed twice, once for the same period and once for the next time slot. As a different sequence of assignments to one time slot does not construct a different solution, the lessons that can be used in the first type of recursion must have been defined after the last picked one, according to the order from instance.

Periods of time are analyzed by the algorithm in a day-major order, to allow the validator to conduct fast checks of constraint 2.3.6. Please see section 4.1.3 for further reference.

The algorithm uses the following escape criteria:

- There are no more periods to be analyzed
- All the needed lectures are allocated

If the algorithm reaches the first escape criterion, the analyzed solution is infeasible according to constraint 2.3.5 and should be dismissed. If the second escape criterion is fulfilled the algorithm has found a solution, which ought to be evaluated, compared to the already known one and remembered in case it is better.
4.3.2 Bounding

As there are two kinds of constraints, bounding is conducted in two phases. In the first phase, the proposed configuration is validated; second one, the lower bound is computed and the decision whether to dismiss the set of solutions or not is made.

The first phase is conducted while searching for possible lessons that can be scheduled. Due to the fact that checking should be done for each lesson provided in an instance, the procedure is performed with the complexity equal $O(l)$.

The second phase is conducted after picking a new lesson. It aims to predict the weight of the worst solution in a branch. Three approaches are introduced:

1. The number of periods left to be analyzed may not be lower than the number of lectures left to be assigned for any class

2. The number of periods left to be analyzed may not be lower than the number of lectures left to assigned for any teacher
3. The weight of a partial solution plus the estimated weight of still unassigned lectures should not be greater than the weight of the already known value.

To properly implement the second phase, the concept of an estimator was introduced. The Estimator is another object that takes the advantage of the dynamic process of constructing a timetable. To be employed, it needs to be synchronized with the solution. The synchronization process assures that the estimator is informed about each already assigned lecture. Additionally, a set of precomputations is conducted to reduce the time needed to bound a branch.

Bounds 1 and 2 are implemented with the complexities $O(c)$ and $O(t)$ respectively. To achieve it, the estimator allocates arrays that store the number of still unallocated lectures. To calculate the initial values to be put to the memory, the estimator must analyze every lesson provided in the instance. For each analyzed lesson, the values in arrays increase according to the number of needed lectures. Assuming that $l >> t$ and $l >> c$, the complexity of this step equals $O(l)$. Each time the estimator is informed about lecture assignment, the values of the affected class and teacher decrease. Once a lecture is reverted, values are increased. When estimator is asked to bound the partial solution, it compares every cached value with the number of unassigned time slots. Despite the fact that the process of checking may be sped up with more complex data structure, values $t$ and $c$ should be relatively small, about 40 for teachers and 15 for classes. Improvements would not cause a significant time reduce. Additionally amounts of time needed to handle lecture assignment and revert would be affected.

Bound 3 makes use of the values calculated by the previous bounds to get the number of lectures that still have to be assigned. It has to be chosen which group to use for the analysis: classes, teachers or the group that provides the best bound. It is worth noting that last probability will require the biggest amount of time, but it will assure the best bound is used.

There is also the possibility to choose a method to obtain the lowest weights from unassigned periods. It is possible to:

- pick the lowest weight and multiple it by the number of unassigned lectures

A very quick heuristic just takes advantage of the fact that if the lowest weight is multiplied by $n$ it will still either smaller than or equals the sum of $n$ lowest weights. Despite it robustness, this method provides poor estimation. As it needs to find the lowest value from the set of all periods, precomputations have the complexity $O(p)$.

- find a sum of $N$ lowest weight for each period at the beginning of the algorithm

This approach finds a sum of the lowest $n$ subsequent periods for each time slot. Precomputations takes less time than the first method but it provides estimations of better quality. As is showed later in this section the precomputations takes $O(p^2 \log p)$. 
• find a sum of N lowest weight for each period for each class and teacher at the beginning of the algorithm

This approach uses the previous method for each class and teacher. It assigns maximum weights for periods of teacher or class unavailability. Despite being most complex, this method provides best estimations from the presented ones. Precomputations takes $O((t + c)p^2 \log p)$.

Listing 4.2 presents the algorithm used to precompute estimations for the minimal weight bounding. As a result table is created to enable the estimator to receive the number of possible lectures for each period. This procedure needs to allocate and initialize a $p \times p$ array, this can be achieved in $O(p^2)$. The algorithm sorts each row of the array. Assuming that used sorting algorithm has complexity equal $O(n \log n)$, it can be conducted in $O(p^2 \log p)$ The last stage contains incremental sums in each row of the array, it is conducted in $O(p^2)$. The overall complexity of this algorithm is $O(p^2 \log p)$

Listing 4.2: Precomputations for the minimum weights estimator

```python
1  estimations[ len(periods)][ len(periods) ] = {inf}
2  for i in range(len(periods)):
3      for j in range(i,len(periods)):
4          estimations[ j ][ i ] = weight(period[ i ])
5  for i in range(len(periods)):
6      sort( estimations[ i ][:])
7  for i in range(len(periods)):
8      for j in range(2,len(periods)):
9          estimations[ i ][ j ] += estimations[ i ][ j-1]
```

4.3.3 Implementation

As this approach assumes a very deep recursion, an alternative to an ordinary call stack has to be implemented. The LIFO queue was used to achieve it. The queue provides four standard methods: pop, top, push, empty. All those methods have the constant complexity. The queue stores structures that contains:

• an information about analyzed period - day and period number,

• an id of last assigned class with the special value 0 if no lecture was yet assigned,

• a flag indicating whether algorithm should continue searching or just revert the lecture assignment.

It is worth noticing that to properly use the LIFO queue, the algorithm adds tasks in reverse order. In the concept a context with assigned lecture was checked
first, now it is added to queue last. A task starting analysis of the next period was executed at the end of iteration, now it is added at the beginning. This change have its origin in the nature of the LIFO queue, tasks that are meant to be executed first, should be queued last.

The iteration aiming to revert the context have to pick task from the queue and revert it in both the evaluator and the validator used during algorithm execution. The last part of flow is the most complex task - $O(p)$, other have a constant complexity.

Iterations that analyzes lecture assignments have to conduct:

- one lecture assignment($O(p)$) - for the validation, $O(1)$ for the evaluation,
- two context estimations($O(c + t)$),
- and two tasks may have been added to the stack ($O(1)$).

The overall complexity of this part of algorithm is $O(c + t + p)$.

### 4.3.4 Distributed Algorithm

Proposed distribution method uses the master-slave model. It splits a search space by creating Tasks that have already preassigned lectures. The information about predefined lectures are coded as a bit string. Each bit stores information whether the lecture should be assigned or not. During the encoding phase, a worker finds the lectures possible to assign in the same way as the branch and bound method is going to do so. The worker iterates through results dismissing lectures that has the bit set to 0. If the bit 1 is found the lecture is assigned and a new search for new possibilities is conducted. After each check, the first bit from the bit string is deleted. When the bit string is empty the decoding procedure is completed. The branch and bound should begin on a period that were used by the decoding procedure lastly.

Let $b$ be the length of bit strings defining tasks. The number of possible tasks can be computed with the following equation:

$$n_t = 2^b$$ (4.1)

Within each task definition the objective value of the best already know solution is provided. After completing the task, the worker sends a computed weight of a result and a solution.

If the worker did not manage to find any possible lessons to assign, and bit string is not empty yet. The worker informs the master about this fact, providing the already analyzed bits. The master will use this information to dismiss infeasible tasks. The bit string will be also checked if it contains only 0. In such case the solution is evaluated and if it met all constraints the node should inform the master about a new solution.
Chapter 4. Developed algorithms

4.4 Simulated annealing

4.4.1 Main concept

To properly use a simulated annealing scheme to solve a particular problem a number of aspects should be defined.

1. the temperature changing criteria

Figure 4.2: Branch & Bound with LIFO Queue

The master generates the tasks in a lexicographical order, sending bit string reversed. This approach allows easier handling of broken tasks. After receiving information about broken tasks, the master compares whether next task have the same prefix. If so, it dismisses it and all subsequent tasks with the broken prefix.
2. the cooling scheme

3. the initial solution generation

4. the neighborhood

The first and the second aspects are rather the parameters of the algorithm. There are not many possibilities that can be picked, and one can alter them to empirically fine tune the algorithm.

Two other aspects defines a logic that the algorithm is using. For the initial solution generation the constructive procedure described in section 4.2 is used. It provides a feasible solutions, with additional care about the quality of it. It is not deterministic method, so two runs on the same instance may provide different results. This feature will be useful in a Multiple Independent Run approach.

The neighbor generation method is the most complex part of this algorithm. Mainly because of constraint 2.3.6. The fact that a result should be still a feasible schedule makes swapping or moving a random lecture not possible or very hard to achieve.

Another approach will be used. A neighbor will be generated by reassigning lectures of picked classes in picked days. As a schedule of whole days will be rearranged, the validator described in section 4.1.3 will be usable. The scheme that are going to be used to pick days and classes will be parametrized by

- the number of days to reassign
- the number of classes to reassign

It is worth noticing that the bigger those values are the broader the neighborhood is, but in cost of more complex computation.

The implementation of a reassign procedure is going to differ in two proposed distributed approach. Please refer to a following subsection for further descriptions.

4.4.2 Multiple independent runs

The constructive procedure is going to be used to construct an initial solution on each node. Because it is not deterministic, it should make the instances starting from a different place of a search space, increasing the probability of finding a global optimum.

In this mode of operation the constructive procedure will be used for the neighbor generation too. The algorithm will behave the same way that is described in section 4.2, but not dropped lectures will be already assigned. It provides a result a much faster than branch & bound method and it takes advantage of the fact that annealing scheme is executed only on one instance. Both Validator and Estimator may be informed about the dropped lectures and reused during
the reassignment. This approach reduces the need to synchronize the solutions with a new instances of those objects.

The proposed method will be able to allocate reassigned lectures in days that was not picked to change. This feature and the non deterministic nature of the constructive procedure assures that there is no possibility that some solutions are impossible to find with the use of this method.

4.4.3 Parallel moves

In this distribution model nodes are meant to cooperate during the neighbor construction. To achieve it the distributed branch & bound will be used. It will reassign picked lectures in an optimal way, enabling the possibility to move the lectures across the days. The reduced number of lectures to allocate in schedule should significantly decrease the amount of time needed for the algorithm to end.

A small alignments are necessary to make the proposed in 4.3 algorithm usable in this situation.

- The master should send the partial solution to nodes
- The master should not stop the working nodes after all tasks are executed.
Chapter 5

Measurements

In this chapter empirical measurements of the proposed algorithms are presented. Before the experiment a set of parameters was picked for each algorithm. The parameters were picked to allow the algorithms to provide possibly best results within a fixed amount of time.

Because real instances provided by an elementary school are used, the decision not to check how the size of instance affects the features of the proposed algorithms has been made. Thus algorithms are executed on the set of those instances, each having:

- the number of classes: 12
- the number of teachers: 28
- the number of lessons to be assigned: 400

Please refer to subsection 5.2 for results of the Distributed Branch&Bound algorithm. In subsection 5.3 results of both MIR and Parallel moves approaches are described. Additionally, in subsection 5.3.3 a comparison of these two models is made. In next subsection used environment is described.

5.1 Environment

By courtesy of an elementary school an infrastructure from a real school computer laboratory was used. Computing cluster contained 8 nodes. The machines were build with the same parts, that is:

- Processor: AMD Phenom(tm) II X4 840
  - Capacity: 3200 MHz
  - Cores: 4
  - L1 cache: 128 KiB
  - L3 cache: 512 KiB

- Motherboard: Gigabyte M68MT-S2P
Chapter 5. Measurements

• RAM Memory: 2 x 2 GiB 1333 MHz

It is worth noticing that those machines use processors with four cores. It provides an opportunity to measure how well the proposed algorithms scale with respect to both the number of nodes and the number of cores used.

The executable files, test data and storage for results were stored on one additional computer that did not had any other role in the cluster. The NFS share on an ext4 partition was used.

The machines were connected directly with the use of a Lantech FE-2400R switch. The standard configuration was used, that is: 100 Mbit/s, full duplex.

The nodes were running xubuntu 12.04 (Linux 3.5.0-23-generic i386) in a live CD mode, the following libraries were used:

• openmpi - 1.5
• boost-mpi - 1.46
• boost-serialization - 1.46
• boost-program_options - 1.46

5.2 Distributed B&B

The distributed B&B ends up with the best solution. Thus the quality of results will not be compared. The important feature in this case is the amount of time that the algorithm needs to be executed. The proposed B&B method have two parameters. The first is a maximum number of lectures that may be preassigned in one task, the second is an estimating method.

The first parameter was fixed and set to 10. That is there will be up to $2^{10}$ different tasks. This number should make the algorithm able to use the full potential of the cluster.

Despite of the used bounding methods, the algorithm needs too much time to be executed on the real instances. The instances were simplified by randomly picking only 2 classes. One hundred instances were constructed using this approach.

Each of the constructed instance was executed with the use of 1, 3, 7, 15 and 31 slave nodes. Because there were 8 nodes in the cluster, in the first three configurations there is only one instance of an algorithm executed per used machine. In two other configurations more than one instance is executed on one node. The configuration with 1 slave node were used as a reference value for other configurations.

Not all instances were executed in with the 31 slave nodes’ configuration. Intense computation on a four cores significantly increased processors temperatures.
To reduce the influence of the temperature, the machines had the time to cool down between two subsequent runs.

Figure 5.1 shows mean speedups of the algorithms. The figure shows that executions on 31 slave nodes needed more time than executions on 16 nodes. During those executions working nodes overheated and throttled down processors to cool down.

Despite the fact that such a situation did not happen on the configurations with a lower number of slaves, it is clear that the proposed algorithm does not scale linearly. Instead the relation between the speedup and the number of nodes is close to being logarithmic.

Figure 5.2 shows a box plot of the execution times for different number of computing nodes.

For some instances the needed time is not decreasing as additional nodes are introduced. But there is a number of instances that scale very well. One may see that the maximum speedups for the configurations 3, 7, 15 is almost equal the number of used nodes.

Medians behave very similarly to the averages presented on figure 5.1. They increase logarithmically with the number of nodes. For the 31 nodes configuration, the medians behave similarly to the averages.

5.3 Simulated annealing

In case of the SA algorithms, results may not be optimal. Thus the problem of comparison of two algorithms is more complex. Some algorithms may be able to finish quickly but the quality of a solution will be worse than results of a long...
Figure 5.2: Execution time of distributed branch and bound for different number of slave nodes

It may be worth to note that both simulated annealing algorithms were designed to search for a timetable with the lowest objective functions. Thus the lower the objective of a found solution the better an algorithm is.

To take measurements of the Multiple Independent Runs and the Parallel Moves approaches the three real instances described in section 2.2 were used. Some alignments of the instances to the problem were necessary:

- when a lesson needed more than one period, such a constraint was dismissed and a number of the lesson’s occurrences was multiplied,
- when class was split to subgroups, only one group was chosen, while the second was dismissed

As no preferences of a time slots were written in any formal way, this part of the instances had to be generated. Three types of instances were generated:
- weights of periods in each day allocated in increasing order
- weights of periods in each day allocated in decreasing order
• weights of periods in each day allocated in randomly

Such a decision has been made to be able to check how different types of instances behaves. The first allocation approach should work very well with the construction algorithm, allowing creating a near optimal solution with it. The second way of allocation should emphasize the greediness of the construction algorithm. The third type of the instances is meant to present how the algorithms behaves for average cases.

5.3.1 Multiple independent runs

The MIR scheme was executed with the use of 2, 4, 8 and 16 cores. An attempt to conduct the algorithm on 32 cores failed, because computations were too intense and the used machines overheated. This shows a flaw in the idea to use the computer laboratory as the computing cluster. The machines’ cooling mechanisms are not ready for a long term intense computation.

It is worth to note that for each configuration one node have a role of a master, what means it have an additional responsibility of gathering results from other working nodes. Due to the fact that only improvements will be reported, any additional overhead should be negligible. For each combination of an instance and a weight set three different runs were conducted to reduce the influence of stochastic nature of a simulated annealing.

As the main idea of the measurements was to find out how the quality of solution behaves when the number of nodes is changed other properties of the algorithm were fixed to:

• number of temperature levels: 20
• number of iteration per temperature level: 1 000 000
• initial temperature: 1 000 000
• number of classes reassigned: 2
• number of days reassigned: 2

A temperature decreased exponentially with a temperature level.

Figure 5.3 shows the relation between result’s quality and the time of the execution for randomly assigned weights. The first minute of algorithm execution provides a lot of improvements. It is hard to tell which configuration tends to be better in that period. Later the improvements are less intense.

The most efficient was the 8 nodes configuration, it managed to provide better results. Unfortunately the figure shows that the algorithm does not scale with the number of nodes. The configuration with only one slave node outruns the 4 nodes configuration. Probably the simplicity of synchronization of the one slave node configuration makes it so efficient.
Figure 5.3: The relations between the quality of results and execution time for instances with randomly assigned weights

The 16 nodes configuration is slower than the 8 nodes one. It may be explained by the configuration of the nodes. For the 8 nodes, each instance of the algorithm have its own physical machine, for any bigger number of the nodes the machines has to be shared. Additionally it may be caused by a temperature of the processors, which made the 32-node configuration impossible to execute.

The instances with the periods allocated in the increasing order were meant to be easiest for the initial solution construction. The smallest improvement on the plot in figure 5.4 shows it.

The quality of results have increased significantly with the number of used nodes. In this case the problem of shared machines did not affect the results and configuration with 16 nodes have the best performance.

It is worth notice that two configurations seems to stuck in a local optimum, because for a long period no improvement was shown. The increased number of nodes may be a way to reduce such a risk.

Figure 5.5 shows the relation between the quality of found solutions and the time of execution for the instances that are supposed to be hardest for the used initial heuristic. This configuration was the only one for which the set of runs succeeded for the 32-nodes configuration.

For this type of instances similarly to the average cases the algorithm has not scaled with the number of nodes. The configuration with 2 nodes was faster than both the 4-nodes and 16-nodes configurations.
5.3.2 Parallel moves

The Parallel Moves approach uses the distributed B&B algorithm for a neighbor generation. It consumes much more time than the greedy algorithm used in the MIR, even executed on a distributed system. Due to this fact and the limited amount of the time that the cluster might be used for measurement the properties of Simulated Annealing were set to reduce the execution time of the algorithm:

- number of temperature levels: 5
- number of iteration per temperature level: 10
- initial temperature: 1 000 000
- number of classes reassigned: 1
- number of days reassigned: 1

The instances with randomly assigned weights were used as the input data set.

Due to the limited time, even with the reduced number of iteration only configuration with 8 and 16 nodes were executed. Figure 5.6 shows the results of the experiment. The parallel moves approach provides big improvement in the first few second of the algorithm execution. Later the improvements are smaller and they are rare.

*Figure 5.4:* The relations between the quality of results and execution time for instances with increasing weights
Chapter 5. Measurements

5.3.3 Approaches comparison

Having the measurements it may be interesting to compare the two proposed approaches. To do so, the configurations with the equal number of nodes are compared on figure 5.7.

The parallel moves approach provides better solutions in the first stage of the algorithms’ execution. After the first 10 seconds, the multiple independent runs behaves more efficiently and significantly outruns the competitor.

The plot shows that the Branch & Bound shows tendency to get stalled, this happens when used bounding methods does not reduce the number of possible solutions good enough.
Figure 5.6: The relations between the quality of results and the execution time for the parallel moves approach

Figure 5.7: Comparison of MIR and Parallel Moves
Chapter 6

Conclusions

The problem of a school timetable generation is still hard to define unambiguously. The main reason are differences of educational systems of different countries. Even if the problem is analyzed with the requirements of only one country, it is still too vague to define it formally. Law regulations may provide the possibility to relax some requirements in case they can not be met by a school. It is necessary, because even simplified problems show the tendency to be unsolvable in a polynomial time.

After the analysis of the current Polish law regulations and the real instance from the average elementary school an optimization problem was defined. The problem has been proven to be NP-complete, to explain why no effort to find an optimal polynomial algorithm have been made.

The branch and bound algorithm is an example of an approach to get the best possible solution. The author of this paper did not manage to found any information in literature about the application for a B&B in a Class Teacher Timetabling Problems set. Three different types of bounding method were presented.

The B&B scales logarithmically with the number of nodes in a cluster. The algorithm shows a tendency not to scale well on multiprocessors. The algorithm in such a configuration may have a smaller performance than one working on only one node.

The B&B algorithm needs an unfeasible amount of time to execute for a practical instance, thus the distributed meta heuristics were proposed. The first (MIR) is a very simple modification of a simulated annealing algorithm, which creates independent instances of a simulated annealing on each node in a cluster and uses results found by all of them. The second (PM) uses the proposed distributed B&B method to generate a neighborhood. The B&B is used only to reassign part of the whole schedule.

The Parallel Moves seems to scale with the number of nodes in a cluster, but due to the time consuming method of the neighbor generation it still is significantly slower than the Multiple Independent Runs. Unfortunately the empirical measurements showed that the Multiple Independent Runs does not scale with the number of nodes.

Infrastructures in the schools may be used as computational clusters. But a person that tries to use them should be aware that it may be impossible to use
a full computational power of such configurations. An infrastructure may not be ready for very intense long running tasks.

All those information suggests that the answer for the research question is negative. The algorithm that does not scale with the number of nodes are much more efficient than the algorithm that scales. B&B scales logarithmically so the number of nodes needed to outrun simple heuristic algorithm is bigger than an average school can afford.

For future work, it may be interesting to try to change the approach of splitting the search space across nodes in the distributed branch & bound. The method used in this paper spent a lot of time at the end of the neighbor generation, while not providing significant improvements. Both a different split method and some heuristic to estimate whether it is worth to spend more time trying to get a better solution may have a positive impact on the algorithm performance.
List of Figures

3.1 Storage of the solution concept . . . . . . . . . . . . . . . . . . . 16
3.2 Local Search Algorithm Block Diagram . . . . . . . . . . . . . . 25
3.3 Simulated Annealing Block Diagram . . . . . . . . . . . . . . . . 26
3.4 Multiple Independent Runs Concept . . . . . . . . . . . . . . . . 26
3.5 Parallel Moves Concept . . . . . . . . . . . . . . . . . . . . . . . 27
3.6 Genetic Algorithm Concept . . . . . . . . . . . . . . . . . . . . . 27
3.7 Genetic Master-Slave Model . . . . . . . . . . . . . . . . . . . . . 28
3.8 Genetic Distributed Model Concept . . . . . . . . . . . . . . . . 28
4.1 Branch & Bound Concept . . . . . . . . . . . . . . . . . . . . . . 35
4.2 Branch & Bound with LIFO Queue . . . . . . . . . . . . . . . . . 39
5.1 Means speedups . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
5.2 Execution time of distributed branch and bound for different num-
ber of slave nodes . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
5.3 The relations between the quality of results and execution time for
instances with randomly assigned weights . . . . . . . . . . . . . . . 47
5.4 The relations between the quality of results and execution time for
instances with increasing weights . . . . . . . . . . . . . . . . . . . . 48
5.5 The relations between the quality of results and execution time for
instances with decreasing weights . . . . . . . . . . . . . . . . . . . . 49
5.6 The relations between the quality of results and the execution time
for the parallel moves approach . . . . . . . . . . . . . . . . . . . . . . 50
5.7 Comparison of MIR and Parallel Moves . . . . . . . . . . . . . . . 50


