bachelor's thesis

A hybrid approach to the general high school timetabling problem

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Algorithmen und Komplexität

by:	**Valentin Dreismann**
supervising professor:	Prof. Dr. Klaus Jansen
supervisor:	Malin Rau

Kiel, 28th of August 2015

Details

full/first name:	Dreismann, Valentin
enroll. No:	1011659
major:	Informatik B.sc.
supervising professor:	Prof. Dr. Klaus Jansen
supervisor:	Malin Rau
institute:	computer science
working group:	Algorithms and Complexity

This thesis is also available electronically on the enclosed CD. Furthermore, the CD also includes the relevant source codes and the data for the generated statistics. The Roepstorff model was compiled with Qt 5.2.1 64bit, the XHSTT source code was compiled with the GCC toolchain in version 4.8.4. Additional libraries such as Boost and Expat are the packages currently shipped with Ubuntu 14.04. The KHE library is the current (19 May 2015) one with an additional bugfix. The IP solver in use is the Gurobi Optimizer 6.0 64bit by Gurobi Optimization Inc. with a free academic license.

I would like to thank my supervisor Malin Rau for her support.

Statement of Authorship

I hereby certify that this bachelor's thesis has been composed by myself, and describes my own work, unless otherwise acknowledged in the text. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged.

....
Valentin Dreismann

Contents

List of Figures

Abbreviations

TT	timetabling
poi	point of interest
poa	point of application
LP	linear program
ILP	integer linear program
IP	integer program; same as ILP
MIP	mixed integer program
ITC	International Timetabling Competition

1 Introduction

1.1 The Timetabling Problem

Timetabling (abbreviated: *TT*) is the allocation of resources to certain time slots within the timetable while abiding by certain constraints. Students, teachers and rooms are common examples for resources. There are many different variants of timetabling problems, tailored to the specific institutions' or companies' needs in question; [BC98] proposes a classification into teacher assignment, classteacher timetabling, course scheduling, student scheduling and classroom assignment. Teacher assignment only assigns teachers to courses, while class-teacher assignment only assigns teachers to classes, disregarding rooms and courses. The other subproblems are similarly defined; they only cope with a subproblem of the general case.

Typically, two distinct types of constraints are considered: *soft* and *hard* constraints. Soft constraints are often preferences of teachers and students, such as workdays ending at 04:00 PM and a minimized count of free hours between courses. They may be violated without making the timetable infeasible; as their violation is undesired, it is usually penalized with a cost function. Hard constraints make a solution infeasible if violated. For instance a teacher cannot teach two courses at the same time. However, hard constraints are often modeled as a (steep) cost instead of directly rendering the solution infeasible. This is also used as *Lagrangian relaxation* to generate initial solutions for genetic algorithms or local search algorithms.

Many of the standard timetabling models as well as their subproblems have been shown to be NP-complete [CK95]; however, NP-completeness cannot readily be claimed for every model. April Lin Lovelace elaborated extensively on the changes in hardness while successively extending her model [Lov10]. In particular, she found constraints which indeed change the complexity of an NP-complete timetabling problem such that it is in fact in P.
Decomposing the general case to its subproblems does allow for different algorithms to tackle different parts, often resulting in a multi stage solution (e.g. [AG08]). There are many different approaches to solve TT problems, such as constraint logic programming ([GJBP95]), genetic algorithms ([IDw13], [AG08], [Wan02], [BMKL08], [QA13]), local search heuristics ([CDB09], [HS07], [GNP12] and [LT15]), integer linear programs ([THH97], [Akk72] and [Sk14]) as well as swarm optimization ([TB12a] and [TB12b]). Many approaches use different algorithms on different stages to harness advantages specific to those algorithms. A more comprehensive presentation can be found in [SK13] and [dBBB$^+$13].

The diverse requirements for timetabling led to a multitude of mostly incompatible models. Therefore researchers were unable to compare the performance of their solutions with competing cutting-edge approaches, making a fair assessment of the algorithms' actual performance difficult or even impossible. Furthermore, the instances those algorithms ran on were naturally limited. All of the above-mentioned shortcomings led to the development of a common model (and XML file format) for

TT problems: *XHSTT* ([PAD$^+$12]). This format does not only provide a uniform modelling, but also allows to embed solutions in the instance files themselves. The International Timetabling Competition (ITC) in 2011 did rely exclusively on XH-STT. Up to now, there have been three competitions, the latest took place in 2011 [ITC]. It does not only feature a common file format, but also provides a host of real world instances with the best yet found solutions for each. The instances are constantly being updated; the last available archive is ITC-2014.
As there are instances in ITC-2011 for which none of the finalists could provide a feasible solution, hard constraints generally yield a higher cost, but do not necessarily disqualify solutions in the search space from further investigation.

1.2 Aim of this thesis

This thesis explores the capabilities of a two stage algorithm; the first stage assigns parts of the resources (e.g. teacher-course scheduling) while the remainder is solved using an ILP (integer linear programming) approach.
Using an ILP solver at the second stage allows us to retain provable optimality at least for this part. While it may not be practical to calculate the optimal solution at the second stage every time, we do have a means to base assessments of genetic operators on sound arguments, especially the *optimal* solution quality given the constraints from the first stage.
We draw on the work of [Sk14] for the integer model, which presents the first LP model for the general XHSTT format altogether. The approach, measured by the rule framework of ITC2011 ([MSP$^+$10]), performs better than the worst competitors, but is not quite as competitive as the top algorithms (see T.2 [Sk14]). As this LP approach uses commercial software (Gurobi), it is barred from taking part in the ITC competition. A genetic algorithm ([JD12], HFT) has been one of the finalists in ITC2011. While it is not competitive with the other finalists (all of the top 3 used local search heuristics), cost incurred by violations of the hard constraints are generally lower than those of the LP approach, given that neither found a feasible solution. For some instances, we do know about feasible solutions, even though the LP solver could not find them within the time limit.
The point of interest is whether the genetic algorithm can guide the LP solver towards promising regions. It is known that increasing the number of decision variables for the LP model greatly decreases the solver's ability to search the solution space thoroughly. For some ITC instances, the variable count will go up to $10^5 - 10^7$([Sk14]) and indeed all of the not feasibly solved instances do have a variable count of that magnitude.
Originally, the aim of this thesis was to *develop* such a genetic algorithm. However, our planning efforts did not pan out as expected. In particular we hit several unforeseen obstacles on the IP part of the problem. Therefore, instead of *developing* such an algorithm, our focus shifted towards the *feasibility* of such an algorithm. In particular: *Can* an effective algorithm of that kind be conceived? What are the

principal limitations such an algorithm is subject to?

1.3 Related work

Multi stage approaches typically start by constructing feasible or nearly feasible solutions in the first stage, while focusing on the optimization of soft constraint violations in the second stage (as in [Suy10]). Indeed, this is also common practice for IP formulations such as in [Sk14], where *Langrangian relaxation* is a natural way of guiding the solver during the first stage. *Local search heuristics* are algorithms that explore the solution space by moving step by step from initial solutions; typically they rely heavily on a-priori knowledge. The second stage typically features a different algorithm than the first stage, such as a local search heuristic ([MSK12]). While there is an abundance of research on combinations of genetic algorithms and local search, fewer researchers focus on special fields such as swarm optimization and constraint logic programming or even neural networks.
Probably due to the fact that computing power did only dramatically evolve to broad practical applicability in the last two decades, the research on IP formulations with respect to the TT problem is still quite limited. However, special techniques such as *Column Generation* allow for the reduction of complexity of (I)LP problems. This incurs the loss of guaranteed optimality. Such an approach has been applied successfully in [vdBH12]; in this case, it is combined with a heuristic approach. Another trial of simulated annealing in conjunction with a very large neighborhood search (abbrev. VLNS) performed by solving an IP problem can be found in [ADSV07]. We do not know of any prior attempt at exploring genetic algorithms in conjunction with an IP problem formulation as the second stage. In particular, we can safely assume that this is the first attempt to solve the general XHSTT problem definition with a combination of an IP formulation and a genetic algorithm (or even a heuristic).

2 Approach

2.1 Assessment

The aim of this thesis is to improve the performance of the pure IP (integer programming) model. Quality does have many aspects; the most important ones may be the violations in hard and soft constraints for specific instances. Another crucial factor is the runtime and memory consumption needed to achieve a certain quality. In the end, we also prefer to be certain that the algorithm will eventually find any feasible solutions at all.
We seek a direct comparison to the underlying IP model, to analyze if the IP model is better off without this additional evolutionary layer.
We use the *Gurobi* solver, version 6.0 with a free academic license. At this point I would like to thank the authors for bestowing their state-of-the-art solver for research purposes.
Some researchers did choose not to implement the recombination operator [BMKL08] after assessing its performance; others did not assess its performance individually. We set out to explore the capabilities and limitations on a more theoretical basis. Currently it remains an open question if the complexion of the timetabling solution space actually does allow for such operators to perform better.
In this thesis, we will try to overcome this by analyzing the behavior of our operators using an approach as suggested by [SL12] and originally introduced by Weinberger ([ED94]). This approach analyzes the movements of operators on *fitness landscapes* in order to quantify the performance of operators.
In particular let Ω be a set of solutions, let $\Phi : \Omega \rightarrow \mathbb{R}$ be a fitness function that assigns a *fitness value* to each solution and let $N_k : \Omega \rightarrow \mathcal{P}(\Omega)$ be a neighbor function for $k \in \mathbb{N}$ defined by

$$N_k(u) = v \in \Omega : d(u, v) \leq k \quad \forall u \in \Omega$$

The metric $d(u, v)$ is given by the minimum number of operator applications to reach solution $v \in \Omega$ from $u \in \Omega$. Then, the *fitness landscape* L is defined as $L = (\Omega, \Phi, N_k)$. One of the proposed measures is the *ruggedness*: "the autocorrelation of the time series recording the solution quality generated by a random walk process". I.e. (σ^2_Φ denotes the variance):

$$p(s) \sim \frac{1}{\sigma^2_\Phi (m-s)} \sum_{t=1}^{m-s} (\Phi(u_t) - \overline{\Phi})(\Phi(u_{t+s}) - \overline{\Phi})$$

For a time series $\Phi(u_t)$, the above measure defines the correlation of two points' fitness whose distance is s along a random walk of length m through Ω. This measure is particularly interesting for the analysis of the recombination operator, as it effectively measures if the recombination does indeed yield better quality.
Eric Angel and Vassilis Zissimopoulos have shown that for several NP-hard problems, a classification by their hardness according to the above-mentioned measure is backed by research by the scientific community ([AZ00]). It should be noted that this research does classify the hardness for *local search algorithms.* However, genetic algorithms work quite similar to local search heuristics; the selection, recombination and mutation scheme does in effect resemble the neighbor functions of local search heuristics.
The second measure correlates the fitness value and the minimum distance $d(u), u \in \Omega$ to the nearest optimum in the search space:

$$p(\Phi, d) \sim \frac{1}{|\Omega|\sigma_\Phi\sigma_d} \sum_{u\in\Omega} (\Phi(u) - \overline{\Phi})(d(u) - \overline{d}) \tag{2.1}$$

Marco Chiarandini and Thomas Stützle did an interesting and extensive landscape analysis ([CS03]) on a simpler TT model. For this purpose they explored a common representation which places the events in a matrix designated by the cartesian product of available times and rooms. The local search operators then operate by moving events within the matrix. They concluded that the respective quality of local optima does indeed influence the final solution quality, which is hardly surprising. However, they were able to reject the conjecture that the local optima are clustered in one region or that local optima "close" to the global optima tend to be superior to those farther away. Furthermore, they could not support the notion of large plateaus generally being present on the landscape; even when they did exist, the influence on the solution quality was inconclusive. The genetic approach presented in this thesis does operate on another landscape. Therefore, the aforementioned results do not necessarily apply to our approach.
Furthermore, the abstract XHSTT format allows to prepend our algorithm to any available implementation by means of the *preassigned* feature. This enables us to test the algorithm's impact on several different sets of algorithms.
We acknowledge that this approach might cut both ways, while it may improve certain aspects, it will perform worse in others.

2.2 Genetic Algorithms

Genetic Algorithms imitate the biological principle of evolution, as suggested by Charles Darwin. The fittest individuals tend to survive by virtue of a natural selection process. Subsequently more features have been added: in particular *mutation* as well as *recombination. Mutation* is a random and undirected change in the encoded information. Superior individuals pairing off to combine their genes is called *recombination.* In the realm of computer science individuals are also called solutions.

Furthermore we need the notion of *feasible* and *(sub)optimal* solutions. *Feasible* solutions are those which can be implemented directly; all prescribed assignments can be followed. *Infeasible* solutions on the other hand may yield to teachers having to teach two classes simultaneously, two classes having to attend to lectures simultaneously etc. Those violate logical or intuitive requirements to a solution. Why we consider infeasible solutions in the first place will be explained at a later point. We will also introduce methods to measure up solutions. Thus, the definition of *optimal* and *suboptimal* solutions naturally arises. Please note that while our solution space is vast but finite, an optimal solution does exist without invocation of Zorn's Lemma.

Mathematically we can model this as follows:

Let P be the population of individuals. The recombination selection function $selRecomb : \mathcal{P}(P) \to \mathcal{P}(P)$ selects a subset of P according to the fitness as indicated by the fitness function $fitness : P \to \mathbb{R}_+$. Promising regions in the notional *fitness landscape* may be engulfed by valleys. A naïve selection according to the fitness function will prevent the algorithm to reach such a region. Therefore, we will choose a probability distribution weighted by the fitness as assessed by $fitness$ for use in $selRecomb$.

The fitness function assesses the quality of the solution with a predefined measure; in the context of timetabling, this will be cost incurred by constraint violation. The solutions selected by $selRec$ are paired off by invoking the recombination operator $recomb : P \times P \to P$. One may introduce another function $findPairs : \mathcal{P}(P) \to (P \times P)$ to determine how to find pairs in the subset selected by $selRecomb$. The results of applying the recombination on the selected subset are subsequently reinserted into P. Then solutions of P are randomly selected, mutated using the mutation operator $mutate : P \to P$ and added back to P again.

In the end, the selection function $select : \mathcal{P}(P) \to \mathcal{P}(P)$ removes solutions of poor quality as assessed by $fitness$ from P.

To apply genetic algorithms, we need an initial population for P in the beginning. It is typically generated by some other means; there are some trivial solutions for other problems such job scheduling; The most trivial schedule might be scheduling all jobs on a random machine; for TSP (TravellingSalesMan) problems an initial solution might be a random walk without repetition over all cities. For timetabling, there is most likely no trivial initial solution [1]; we will therefore resort to a simple constraint based heuristic. We do not expect to generate feasible solutions for the initial population, as we do not assert to find feasible solutions for all solutions in the first place. A sparse initial population covering the majority of the search space is of great importance; while mutation operators *should* be able to reach the entire search space in theory, in reality local basins might prevent algorithms from finding globally optimal solutions.

[1] While we *could* randomly schedule classes to teachers, we were very likely to violate any number of hard constraints. While those violations do not make our initial solutions useless, we will try to keep them as rare as possible.

2.3 Integer Linear Programming

Linear programming (LP) is a special, constrained mathematical optimization problem. We are in search of a tuple of continuous variables $(x_1, \ldots, x_n)$ which maximize a linear function z, also called *objective function*, obeying a set of linear constraints. Formally, this can be defined as follows:

$$\begin{array}{ll} \text{Maximize} & z = \sum_j c_j x_j \\ \text{subject to} & \sum_j a_{ij} x_j \leq b_j \quad i \in \{1, \ldots m\} \\ & x_j \geq 0, \quad j \in \{1, \ldots n\} \end{array}$$

An *integer linear program* (ILP) is a linear program such that $x_j \in \mathbb{N} \quad \forall j$.
While the roots of linear programming date back as far as 1827 (Fourier), it was until World War II that increased attention towards LP problems led to tangible progress ([Sie02]). Numerous dynamic transportation problems and cost-/revenue calculations at those times are, besides pure economical problems, also the source for its name: “programming” does indeed refer to “planning” resources.
(I)LP formulations do have convenient properties, which make them suitable for many problems:

1. they can be solved to optimality, though time consumption may be unappealing
2. many solvers follow an iterative approach
   i.e. after the first solution is found, its quality is subsequently improved
3. they do not require any programming skills
   (indeed there exist highlevel modeling languages such as AMPL, LINGO and MPL)
4. they provide a problem independent approach to solve for many different and unrelated problems

The canonical way to introduce solution methods for LP problems nowadays is the *Simplex Method* published by George B. Dantzig. However, modern solvers do also rely on interior-point methods, Branch-and-Bound, Branch-and-Cut, Branch-and-Price and several heuristics ([Che10], see dedicated chapters). As we will see later on, the distinction between these methods is important; while interior point methods optimize *LP* problems, the branch algorithms reverse the relaxation and try to find an *ILP* solution from a given *LP* solution. There exist weak polynomial algorithms for solving LP problems. *Weak* refers to the fact that either space or runtime complexity are polynomially bounded, but not necessarily both. Unfortunately this is not the case for the second phase, i.e. solving for the *ILP*. Surprisingly many problems can be formulated as (I)LP problems; for many NP-problems there are even several formulations as (I)LP problems available.

(I)LP formulations of the TT problem did already come up in the 60s: Norman L. Lawrie described a school TT problem in [Sie02]. However, at the times the computing power did not suffice to solve large (I)LP instances, therefore his analysis was merely confined to theoretical considerations and small test samples. Nonetheless, especially with the advent of new solution techniques such as interior-point methods and the continuous progress in computer power, it is possible to tackle considerably larger (I)LP instances even on consumer market hardware, as done in [Sk14].
Linear Programs (LP) mark the general case while Integer Linear Programs (ILP) are special cases as noted above. Mixed Linear Programs (abbrev. MIP) are LPs with both continuous and integer constrainted variables.

3 Model

In the following chapter we will give an overview of the definitions in the XHSTT format. Two subsections are dedicated to the modelling of those definitions in the LP formulation as well as in the genetic algorithm. We will not try to cover the XHSTT format exhaustively, as this would take up too much space in this thesis. Instead, the basic ingredients will be covered, for an in-depth discussion of the LP part please refer to [Sk14]. The following chapter closely adheres to the above-referenced description.

3.1 XHSTT Model

Each instance of XHSTT mainly consists of the following objects: available *times* (or: *time slots*) given in the set T, *time groups* given in TG, *resources* in R, *events* in E, *event groups* in EG and *constraints* in C.
Time slots are a non-overlapping partition of the available time. Resources may be rooms, teachers or students/classes. Resources are characterized by the fact that they do not allow superposition; i.e. a teacher must not be assigned two classes for the same timeslot. Similar constraints apply for classes and rooms.
Events are objects which can require certain subsets of resources and times to be assigned. Each event $e \in E$ has a predefined duration $D_e \in \mathbb{N}$. Examples are courses to be scheduled in a week; i.e. the course "math6a" with a duration of 4 hours could require the resource "6a" of type "class" (preassigned), a resource of type "teacher" of a certain subgroup such as "math teachers" as well as a number of times whose sum is 4.[1]
Groups are introduced for ease of use; they pass down all their properties to their children. Constraints represent penalizing hard and soft constraints. Their definition is more involved and we will cover them in-depth later on.
Events are the core objects. Events may be split into sub events subject to special constraints. We denote the set of sub events as SE, furthermore we write $se \in e$ for sub events se of e. Sub events inherit all properties from their parent, in particular all resource assignments. Sub events do have a duration $D_{se} \leq D_e$ and satisfy $\sum_{se \in e} D_{se} = D_e$. The sub events allow us to split an event into arbitrary long pieces. For instance, the event "math6a" with a duration of 4 hours altogether may be taught at 4 different times, each one hour; alternatively, you could teach two times, but each time two hours. A decomposition of 5 into sub events is

$$1 \times 5, \quad 1 \times 4, \quad 1 \times 3, \quad 2 \times 2, \quad 5 \times 1$$

and valid splits are:

$$1 \times 5, \quad 1 \times 4 + 1 \times 1, \quad 1 \times 3 + 1 \times 2, \quad 2 \times 2 + 1 \times 3, \quad \cdots$$

Each event has assigned event resources denoted as $er \in e$; each resource requirement must be satisfied by assigning an eligible resource of the resource type in question.

[1] Please note that Events do not enforce assignments themselves. They merely offer "slots" whose assignment is then enforced by specialized constraints.

Resources may be preassigned. When resources are assigned to an event, they may assume specific roles $role_{er}$. Depending on the role undertaken, different sets of constraints may apply.
Times T are chronologically ordered and indexed by $p : T \to \mathbb{N}, t \to p(t)$. The dummy time (and resource) is specific to the ILP implementation and not defined in XHSTT. It is treated separately and doesn't have a specific index. Similar to events, we write $t \in tg(tg \in TG)$ if time t is member of time group tg.
Constraints have *points of interest* (abbrev. poi), i.e. events, event groups, times, time groups or resources. Points of interest represent the larger buildings blocks constraints refer to. *Points of application* (abbrev. poa) on the other hand are components of the poi. They are the part relevant to calculating the induced cost. In particular, each poa induces an associated *deviation*. Those *deviations* then, by virtue of a *cost function*, contribute to the total cost of the constraint at hand.[2] When you define the model you will typically think in terms of *poi* while the model will calculate the cost in terms of *poa*.
Consider event $e \in E$ in an *AssignResourceConstraint*. Then e itself may be a point of application while each "slot" (in XHSTT parlance: *event resource*) is a poa and creates its own deviation; if an event resource is not properly assigned, the deviation of that event resource will assume a certain value. The cost of the *AssignResourceConstraint* is then the value of the *Cost Function* of its deviations. Accordingly we write for Event $e \in E$, $r \in R$, $eg \in EG$ and $c \in C$: $e \in c$, $r \in c$ and $eg \in c$ if those are poi of c.

3.1.1 Objective Function

Both hard and soft constraints contribute to the objective function. In XHSTT, cost functions are generally non-negative, therefore a cost of zero trivially indicates an optimal solution (which indeed may be the only way to prove optimality apart from the IP formulation). The poi p of a constraint c are denoted by $p \in c$. The deviation of each assigned poa is denoted by $d \in p$. The cost incurred by a constraint depends on whether it is a hard or soft constraint, the individual weight $w_c \in \mathbb{N}$ and the cost function CF. Let $c \in C$. Then $s_{c,p,d} \in \mathbb{N}$ denotes the value of the deviation $d \in p$ for poa $p \in c$. The cost of constraint $c \in C$ is denoted by $f(s_{c,p,d})$ and defined as follows:

$$f(s_{c,p,d}) = w_c \cdot CF(s_{c,p,d})$$

For the considered XHSTT model (which is already deprecated[3]) there are five types of *CostFunction* altogether. Almost all instances use only the most basic type, the linear cost function:

$$\mathrm{CF}^{\mathrm{Sum}} = \sum_{p \in c, d \in p} s_{c,p,d}$$

[2]Thus XHSTT distinguishes between the *measurement* of deviations and their *penalizement*
[3]Note that in the most recent specification of XHSTT, there are only 3 cost functions. All of those are part of the old specification; therefore, costs for the new model equal costs for the old models if none of the remaining 2 cost functions are in use.

Cost is measured as cost incurred by violations of hard constraints and cost due to soft constraints. A cost can therefore be denoted as a pair (hard cost, soft cost). This concept proves useful to implement a two stage approach, thus dramatically reducing the dimension in the first stage.

3.1.2 Basic variables

Throughout the MIP formulation we will make frequent use of the few basic variables defined in the following:
$x_{se,t,er,r} \in \{0,1\} = 1$ iff sub-event $se \in SE$ starts at $t \in T \cup \{t_D\}$ with resource $r \in er$ being assigned to event resource $er \in se$. t_D is the dummy time, r_D the dummy resource. Both of those are not defined by XHSTT but used as placeholders in the ILP model.
$y_{se,t} \in \{0,1\} = 1$ iff sub-event $se \in SE$ starts at time $t \in T$.
$v_{t,r} \in \mathbb{N}_0$ represents the number of times the resource r is being used at time t. Note that this is bounded to 1 for feasible solutions; however hard constraint violations are only recounted here, it is until the *AvoidClashesConstraint* that this violation is penalized.
$w_{se,er,r} \in \{0,1\} = 1$ iff sub-event $se \in SE$ is assigned resource $r \in R$ for event resource $er \in se$.

3.1.3 Basic constraints

In this section we are going to introduce a few basic hard constraints. If violated, the resulting schedule will be infeasible. Furthermore, many presented constraints will not only enforce feasibility but reasonable domains for the constrained variables in the first place. The following paragraph stronly adheres to the structure as given in [Sk14].
The first constraint ensures that each event resource of event e is assigned exactly one resource:

$$\sum_{t \in T, r \in er} x_{se,t,er,r} = 1 \quad \forall se \in SE, er \in se \tag{13}$$

Let $|er_{se}|$ be the cardinality of the set er_{se} (i.e. the set of *event resources*) for any sub-event se. Let

$$\sum_{er \in se, r \in er} x_{se,t,er,r} = |er_{se}| \cdot y_{se,t} \quad \forall se \in SE, t \in T \tag{14}$$

This constraint ensures that

1. $y_{se,t}$ is true iff se starts at t
2. a sub-event is assigned exactly one starting time

Note that unused sub-events can be assigned the dummy time t_D.
Let t_1, t_2 be starting times for $er \in se$. Then the right hand side will equal $|er|_{se}$ for both y_{se,t_1} as well as y_{se,t_2}. As $x_{se,t,er,r}$ is binary, for both t_1, t_2 we need $x_{se,t,er,r}$ be true for $|er|_{se}$ combinations of er and r. Constraint (13) ensures that each event resource is assigned only one resource. Therefore, er must be distinct for t_1, t_2. This is a contradiction because (14) is only allows $er \in se$ as event resources, we cannot have to distinct subsets of cardinality $|er|_{se}$.

In the following we will introduce some more basic variables. Those are frequently built upon when formulating various constraints. Let $T^{start}_{se,t} \subset T$ be the set of possible starting times for which sub-event $se \in SE$ spans over time $t \in T$. The following definition is straightforward:

$$T^{start}_{se,t} = \{t' \in T \setminus \{t_D\} | p(t) - D_{se} + 1 \leq p(t') \leq p(t)\}$$

($p(t) \in \mathbb{N}$ is the index of $t \in T$; recall that the times are ordered consecutively with ascending index)
The following constraint derives the value of $v_{t,r}$ from $x_{se,t,er,r}$; for $t \in T, r \in R$ it counts the concurrent usages of r during t:

$$\sum_{se \in SE, er \in se, t' \in T^{start}_{se,t}} x_{se,t',er,r} = v_{t,r} \quad \forall t \in T - \{t_D\}, r \in R$$

Furthermore we initialize the indicator $w_{se,er,r}$ as follows:

$$\sum_{t \in T} x_{se,t,er,r} = w_{se,er,r} \quad \forall se \in SE, er \in se, r \in er$$

This variable indicates whether the sub event se is assigned resource r for the event resource er.
Sub-events cannot be assigned start times that will cause them to exceed the space of continuous times:

$$y_{se,t} = 0 \quad \forall se \in SE, t \in T \setminus \{t_D\}, p(t) + D_{se} - 1 > |T|$$

For the formulation, we create the set of all possible sub-events for each event $e \in E$. However, only the sub-events being used are *active sub-events*, meaning that all others are to be discarded. A sub-event becomes active if it is assigned a starting time or at least one non-preassigned resource. Let $u_{se} \in \{0,1\} = 1$ iff $se \in SE$ is active. Furthermore let $PA_{er} \in \{0,1\}$ iff $er \in se$ has a preassigned resource. The following constraint ensures that se is indeed considered active when assigned any non-preassigned resource:

$$\sum_{r \in er - \{r_D\}} w_{se,er,r} \leq u_{se} \quad \forall se \in SE, er \in se, PA_{er} = 0$$

Any assigned starting time will lead to the activation of se:

$$\sum_{t \in Tt_D} y_{se,t} \leq u_{se} \quad \forall se \in SE$$

If neither a time nor a not-preassigned resource is assigned, $se \in SE$ must be inactive:

$$\sum_{t \in T \setminus \{t_D\}} y_{se,t} + \sum_{r \in er - \{r_D\}, er \in se, PA_{er}=0} w_{se,er,r} \geq u_{se} \quad \forall se \in SE$$

The sum of the active subevents' durations must equal the event's duration:

$$\sum_{se \in e} D_{se} * u_{se} = D_l \quad \forall e \in E$$

Analogously to the definition of active sub-events, we will also indicate if resources are *busy* at a given time. Intuitively, a resource is *busy* if it is assigned to at least one active sub-event (also called *solution event*) at time $t \in T$ and busy at time group $tg \in TG$ if it is busy for at least one $t \in tg$. Let $q_{r,t} \in \{0,1\}$ with $_{r,t} = 1$ iff $r \in R$ is busy at $t \in T$, let $q_{r,tg} \in \{0,1\}$ with $q_{r,tg} = 1$ iff $r \in R$ is busy in time group $tg \in TG$.
Specifically

$$|SE| * q_{r,t} \geq v_{t,r} \quad \forall r \in R, t \in T - \{t_D\}$$

ensures that $r \in R$ is considered busy in $t \in T$ if at least one sub-event does use it at that time. Furthermore, no more than $|SE|$ sub-events can use the resource simultaneously. While this bound could be chosen higher, this obviously is the lowest bound guaranteed to be valid in all cases.
Furthermore, a resource is not considered busy if it is not used:

$$q_{r,t} \leq v_{t,r} \quad \forall r \in R, t \in T - \{t_D\}$$

A resource $r \in R$ is used in $tg \in TG$ if $\exists t \in tg : r$ is busy in t.:

$$p_{r,tg} \geq q_{r,t} \quad \forall r \in R, tg \in TG, t \in tg$$

Conversely, a time group tg does not use a resource if there is no such $t \in tg$:

$$p_{r,tg} \leq \sum_{t \in tg} q_{r,t} \quad \forall r \in R, tg \in TG$$

3.1.4 Corrections

There are some issues with the model as stated in the original publication. Note that the author's implementation is correct; however, some constraints were not stated correctly in the paper. All corrections have been verified with the author.

1. **AvoidSplitAssignmentsConstraint**
$$\sum_{r \in R \setminus \{r_D\}} k_{c,eg,r} - 1 \leq s_{c,eg}^{\text{avoidsplitass}} \forall c \in C, eg \in c$$
The dummy resource r_D has to be excluded; $k_{c,eg,r} = 1$ iff event group $eg \in c$ is assigned to $r \in R$ by $c \in C$, therefore assigning some sub events to the dummy resource and others to just one teacher will cause the formulation not explicitly excluding r_D to (wrongfully) report simultaneous assignment of two teachers.

2. **LimitWorkloadConstraint**
$$U_{bottom,upper} \sum_{e \in c, t \in T, se \in e, er \in e} W_{e,se,er} * x_{se,t,er,r} \leq s_{c,r}^{limitworkload} \forall c \in C, r \in c$$
The dummy time t_D *must* be included as assigning a non-dummy resource to a sub event se will cause it to be active - regardless of whether a non-dummy time has been assigned. The workload is then considered to be assigned - the solver just did not specify when the workload is to be incurred.

3. **LimitIdleTimesConstraint**
$$|tg| - (|tg| - (p_{tg}(t)) * q_{r,t} \geq h_{r,tg}^{\text{first}} \forall r \in R, tg \in TG, t \in tg$$
as well as
$$p_{tg}(t) * q_{r,t} \leq h_{r,tg}^{\text{last}} \forall r \in R, tg \in TG, t \in tg$$
where
$$p_{tg} : T \rightarrow \mathbb{N}_{\not{\vdash}}, t \rightarrow p(t) - min\{p(t')|t' \in tg\}$$
The constraint must use relative indices as otherwise, the first index will be bounded by $|tg|$. Consider $|tg| = 7$ for weekday groups and consider the time group associated to Wednesday. As times are continuous in XHSTT, the first time of that time group could be $2 * 7 - 1 = 13$. However, as stated above, the index is still bounded by 7.

In the following we concisely state how to preset variables in the ILP model. While being intuitive, this needs to be defined.
Let $r_p \in R$ be a preassigned resource to $er \in e$ for $e \in E$. Then:

1. $w_{se,er,r} = 1$ of $r = r_p$
2. $w_{se,er,r} = 0$ if $r \neq r_p$ and $r \neq r_D$.

.

Likewise for $se \in e, e \in E$ starting at $t_s \in T$ set

1. $y_{se,t} = 1$ if $t = t_s$
2. $y_{se,t} = 0$ if $t \neq t_s$ and $t \neq t_D$

.

3.1.5 Model Inference

After solving the timetabling problem to optimality or to an approximation we will infer the solution from the MIP model variables. KHE not only provides functionality to read instances from XML files, but also to embed new solutions (or entire solution groups) into them. Additionally, rebuilding the solution with KHE allows to verify the correctness of the obtained solutions.
The definition of the basic variables in the previous chapters makes inferring the relevant data a snap. We need to infer the following data:

1. the *active* sub-events and their start time (if any) [4]
2. the resource assigned to each event resource for all event resources of any given *active* sub-event (if any)

Recall that a sub-event is *active* iff it is assigned a non-trivial resource or starting time (i.e. exclude t_D, r_D). Furthermore sub event $se \in SE$ is active iff $u_{se} = 1$ by definition.
Let $SE_a \subset SE$ be the active sub-events. Then,

$$A_t = \{(se, t) \in SE_a \times T - \{t_D\} \quad | \quad y_{se,t} = 1\}$$

Furthermore, we define

$$A_t = \{(se, er, r) \in SE_a \times ER \times R - \{r_D\} \quad | \quad er \in se, \quad w_{se,er,r} = 1\}$$

Those two sets provide us with the time and resource assignment for all active sub events. By definition, those assignments are unique.

3.2 Genetic Representation

While the LP model in stage 2 does already admit all necessary data, it is not in a suitable format for usage by stage 1. Therefore we will introduce an additional genetic encoding to be used by the genetic algorithm. The genetically encoded information can easily be added to the LP model by simply presetting the variables in question. Please refer to the subsection “transform of genetic information” for details.
The genetic algorithm preassigns teachers to courses. This preassignment is then transferred into the IP model by fixing certain variables. Afterwards, the IP model can be solved by regular means. We will use the parlance of XHSTT to describe the genetic representation.

[4]note that each sub-event does already have a fixed duration; this is specific to the MIP model

3.2.1 Encoding

There is a dedicated entity to represent the concept of courses in XHSTT: *Events*. Teachers are represented as *resources* of type “Teacher”. Each event only admits of a certain subset of all available resources. This reflects the notion that each teacher does have his personal set of classes he is entitled or willing to teach. Event-resource assignments are, among others, subject to constraint *AssignResourceConstraint* which penalizes non-conformance with resource assignment requirements. However, this constraint does allow for optional resource assignments - even for the teacher resource type. For simplicity, we will assume that *all* teacher resources there are to an event *must* be assigned. Please note that deviating events can be handled by introducing a special number such as -1.

Let $n_e \in \mathbb{N}$ be the number of events. For each event $e \in E$ let $T_e \subset R$ be the teachers who may teach this event. From now on, we will index T_e as an array T_e^i where $0 \leq i < |T_e|$, E_i is similarly defined. Then let $A \in \mathbb{N}^{e_n}$ be the *genome*, i.e. the component comprising the information to describe an instance in the population entirely. A is defined as follows: $A_i, i \in \{0, \dots, e_n - 1\}$ denotes the i-th entry in the array; A_i contains the index for T_e, thereby denoting the teacher assigned to the event E_i.

We will group the events in A by their subjects [5] while imposing an order within the groups by a measure developed in the subsection “Ordering within Subject Groups”. The interior order will be descending. We will see that this structure does yield some nice properties.

1., 2. and 3. in the following are requirements for a valid genome; all of the following is obviously enforced by this encoding:

1. Each event has an assigned teacher

2. Only up to one teacher can be assigned

3. The teacher can indeed teach this event

4. Changing any number inside its respective scope in A does yield a valid instance

   we do neglect constraint violations due to workload for now

5. Let A and B two genomes. Let $0 \leq j < e_n$, and

$$C_i = \begin{cases} A_i & i \leq j \\ B_i & otherwise \end{cases} \tag{3.1}$$

   Then C yields a valid instance again.

[5] Please note that XHSTT does not provide us with information about subjects. However, we can deduce the structural information by plain inference.

3.2.2 Initial Solutions

We use a traditional *urn model* for the generation of initial solutions. It turns out that *LimitWorkloadConstraints* tend to make the generation of feasible solutions particularly hard *if both lower and upper bound are specified.* Therefore, we will focus on satisfying the *LimitWorkloadConstraint.*
Let $B \subset T \times \mathbb{N}$ the set of all pairs $(t, i) \in T \times \mathbb{N}$ of teachers and their ideal workload. Then let

$$M := \{(t, j) |\, t \in T, j \in \mathbb{N}\, \text{and} (t, j + \text{maxGap}) \in B\}$$

be the set of all pairs of teachers with their mandatory workload. Falling short of the number specified in this set for any teacher incurs a hard constraint violation. Furthermore let

$$O := \{(t, 2 \times \text{maxGap}) |\, t \in T\}$$

be the set of all pairs of teachers with their optional workload, i.e. workload that *can* be taken on by the teacher, but does not have to be used up.
The initialization procedure works as follows:

1. For each subject s create a list

   $$L_s = \{(c, w) |\, c \in C, w\, \text{number of weeky lessons of s for class c}\}$$

   (pairs with $w = 0$ are omitted).

2. While M not empty:

   - Choose a random pair $(t, j) \in M$ by uniform distribution and remove it from M.

   - Choose a random subject s which t can teach such that $L_s \neq \emptyset$.

     If no such subject s exists, our solution will be infeasible; however we discard this teacher as no assignment is possible [6].

   - Otherwise select a class pair from L_s; if necessary use the corresponding pair in O to fill up needed workload and decrease the workload in O respectively in this case.

3. Assign remaining entries in L_s in random order (uniform distribution) to entries in O.

This procedure makes sure that all mandatory workload requirements are fulfilled before any other constraint is considered (see last step in procedure). That is, while it does not generally bar the creation of infeasible solutions it does instead make

[6]Please note that we do *not* require all initial solutions to be feasible in the first place

feasibility with respect to the workload constraint its main goal. As *previous* randomized assignments are crucial to the feasibility of the final solution, we may indeed still obtain infeasible solutions as there is no backtracking or roll back mechanism. Please note that this representation is rather general; the implementation in XHSTT and/or Roepstorffs' model (to be introduced later on) may differ slightly to fit existing concepts.

3.2.3 Mutation

Mutation is an unguided operator on the search space of all valid instances. In particular, it does not have the same degree of guidance as local search or other heuristic algorithms. However, adding more guidance to pure genetic algorithms is known as *memetic algorithms*.
We can easily define a basic mutation operator: Let $k \in \mathbb{N}$, then define Mut_k as the operator which changes k randomly selected events in A by randomly selecting another eligible teacher for each event. Note that it makes sense to bound k with e_n: let $1 \leq k < e_n$. The preceding subsection stated the requirements for validity and showed that this operator does indeed yield valid instances.
However, there is an apparent drawback. Suppose that the search space does have a smooth structure and a global optimum with most of the local optima in its vicinity. We are not aware of any research discussing the landscape of the teacher assignment sub problem of TT problems under mutation.
If this assumption proves to be incorrect, the mutation operator may loose its significance for finding the global optimum anyway. Assume that for $k \in \mathbb{N}$ and $A \in P$ the mutation yields $M_K(A) = A'$. Assuming that the quality of A was very good in the first place, this does in particular implicate a fair distribution of workload. Let $t \in R$ be a teacher affected by the mutation. Chances are, that the workload of t in A' either decreased or increased; w.l.o.g. let his workload decrease. Then the workload of another teacher who teaches the mutated event now has increased. In effect, this will throw the workload out of balance, the direct result being a higher cost. Working with the assumption that the search space has a favorable structure, chances are that we move the instances away from any optimum by mutation them. While mutation itself is unguided, designing a deteriorative operator does not seem to be particularly helpful.
Therefore we will not solely use a mutation operator but instead run a *smoothing operator* afterwards [7]. This operator tries to shift work from the teachers with the highest workload to their colleagues. Given a resource assignment, the shifting works as follows:

1. For each teacher, calculate the difference between his actual workload and his upper/bottom workload limit. [8]

[7]The concept is also called *repair operator* in some publications

[8]Note that the upper workload *limit* is calculated as optimalWorkload+maxGap where maxGap = 4 for the original instances

2. Sort the teachers according to their workload limit; this yields two queues `upperQueue` and `bottomQueue`
3. Select a teacher $t \in T$ alternatingly from `upperQueue` and `bottomQueue`. Note that `upperQueue` $\cap$ `bottomQueue` $= \emptyset$. Filter the other queue's teachers into tmpQueue such that $\forall tt \in$ tmpQueue $: U_t \cap U_{tt} \neq \emptyset$. Now, while tmpQueue is not empty and t is still overloaded, pop the top $tt \in T$ and shift workload from t to tt.

There are numerous ways to choose the exact workload to shift. Any choice we make might prove to hamper our algorithm for specific instances. Therefore, we choose both the class and the subject randomly using a uniform distribution. The implementation provides for an option to exclude certain moves. In particular, we will want to bar the smoothing operator from reversing a mutation we just applied. Given arbitrary assignments A and B, there is a set of atomic mutations m such that the application of m in arbitrary order on A equals B.
The selected event is shifted entirely from the source teacher to the destination teacher. We will explore running this smoothing operator several times after a mutation. If a mutation M_k with $k > 1$ is run, also smoothing several times is intuitive. This thesis does not explore the capabilities of the recombination operator. Therefore, the operator is not defined at this point.

3.3 Experimental results

Though XHSTT does admit preassignments of resources in any way, almost all instances in the current archive *XHSTT-2014* do not have any non-preassigned resources. In particular, there are only three instances which do have non-preassigned resources, namely AU-BG-98, AU-SA-96 and AU-TE-99. For experimentation purposes, we obviously prefer small instances. Furthermore, those three instances above can hardly be used to obtain statistically remotely significant results.
Therefore we wrote a *ModelImporter* which converts a given instance from the JSON format as defined by Roepstorff to the XHSTT-XML format. However, as the models' capabilities are not equivalent, the resulting instances are not solved using equal assumptions either (for instance, XHSTT currently does have no notion of regular breaks in its XML representation, and therefore no constraint requiring blocks not to span over those. The KHE library does provide that flag). Nevertheless, all of the major constraints can easily be formulated.
This converter allows us to use a handy, small file format for instance generation and then convert that to the much more complex XHSTT counterpart.
When running the converted instances with the ILP model, we discovered that the performance of the ILP solver for those solutions is disastrous indeed. The major hard constraints, i.e. *AssignTime*, *AssignRoom* and *AssignTeacher* as well as *AvoidClashes*, have been weighted with a cost of 1000 while other important hard constraints are weighted with just 1 (i.e. *PreferResourcesConstraint*). After more

than 350 seconds, the best incumbent solution for the tiny instance does have a *hard cost* of roughly 87, 000 (i.e.: roughly 80 assignment violations). For this run, only hard constraints were considered. The test machine has a quad core cpu and 8GB RAM. Note that this result has been verified with the author. On the same machine, it takes the standard heuristic solver shipped with KHE only 2.14s to find a feasible solution with 0 **soft cost**. The ILP solver by Roepstorff solving the original instance needs less than 0.1s (fully linearized version) to solve solely for the hard constraints. Above, all preassigned solutions have been randomly chosen. The dot "on top of the

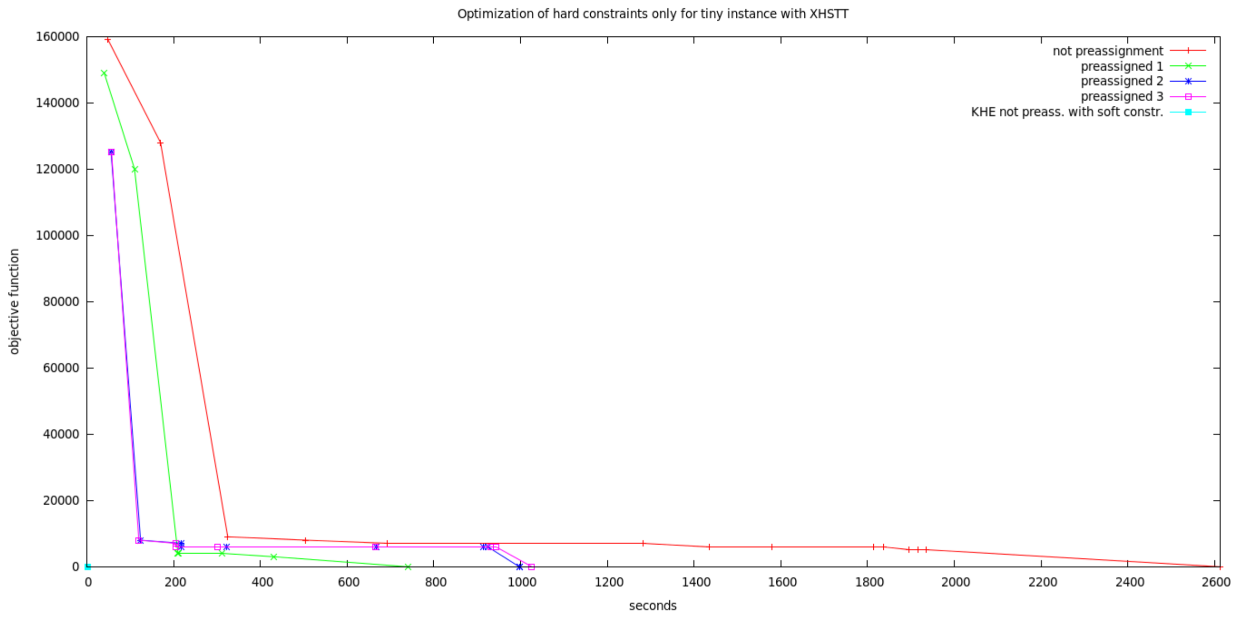


Figure 3.1: Evaluating solution time with ILP solver for unassigned and preassigned solutions. Its performance is compared with the heuristic solver shipped with the KHE library.

origin" is the result of the KHE solver. He does merely need 2.14 to solve with soft constraints and without preassignments to a cost of 0.
Simplifying the models by omitting non-essential constraints such as the *LimitWorkloadConstraint* did not yield sufficient speed improvements. [9]
Even omitting the *AssignRoomConstraint* did not decrease the solution time sufficiently: This type of erratic behavior does make the analysis particularly hard.
Omitting essential constraints (mainly *AssignTime*, *AssignRoom*, *AssignTeacher*,*AvoidClashes* and *PreferResourcesConstraint*) would deprive the model of its sense and was therefore not considered. In particular, omitting them should have the same effect as preassigning those resources (except for the *AvoidClashes* constraint).

[9]Note the aberration for the third assignment. We even observed some assignments took longer to solve for than not preassigned ones.

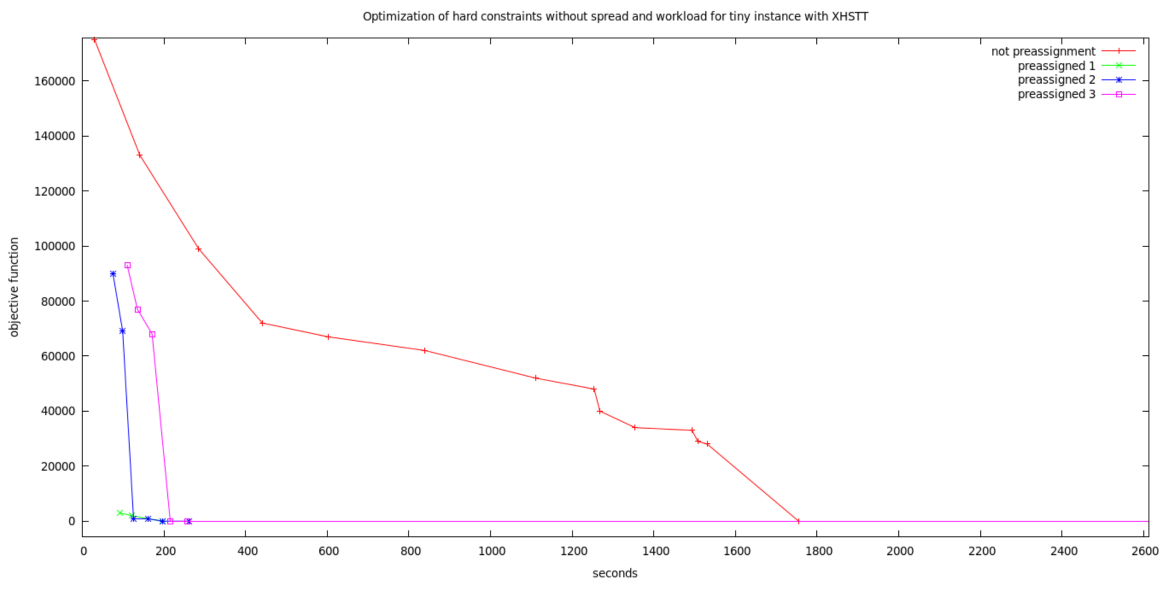


Figure 3.2: Performance of ILP solver when omitting *LimitWorkloadConstraint* and *SpreadEventsConstraint*. While the process *is* sped up, the improvements are *not* sufficient.

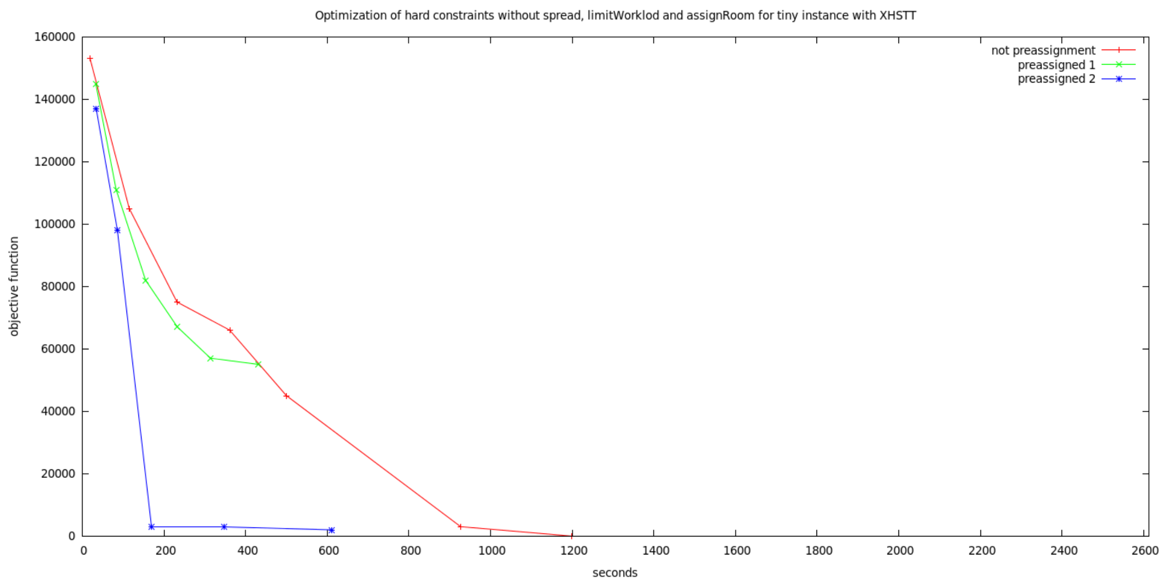


Figure 3.3: Additionally omit *AssignRoomConstraint* to assess speed improvements. It is straightforward to preassign rooms as they cannot be distinguished either (in the Roepstorff model). Again, the improvement is not sufficient.

To our surprise, preassigning resources did *not* improve the runtime enough. Because preassigning resources does imply constraining the solution space, it's hard to compare the ratio of solution quality and runtime as the former may be bounded by the pre-assignment. However, if the speed increased significantly, the bound should be reached faster. Unfortunately, as can be seen above, this was not the case. Note that the ModelImporter does allow to convert instances from the Roepstorff JSON format into XHSTT. The following graph shows the performance of the XHSTT ILP solver for some minimal synthetic instances. Instance *a_b_c* is an instance with *a* classes with each *b* lessons. *c* is a running index.

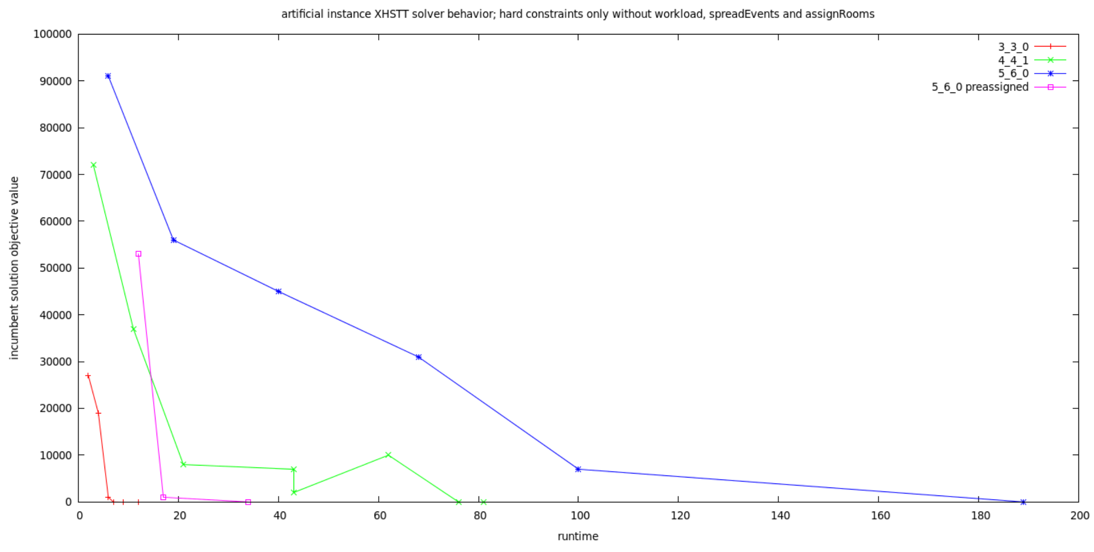


Figure 3.4: Incumbent Solution Time Analysis XHSTT for synthetic instances

Using the Roepstorff ILP model does take less than half a second for all of these instances - *with all soft and hard constraints.* [10]

In such cases the correctness of the solver itself may be questioned. If the solver cut off parts of the solution space due to an erroneous implementation, the behavior as described above would be a reasonable outcome. However, the IP formulation itself has been run against all XHSTT instances by its author, in many cases improving the optimal solution yet. To our knowledge, there are currently no instances for which the solver's best solution falls short of another solver *in the long run.*

And this may indeed constitute the gist of this problem; while correct ILP formulations are guaranteed to be solved to optimal solutions eventually, there is no guarantee whatsoever that good results will be obtained in reasonable time. In particular, we are dealing with ILP relaxations instead of general solution methods. Therefore, the performance will greatly depend on the quality of the ILP relaxation

[10] considering the last incumbent solution time. Please refer to the chapter on Roepstorffs model for details of this procedure.

rather than that of the LP solver itself. Indeed, the LP solver does find an optimal solution (i.e.: objective 0 by virtue of XHSTT's model definition) in only about 27 seconds for the relaxed problem. The remainder of the time as stated above is consumed by the relaxation procedure. While LPs can be solved in polynomial time with respect to the number of constraints and variables, this is **not** the case for ILPs. The understanding of why some ILPs behave significantly worse than others is currently very limited; furthermore an in-depth analysis of the reasons for this particular behavior does exceed the scope of this thesis.
While these results are unfortunate for research purposes, they do a good job illustrating pitfalls inherent to our approach. We cannot assume the IP part to behave reasonably for all instances we might come up with. No matter what the properties of the genetic component are, we cannot readily discard bad behavior of the IP part while assessing the overall performance. Obviously, the solution time as shown above is not suitable for our genetic approach. In genetics, the larger the population the better will be the results. Ideally, we will not consider tens or fifties of instances but instead hundreds or thousands. This is just not feasible with a slow IP component. We hope that though we ourselves do not have any use for the developed KHE C++ interface and IP model, it might help others with their research. In the end, the IP model is only one way to solve given preassigned resources - any XHSTT compatible solver can be harnessed instead, as after preassigning the resources by a heuristic or genetic procedure, it can simply be fed the resulting XHSTT file as if it were the original one.
As this approach did prove to be unappealing, we resort to the original model as used in Roepstorff's thesis [Rö13]. In the following, we will introduce the model and the customizations we made to it for the genetic model. In conjunction with the developed converter, our solver could still be applied to XHSTT, for instance with the standard KHE heuristic solver for the second stage.
In the classification of probabilistic algorithms, the presented one is neither a Las Vegas nor a Monte Carlo Algorithm ([Pas10]). However, it could easily be extended to a Las Vegas algorithm: The variable count of the IP formulation given in [Sk14] has a finite number of variables and covers the entire search space. Therefore, the solution set is *countable* in the mathematical sense. If we were unable to find a feasible solution within a certain time limit, we could drop the genetic component and let the IP solver work on the IP formulation itself. The solver (for this thesis: Gurobi) is guaranteed to find a solution in a finite time; however this time is not bounded. Please note that the search space is probably very large; for instance, for BGHS98 (AU-BG-98) [11] the search space has the hard upper bound $(40 * 56 * 45)^3 87$ which is roughly $2.2 * 10^1 936$.

[11] http://www.utwente.nl/ctit/hstt/datasets/Australia/BGHS98/, taken on 01.05.2015

3.4 Roepstorffś Model

We will introduce the Roepstorff model in the following. As they tackle the same type of problem, the Roepstorff model and the XHSTT model do have much in common.In particular, **timeslots** in XHSTT are more general versions of their counterparts. The ensuing chapter outlines the major conceptual differences, we do only point out differences and refrain from introducing the entities all over again.

3.4.1 Basic concepts

Subjects may or may not require specialist rooms. For instance, Biology might require to be taught in a special room. For each subject, there is a given number of specialist rooms available; thereby, the number of concurrent lessons for such subjects is limited.
Classes are assigned a number of subjects along with the number of hours each subject is to be taught per week. They do have an individual grade which may or may not impose additional constraints; for instance, one might want to reduce late hours for classes of low grade.
Teachers are accompanied by a list of subjects they can teach. Furthermore, they do have a minimum and maximum workload; this is enforced as a *hard constraint.*

3.4.2 Basic variables

Each solution in the solution space can be uniquely identified by the following variable definitions; all other variables such as slack variables can be deduced from those. In particular, those are the only variables we will be concerned with for the genetic part.
We define C as the set of classes, U the set of subjects, S the set of time slots and T the sets of teachers. U_t for $t \in T$ designates the subjects which can be taught by t while U_c for $c \in C$ designates the subjects c is to be taught. T_s for $s \in U$ designates the subset of T of teachers who can teach s.

$$X_{i,j,k} = \begin{cases} 1 & \text{class i has subject j in time slot k} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$y_{i,j,t} = \begin{cases} 1 & \text{if teacher t teaches class i in subject j} \\ 0 & otherwise \end{cases} \tag{3.3}$$

3.4.3 Representation of hard Constraints

Classes and teachers can only have on lesson simultaneously; for classes, the constraint is straightforward. For teachers, we have to combine the two basic variables: First, the constraint for classes:

$$\sum_{j \in U} x_{i,j,k} \leq 1 \quad \forall i \in C, k \in S$$

and for teachers respectively:

$$\sum_{i\in C}\sum_{j\in U_t} x_{i,j,k} * y_{i,j,t} \leq 1 \quad \forall t \in T, k \in S$$

Furthermore, each class must be taught exactly the given number of lessons for each of its subjects. Let $lpw(i,j)$ for class i and subject j denote the number of lessons to be taught:

$$\sum_{k\in S} x_{i,j,k} = lpw(i,j) \quad \forall i \in C, j \in U$$

Each class is taught by one and only one teacher in each of its subjects:

$$\sum_{t\in T_s} y_{c,j,t} = 1 \quad \forall c \in C, s \in U_c$$

Teachers can only give lessons in a certain subset of all subjects:

$$\sum_{c\in C} y_{c,s,t} = 0 \quad \forall t \in T, s \in U - U_t$$

There is only a limited number of specialist rooms; therefore the room usage of each specialist rooms for teach time has to be limited; let $nor(s)$ be the number of specialist rooms for subject $s \in U$. $U_s \subset U$ are all subjects which require a special room. Then:

$$\sum_{c\in C} x_{c,j,k} \leq nor(j) \quad \forall j \in U_s, k \in S$$

In the original paper, the deviation of a teacher's workload from his ideal workload is constrainted as follows; let $maxGap = 4$, then:

$$\sum_{c\in C}\sum_{j\in U_t} y_{c,j,t} * lpw(c,j) \geq lpw(t) - \text{maxGap} \quad \forall t \in T$$

$$\sum_{c\in C}\sum_{j\in U_t} y_{c.k.t} * lpw(c,j) \leq lpw(t) + \text{maxGap} \quad \forall t \in T$$

However, genetic algorithms work by *spraying* the solution space. Following this approach we would like to deliberately accept bad solutions. Those solutions *might* lead us to regions we didn't explore before. Frequenty dismissing bad solutions might let an algorithm end up stuck in a "valley" in the fitness landscape.
In particular, violations of non-relaxed hard constraints yield an *infeasible* model, i.e. the IP solver's response is just that the provided instance is indeed infeasible. This does not contain any information on *how bad* the instance is. While it might be on the brink of a major new (locally) optimal solution, we would not explore it.

Therefore, we *relax* the workload deviation constraint. Let $slack_w \in \mathbb{Z}$ be a new slack variable bounded by

$$-\text{maxslackRange} \leq slack_w \leq \text{maxslackRange}$$

for maxslackRange $\in \mathbb{N}$. Furthermore, introduce helper variable *overloadPenalize* subject to

$$\text{overloadPenalize} \geq slack_w \land \text{overloadPenalize} \geq -slack_w$$

This helper variable represents the magnitude of deviation; it will be highly penalized (e.g. value 1,000). The new constraint is as follows:

$$\sum_{c \in C} \sum_{j \in U_t} y_{c,j,t} * lpw(c,j) + slack_w \geq lpw(t) - \text{maxGap} \quad \forall t \in T$$

$$\sum_{c \in C} \sum_{j \in U_t} y_{c.k.t} * lpw(c,j) - slack_w \leq lpw(t) + \text{maxGap} \quad \forall t \in T$$

In the end, we would like to limit the lessons for each class for a particular subject to two per day. let S_{Mon}, ..., S_{Fri} denote the subsets of T containing all time slots for each day in question. Let S_{days} comprise all of these sets. Then:

$$\sum_{k \in S_{\text{day}}} x_{i,j,k} \leq 2 \quad \forall i \in C, j \in U, S_{\text{day}} \in S_{\text{days}}$$

3.4.4 Representation of soft Constraints

The original paper does introduce more soft constraints than we will in this thesis. We did not consider those constraints in the instance generator because generating them in a meaningful way such that their impact on solution quality is predictable is not intuitive [12]. Those constraints are teacher's preferences (TP) are "same teacher as last term' (ST).
Furthermore, some constraints are only given in their linearized form (i.e. both-subjects-constraint (TS), timetable compactness constraint (TC) as well as the female-male teachers constraint). In order to implement the genetic component, dealing with the intricacies of the linearized model is not necessary; the basic variables remain the same.
w_x where x is an abbreviation for a constraint specifies the weight of the constraint with respect to the cost function.
The remaining constraints are the following:

double lessons (DL)

[12] Furthermore, the original implementation does not implement a means to store constraint related information with the instance itself.

Let C_{hg} be the set of classes this constraint is to be applied to. As already mentioned, $lpw(i,j)$ denotes the number of lessons class i is to be given in subject j. Then the number of blocks is naturally

$$\lfloor \frac{lpw(i,j)}{2} \rfloor$$

Furthermore let

$$S_{\text{blocks}} = \{(k_1, k_2) \quad | \quad k_1, k_2 \in S, \text{k1 and k2 are one block}\}$$

be the set of blocks. Then the cost incurred by this constraint equals

$$\text{DL} = w_{DL} * \sum_{i \in C_{hg}} \sum_{j \in U} (\sum_{(k_1,k_2) \in S_{blocks}} x_{i,j,k_1} * x_{i,j,k_2}) - \lfloor \frac{lpw(i,j)}{2} \rfloor$$

Friday short workday
Let k_{f7} and k_{f8} be the last two slots on Friday. As having lessons in the eighth hour is even more undesirably than in the 7th hour, we will weight this slot even more.

$$\text{FL} = W_{\text{FL}} * \sum_{i \in C} \sum_{j \in U} x_{i,j,k_{f7}} + 3 * x_{i,j,k_{f8}}$$

3.4.5 Instance generation

As already mentioned, large TT instances are generally very hard to solve. When evaluating genetic operators, genetic algorithms or fitness landscapes in general, time consumption is of the essence. Therefore, we generate small instances which can be evaluated exhaustively with minimal computing power; we hope that results on the small scale may be extended to large instances.
There might be intuitive approaches on how to generate small instances for continuous problems such that the essential characteristics of large instances may be retained. In particular, it may be intuitive that properties found to hold for small instances do extend to larger ones.
This is obviously not the case for discrete problems and in particular for the TT problems at hand. While we will present an approach for instance generation, we feel obliged to point out that we cannot rule out systematic bias. Therefore, care has to be taken when interpreting the results. This becomes particularly apparent when taking the statistically insignificant amount of real world testing instances for the Roepstorff model into account.

Main Parameters

We try to control the hardness of a sample instances with two parameters: $n_c \in \mathbb{N}$ the number of classes and $n_s \in \mathbb{N}$ the number of subjects for each class. Due to the *LimitWorkloadConstraint*, the number of teachers depends on the total workload which in turn is characterized by the classes and their subjects. Generally, the more subjects and classes are to be scheduled, the harder is the instance. Consider a number $n \in \mathbb{N}$ of events (i.e. lessons) to be scheduled. *Decreasing* the number of classes will *increase* the instance's hardness due to the *AvoidClashConstraint* (conversely an increase will facilitate solving). Keeping the number of classes fixed while changing the number of events will have the same effect. Therefore, n_c, n_s are a natural choice for characterizing instance complexity for generation purposes.

Class Generation

The model draws from high schools in Germany. Classes' lessons are mostly determined by the grade of the class [13]. Therefore, we can define a set of *profiles*; each profile determines the subjects the class can be taught as well as the associated duration.

Drawing Subjects

For each class, n_s pairwise distinct subjects are drawn from the profile at hand. The subjects' probabilities are distributed uniformly weighted by their total workload. This reflects the fact that more important subjects (which are generally more likely to be assigned, regardless of grade) tend to have longer durations. [14]

Recruitment of Teachers

This part is particularly crucial, because the *LimitWorkloadConstraint* produces hard constraint violations if we do not assign workload carefully. Obviously, we must neither assign too few teachers nor too many teachers. Furthermore, the same is true for the ideal workload we assign each teacher. Failing to take heed of that will result in infeasible instances.
Alike the *class profiles* described above we generate *teacher profiles* from real world instances.
It turned out that a multitude of teacher profiles is beneficial to generating reasonable teacher workloads. Essentially, those profiles only feature a subject combination and a probability to be chosen. When generating arbitrary instances, the available subject combinations for teachers determine which teachers we can use to exhaust the mandatory workload. Our intuitive assignment process (during the generation) draws teachers with much available workload first. The remaining subjects proved to be *sparse*, i.e. initially there were no teachers covering up several subjects at once. This results in assigning 2-3 more teachers than necessary, each taking up only a workload of less than 10 hours. Therefore we inflated the available subject

[13] Though local governments keep pushing reforms; this is at least how it used to be. Note that *scientific profiles* such as STEM, language and culture can be modeled by adding an extra profile.

[14] There is a reason why the subjects' probabilities do *not* depend on the class profile. When generating small instances, using individual probabilities will yield to less shared subjects. This impedes the "swapability" of teachers when scaling down instances. However, class profiles still determine *which* subjects a class *can* be taught.

combinations by importing recommended (clash free) combinations from the CAU Kiel. This measure brought down extraneous teachers to one or less per instance.
We decided not to scale down lessons' durations because it diminished differences in lesson's durations to barely observable remainders; this is due to the discrete nature of this problem. When not scaling down lesson's durations, teachers' workloads cannot be scaled down either.
We don't want to fragment our ideal workloads too much. This is because we assume most teachers to be full time employees (i.e. around 25 teaching hours a week) and only a small subset part time employees (e.g. parents taking time off). We therefore end up with only two different prevalent ideal workloads: 15 and 25 hours a week. The workload is determined by the minimum of an ideal workload and the remaining workload a teacher can take up. The ideal workload is chosen by random weighted uniform dist. from $\{15, 25\}$ with probabilities $80\%, 20\%$ respectively. Furthermore we cut off times at the end of the week to maintain the same level of hardness by the *AvoidClashesConstraint*: the ratio $\frac{5\times8\times3}{\text{totalWorkload}}$ multiplied by the sample instance's workload is the number of times for the sample instance [15]

- While there is workload to assign, draw a teacher t such that there is workload left for at least one of his subjects; teachers are weighted exponentially with their available workload.
- Determine his workload W_t: The ideal workload W_i is drawn as described above. Define

  $$W_s := \text{Sum of remaining workload for his subjects.}$$

  We then choose
  $$W_t = min\{W_i, W_s\}.$$

This procedure ensures that all lessons are taught and no teacher is bound to violate a hard constraint.

[15] factor 3 is a result of our experience. Using smaller values resulted in infeasible solutions too often.

4 Results

4.1 Smoothing effects

An effective smoothing procedure is commonly used in genetic approaches to timetabling. Its implementation and effectiveness greatly influences the assessment of candidate solutions. Figure 4.2 and 4.3 show how the smoother propels the assignments iteratively to better objective values; Figure 4.1 puts emphasis on the cumulative analysis.

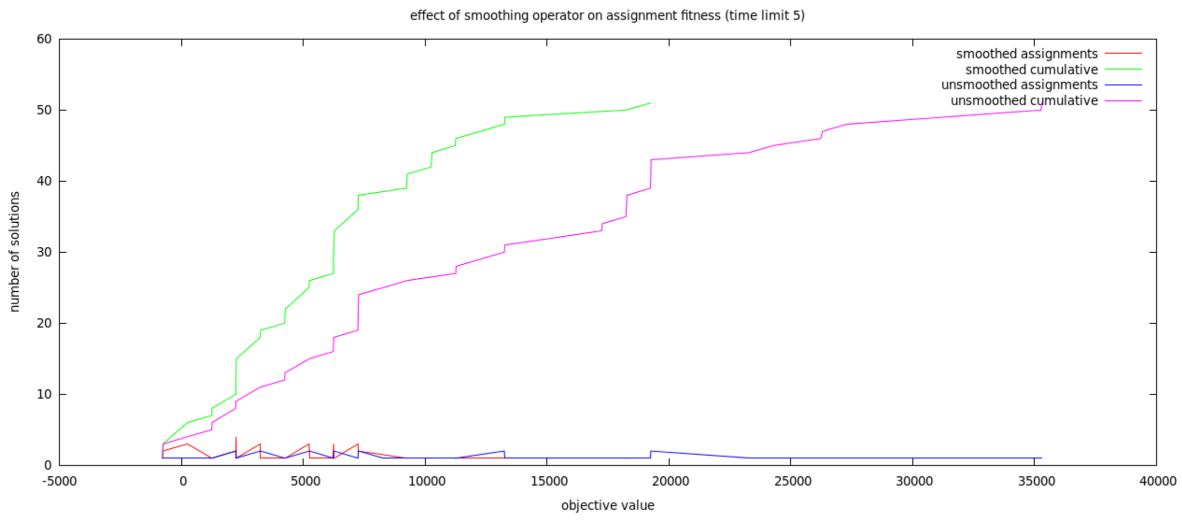


Figure 4.1: s
(Cumulative) Distribution of objective values of 50 randomly drawn solutions (with-/without smoothing) from the tiny instance. Having been smoothed, 40 out of 50 instances' objectives are bounded by $7,500$. The corresponding objective value for the unsmoothed solutions is about $20,000$.

Note that the smoother does not guarantee to improve solutions - he could even worsen them. Reassigning a teacher might, given special constraint weighting, actually worsen the objective value.
We will use the smoother to "smooth" random solutions. Particularly randomly chosen assignments may be arbitrarily bad; we would like to have a good assessment of how good the solution or its vicinity is. On the other hand, too much smoothing transforms our algorithm into a mere heuristic. The graphs above show that the smoothing does transform solutions such that the remaining hard constraints vio-

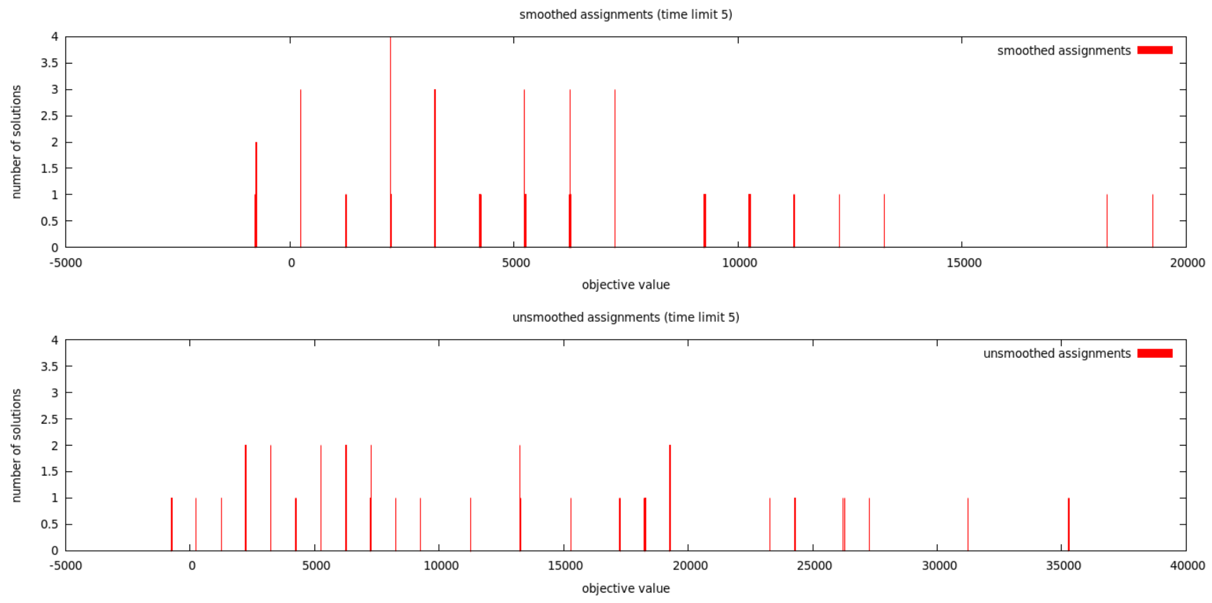


Figure 4.2: Direct comparison of unsmoothed solutions and smoothed solutions. The smoothing operator was limited to two steps. The effect on very bad solutions becomes particularly apparent when considering the scaling of the x axis.

lations are moderate, i.e. feasible or nearly feasible. Furthermore already "good" solutions are not smoothed, especially this process does not distort the behavior of other operators for feasible or nearly feasible solutions. This serves as a good basis for further search operators.

4.2 Runtime behavior

When evaluating whether evolutionary components can speed up a bare IP approach, their respective runtime ratio is apparently one of the first facts to check. I.e. how much faster is the IP solver when a preassignment to resources is made? Roepstorff"s thesis already reported the total runtime for the IP part. However, IP solvers work with *heuristics* or at least *iteratively*. This means that during the process of solving temporary solutions are already available. Those are also called *incumbent* solutions. The incumbent solutions are continuously improved until the optimal solution is found, a time limit is reached or it has been proven that no better solution can exist. Consider the *solution courses* with 50 randomized, smoothed preassignments for the tiny instance:
Notice that while incumbent solutions are frequently found in a very short period at the beginning, no new incumbent solutions are found thereafter. In the following, we will generally not prolong last incumbent solutions to the very end as this greatly

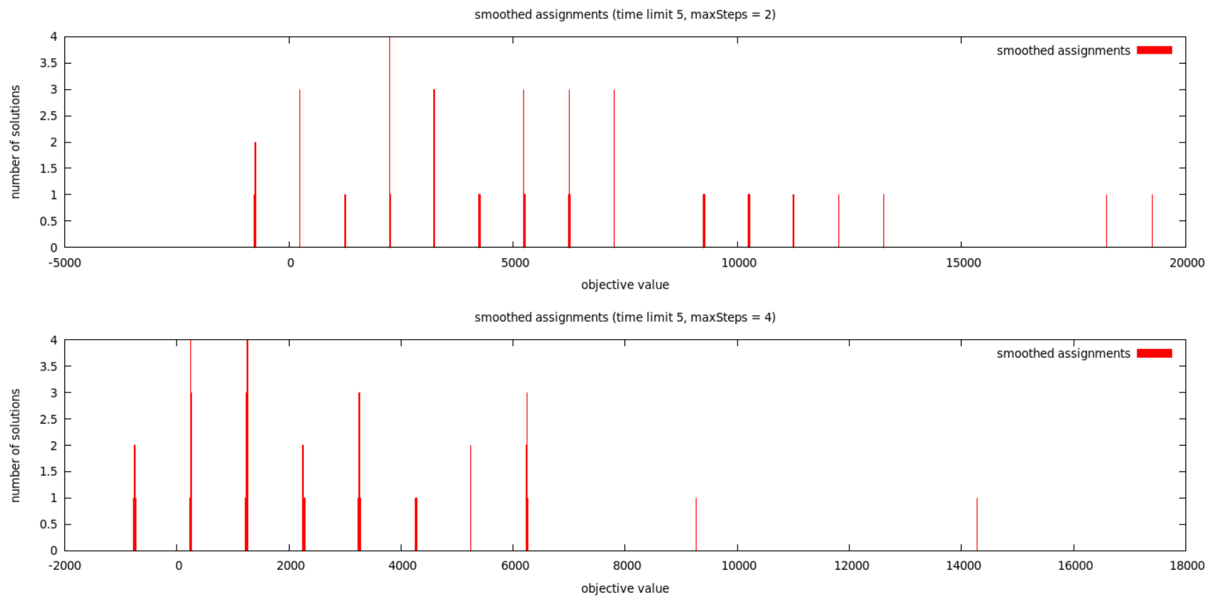


Figure 4.3: Comparison of smoothing effects with more steps (graph above: 0, 2 steps, this graph: 2, 4 steps). "Good" solutions are rarely improved whereas "bad" solutions virtually vanish. The trend of solutions towards the left region continuous when smoothing even more.

decreases readability. [1] The behavior that after a short period no new incumbent solutions are found has been verified with larger instances. They do eventually terminate *without* having found any new solutions. We will use this observation to justify some assumptions later on. In particular we will only analyze the last incumbent solution or a derivative of it; we work with it instead of proven optimal results.

4.2.1 Constraint importance

We can clearly see that while preassigning does decrease the time needed for solving, it is clearly not apt for finding a solution genetically. We might fit up to five or six assignments into the time needed to solve for the optimal solution. That is nowhere near enough to perform genetic optimization. We already narrowed our consideration down to the incumbent solution, so there is no way to dig further. Furthermore, we can see that there is no point in aborting prematurely (i.e. before reaching the last incumbent solution), as the gain is minimal.
However, an important, distinct feature between preassigned solutions and the optimal solution does exist. The optimal solution is meant to be deployed to a real world school, where assignments are to be adequately assessed *for the moment.* Clearly

[1]This behavior can be controlled by the stretchLastIncumbent option of BulkInstanceAnalyzer

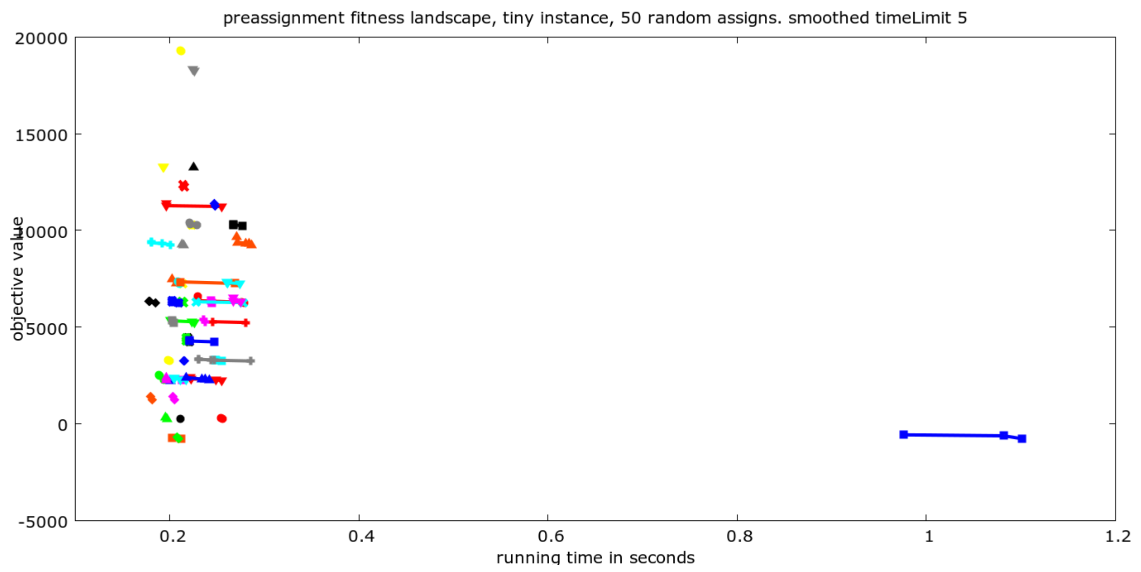
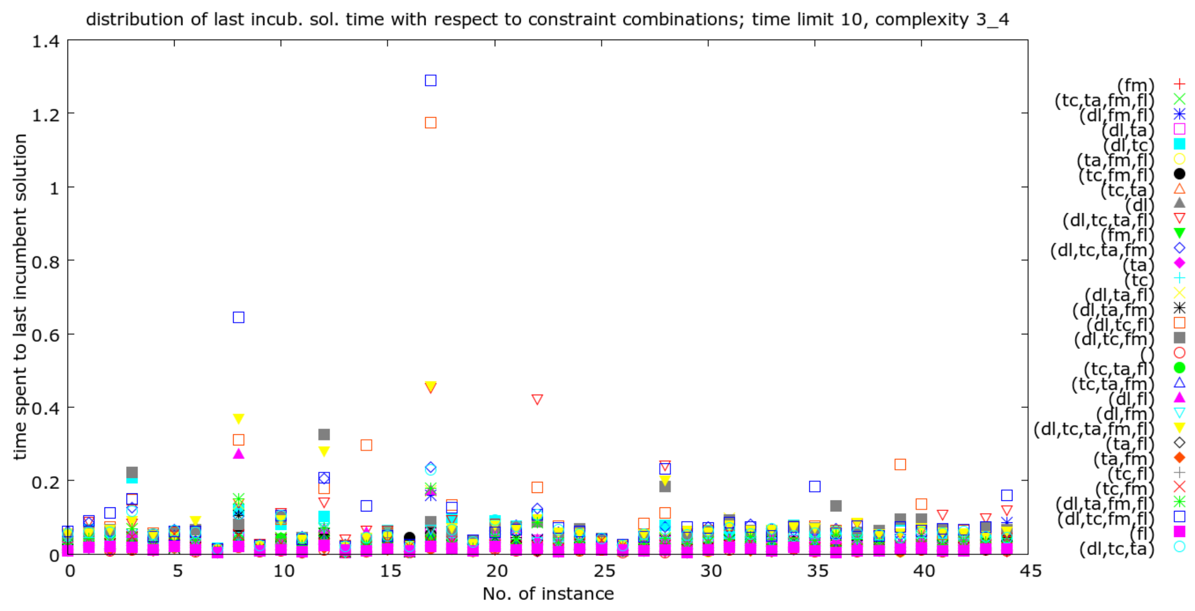


Figure 4.4: Objective course of 50 randomly drawn, smoothed solutions from the tiny instance. The blue line at the very right represents the course of the optimal solution process, that is without having assigned any teachers. All other courses on the left represent solution processes with preassigned teachers. Each point represents a new incumbent solution; the lines in between have been added to visualize the objective course; they can be interpreted as interpolations of the solution process. The overall runtime limit is set to 5 seconds. A solution's course not reaching the time limit does not implicate that the solver is finished. Instead the solver just did not find any new incumbent solutions up to the time limit.

when choosing an assignment we have to solve it just like the optimal solution, however in order to find this particular assignment, we only need a means to assess assignments *fast*.

In the following, we analyze the impact of the soft constraints on the running time. **If** we were to find out that there is a strong correlation between objective value without a certain constraint and objective value with all constraints, we might skip that constraint *during assessment* of the assignments. Once chosen, we can add that constraint and solve *only this assignment* with all constraints.

To put this in another way; consider the objectives associated with a set of preassigned solutions. Suppose they were assessed with one constraint less. Order the solutions according to their objectives. If we reassess those solutions with all constraints - will the relative order be retained?

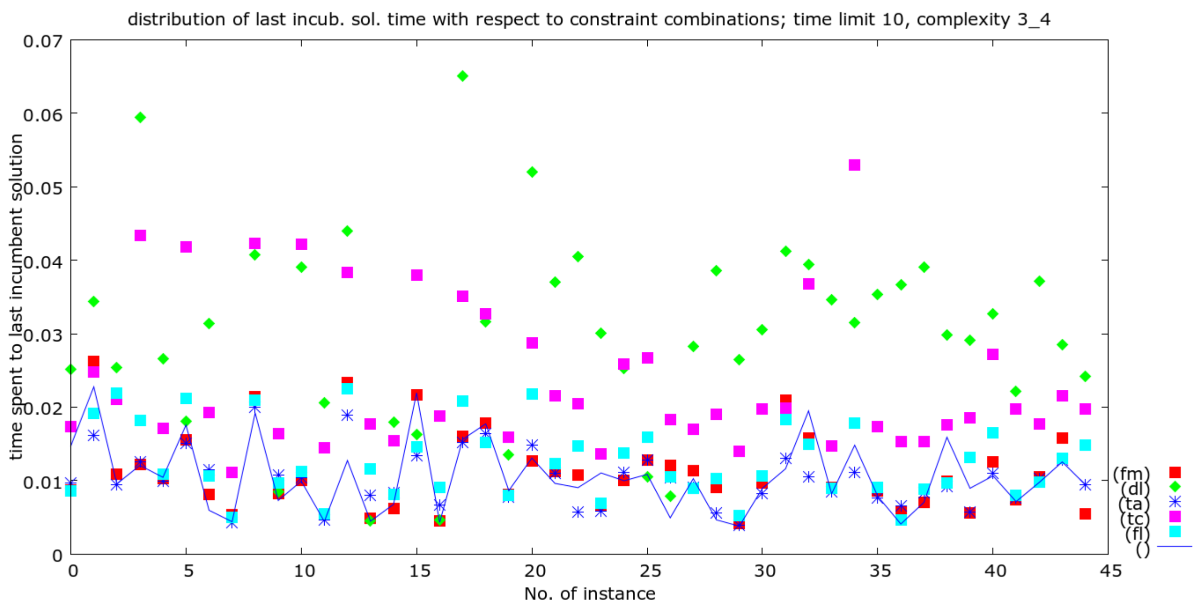


[2]

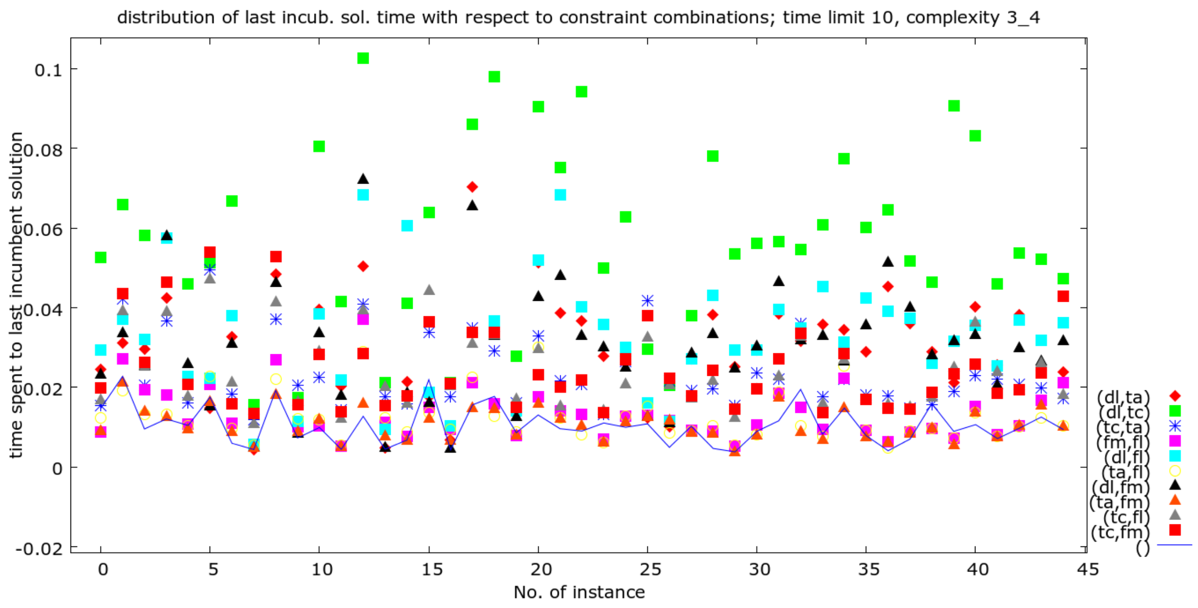
The analysis above shows that all soft constraint combinations induce similar running times. There is no combination that does incur extreme running times.

We first analyze the impact of *adding* one soft constraint to the bare hard constrained solution. This gives us an idea of how hard certain constraints are. Then we will observe the objective value correlation when *excluding* just that constraint. Below is an analysis of each soft constraint's impact on solution time for the last incumbent solution:

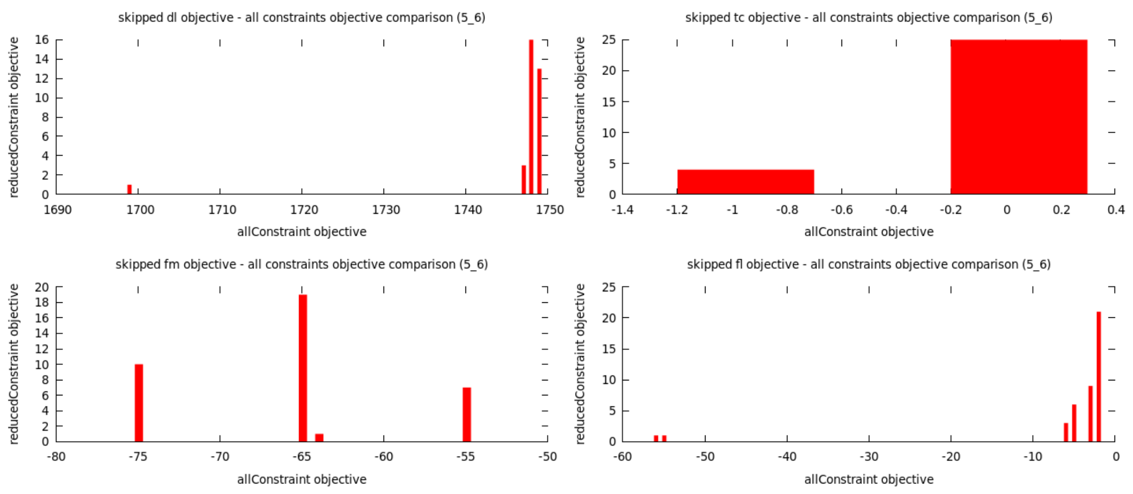
[2] fl = shortFriday, fm = femaleMale, tc = TimetableCompactness, ta = TeacherAllSubjects, dl = DoubleLessons



The impact of *DoubleLessonConstraint* and *TimeCompactnessConstranit* sticks out. This is backed up by the science community. From other TT approaches and instances, those constraints are known to be particularly hard to solve for.
Furthermore, certain constraints may influence other constraint's hardness. Therefore, the following analyzes those relationships:



We can clearly see that the *DoubleLessonsConstraint* does have a huge, negative impact by itself and in conjunction with other constraints. The following graph shows the distributions of differences between a objectives obtained by omitting a specific constraint and the objective with all constraints. These distributions all pertain to preassigned solutions.



Omitting the *DoubleLessonsConstraint* constraint seems very promising; the deviations are all of the same nature (positive) and most of them are distributed in a very small region. Therefore omitting this constraint will likely not “shuffle” our objective values. As the *FridayLover* (short Friday) constraint exhibits a similar distribution, the same applies to him.
The *TimeCompactness* constraint on the other hand does span the origin. In general we will not retain the relative order induced by the objective values. Therefore we should dismiss omitting this constraint.
The *FemaleMale* deviations are all located on the left side of the origin; however, their span is rather huge. If this constraint can be omitted will heavily depend on the constraint weights.

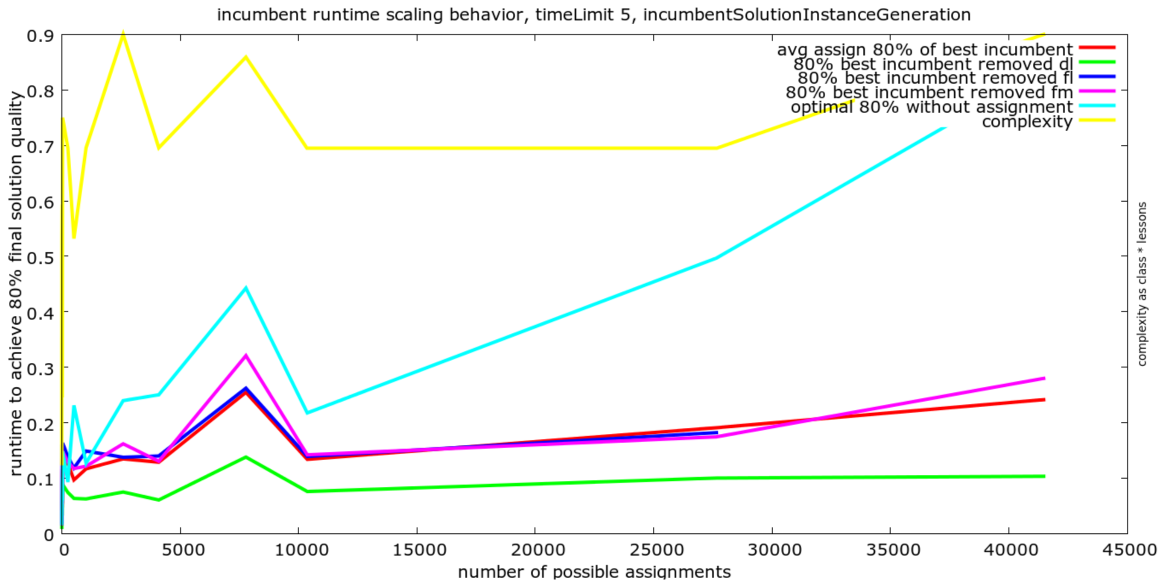
4.2.2 Scaling of instance complexity

Now we have identified the constraints which could be omitted. However, there is still no proof that the relationship obtained for small instances above will extend to larger scales. For this purpose we use a specific test set:

- define a list of ascending hardness levels (i.e. $n_c \times n_s$[3] ascending)
- for each hardness level, create 3 random instances using the generator
- for each instance, draw 50 random assignments and smooth them
- for each assignment, calculate the running time until the first incumbent solution has achieved at least 80% of the *best incumbent solution's*[4] objective value.
- average over the 50 instances

[3]number classes, number subjects
[4]still for this assignment

The above-mentioned procedure yields the following result:



Even with the smoothing process we do see erratic behavior on the smaller scale. This will not only be due to randomized noise but rather a peculiarity to the generated instances. We can clearly see that the removal of all other constraints than the DL (DoubleLessons) constraint does have no tangible effect [5]. However, DL obviously does have a great effect on solution time. Apparently the optimization with all constraints does have an at least exponential time consumption. Please note that the x axis denotes the number of possible assignments and *neither the x nor the y axis is logarithmically scaled.* When adding classes or subjects, the number of options increases exponentially. With that in mind, the steep rise of the optimal solution course suggests a strong exponential rise with increasing instance complexity. We can only fit a few solution procedures with the DL constraint omitted into the same time span needed for one solution with all constraints: however this number increases dramatically as the scale of the instances goes up.

4.3 Fitness Landscape analysis

As stated in the introduction, this thesis sets out to explore the *feasibility* of a genetic approach to TT problems. As such we strive to understand the behavior of the genetic operators (i.e. mutation and recombination) on the *fitness landscape.*

4.3.1 Mutation landscape

In the following, we analyze the local landscape, i.e. the neighboring area of an individual solution. We move from the original assignment to its immediate neigh-

[5] note: that solutions without a constraint sometimes take longer than with all constraints should be attributed to noise

bors by applying the mutation operator once. Accordingly, secondary neighbors are reached by applying the operator twice. We will not consider mutations which reverse previous mutations.
This approach allows us to verify or reject a conjecture: We would prefer a strong correlation between the solution quality and the quality of its surroundings. [6]
This does color the way we think about our mutation operator. If the vicinity's assignments are strongly correlated with the original assignment's objective value, we will think of mutation as moving through a "continuous" fitness landscape. If we were to find out that there is no strong relation, we would rather label the operator's behavior "jumping" through the solution space.
First of all, we consider the tiny instance with minimum smoothing applied; the vicinity analysis for randomly chosen assignments is shown in figure 4.5.

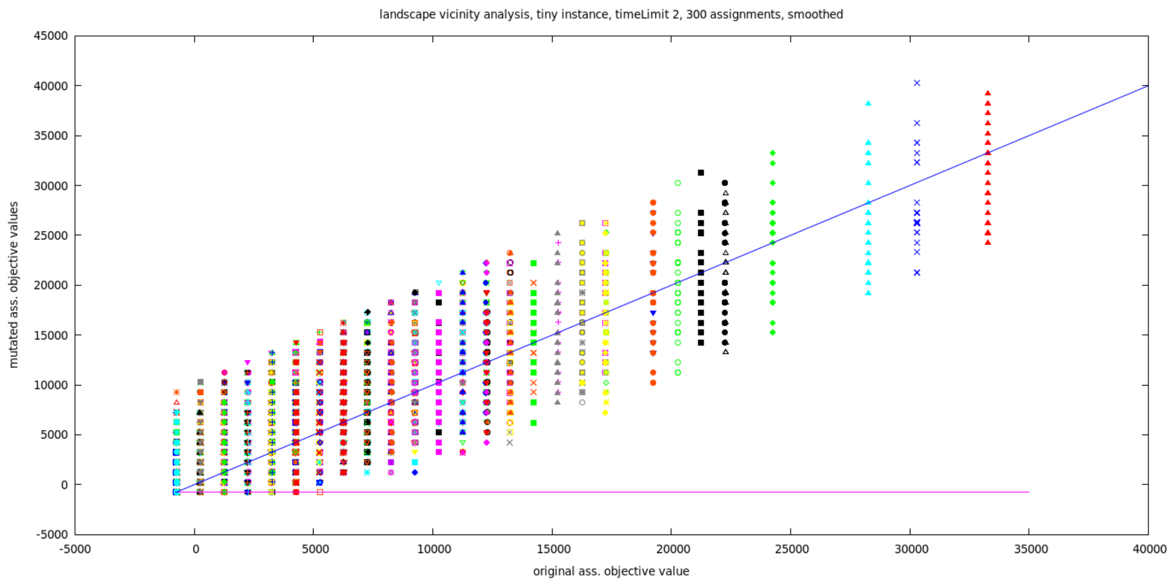


Figure 4.5: Objective value distribution in local vicinity; The bottom line indicates the absolute best objective value, the diagonal the original objective value. Each vertical column represents the vicinity of a randomly chosen assignment. The vicinity is immediate, i.e. mutation is applied only once. The vicinities have been explored exhaustively.

We observe that the fitness landscape in the close vicinity is very structured; it is obviously mainly influenced by workload deviation violations [7]. Furthermore we observe that the solutions are clustered in the left region, having a moderate count of violations. Several assignments with the same pairs of original and mutated assignment objective value overlay each other.

[6]We think of "approaching" an optimum as building a chain of neighboring solutions with respect to the mutation operator.
[7]Those are the only ones weighted with 1,000

We can easily deduce the landscape of multi step mutations. Consider the projections of the neighboring solutions' objectives to the left y axis. The intersections of those spans are, probabilistically interpreted, solutions that can be reached from both original solutions by mutation [8]. Conversely, the union of three intersecting landscapes leads to the vicinity of the original solution with respect to a step of two mutations. Constructing higher mutations' vicinities is straightforward.
This reasoning makes clear that, according to the graph above for the tiny instance, while single or double mutations are bounded by the original assignment's objective value, the maximal deviation for more mutations spans about the entire possible range. In conjunction with the analysis in figure 4.8 below (elaborating on probabilities), we can see that more than two mutations until assessing the incumbent solution is bound to be "erratic" and not suitable for our needs. Typical vicinity fitness landscapes are shown in figures 4.6 and 4.7; they represent a "good" and a "bad" solution's immediate vicinity. The "good" solution's vicinity appears to have a benign complexity; when searching its vicinity, the better the objective value the higher the likelihood of reaching it. On the other and, the "bad" solution's vicinity does choosing "bad" solutions much more likely.

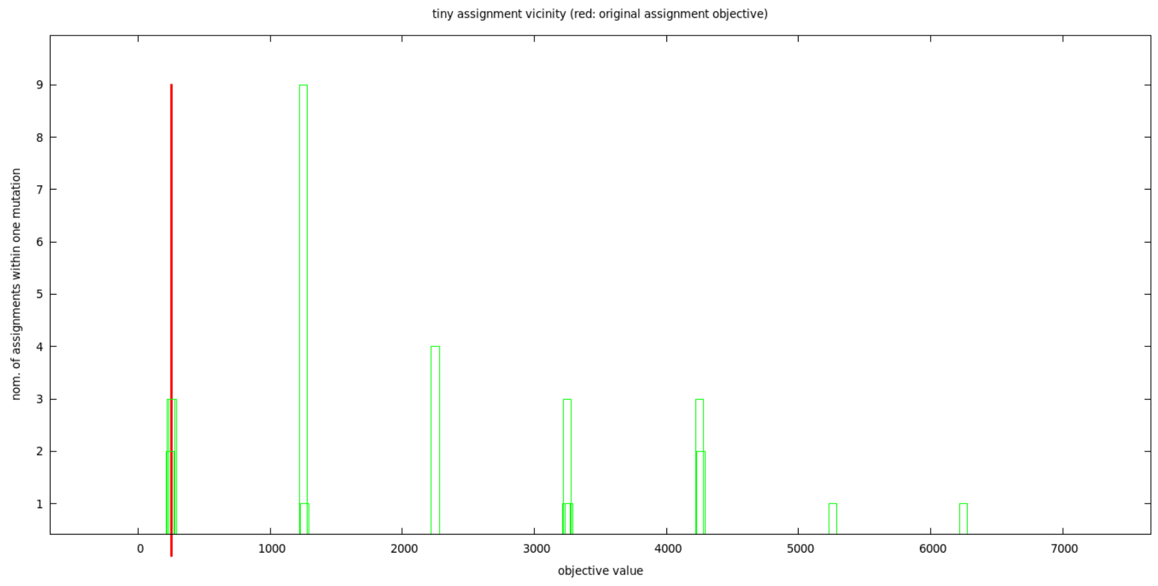


Figure 4.6: tiny instance with assignment of objective 250; the red vertical line indicates the objective value of the chosen assignment.

The average objectives in the vicinity "descend" as the original objective improves. Figure 4.8 shows a more detailed analysis; it outlines the percentage of assignments in the direct vicinity of the original assignment having an equal or better objective value.

[8]Only the observation that the solution objectives' structure is very regular makes this proceeding reasonable

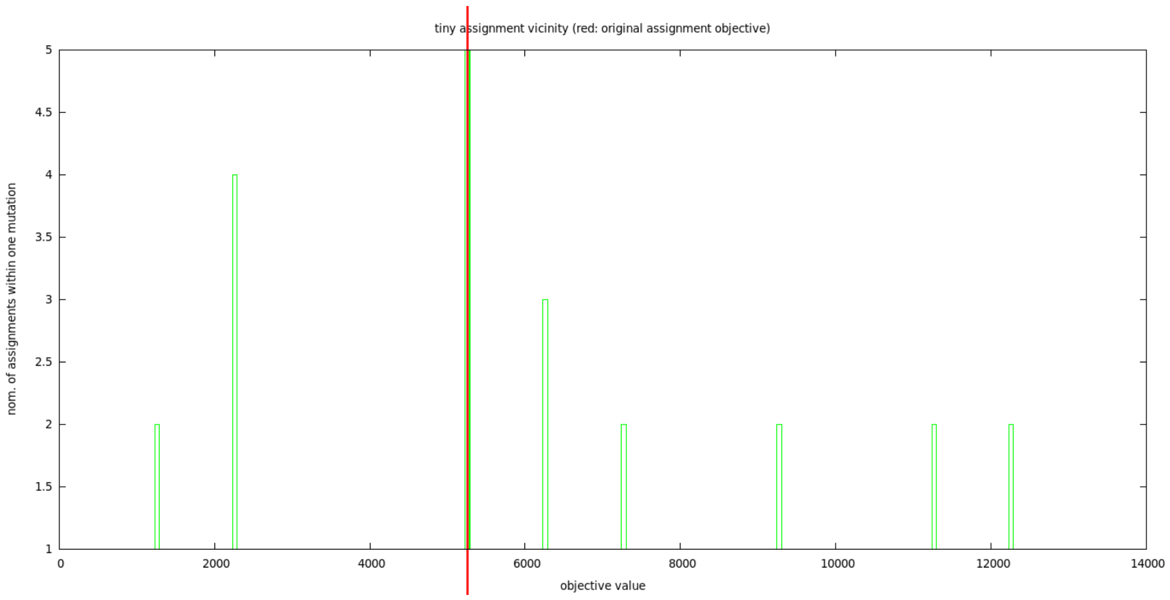


Figure 4.7: tiny instance with assignment of objective 5260; the red vertical line indicates the objective value of the chosen assignment.

Note that applying the smoothing operator with more rounds [9] does yield the same structure (see figure 4.9). In this case, the objective values are of course generally lower and due to the smoother being applied more times, the percentage of better solutions in the vicinity is lower.
The graphs shown above may be misleading as they only measure whether a neighboring solution is *better* but not *how much* better. If we only consider solutions to be better than the original one iff their objective value is better by at least 1000 points (i.e. a limit workload violation by one hour) in figure 4.10, the relationship changes dramatically. We can clearly see that the decline in available options decreased steeply. Therefore we can deduce that while better options become rare when approaching the optima, *palpably* better options become *very rare* indeed.

4.4 Performance of mutation based algorithms

This provides us with a means to estimate the performance of a mutation-based algorithm. For the above instance, we may assume a 10% probability for reaching a solution better by at least $1,000$ points for any assignment. From the smoothing analysis above we could assume starting out at a solution with objective value $6,000$; however suppose the start assignment has an objective value of about $13,000$. For the tiny instance, the vicinity of each assignment comprises about 35 other assignments. Suppose we select random solutions from the vicinity of the original solution. We

[9]the smoother defaults to 2 rounds

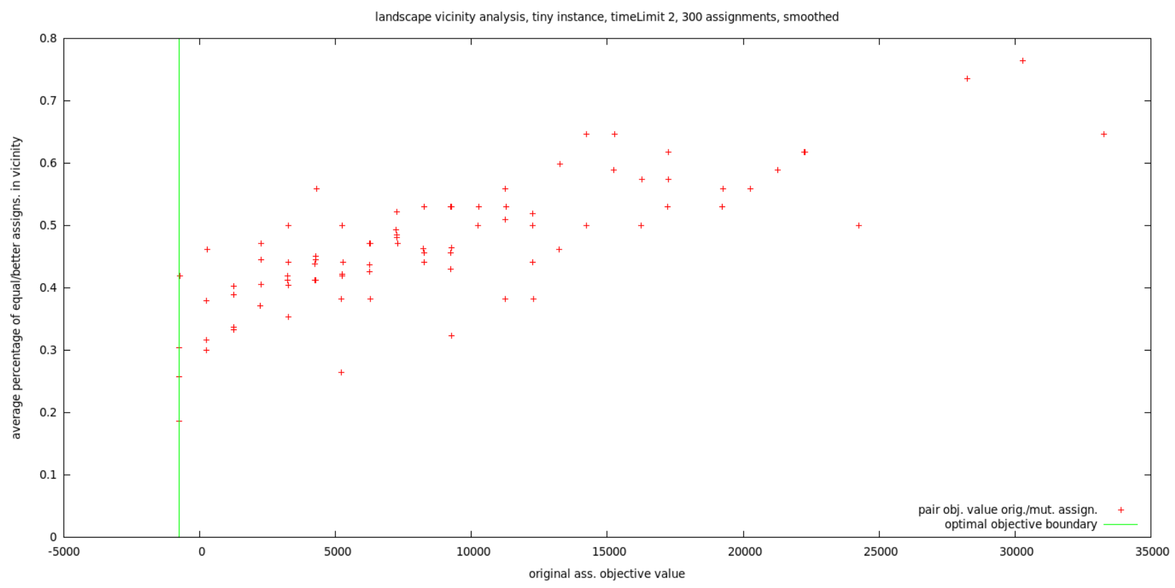


Figure 4.8: Percentage of solutions in immediate vicinity of 300 randomly drawn, smoothed solutions which are equal or better than the original solution. The time limit is set to 2 seconds. The percentages descend with ascending quality of the original solution.

assess all of the selected solutions and choose the best one; we want to select as many solutions such that the probability that the best solution is not an improvement by at least 1,000 points is less than 0.1. Trivial probability theory yields

$$0.9^n \leq 0.1, n \in \mathbb{N} \leftrightarrow n \geq 22$$

This new solution is selected as the new original solution and the algorithm enters its next cycle.
Therefore, probabilistically speaking, we need less than $22 \times 13 = 286$ trials. This calculation discards the fact that under mutation, “good” solutions do not necessarily lead to better solutions than “bad” solutions. The impact of this becomes clearer with the analysis in figure 4.11. The solver does only draw from the best $X\%$ solutions in the incumbent solution’s vicinity. Furthermore we ran three rounds for each probability.
According to figure 4.11, merely 5 rounds suffice to find a solution very close to the optimal solution. In each round, we evaluate all solutions in the vicinity, order them by the their objective value and cull all but the best $X\%$. From this remainder we choose the next incumbent solution randomly with a uniform distribution. While we calculate all the solutions for this algorithm, we could, for the sake of simulating a random selection in the first place, directly draw a mutation.
This decreases the number of solutions to assess by $\frac{22}{35}$. 5 rounds equal 5×35 evaluations. If we reverse the advantage we had by drawing from the best percentage,

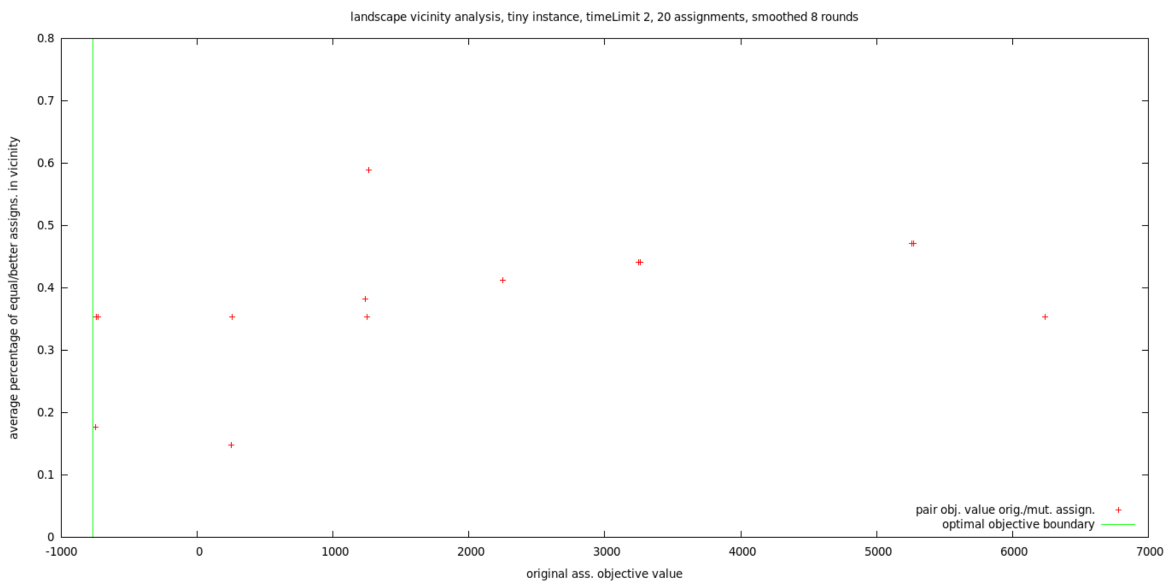


Figure 4.9: Percentage of better assignments in immediate vicinity; the vertical line indicates the absolute best objective value. The percentage descends with improving original assignment.

this yields us $5 \times 35 \times \frac{22}{35} = 110$ assessments. This is considerably less than the estimated 286 assessments estimated above; this trial backs up the upper bound. Furthermore we can be assured that the improvements are often much higher than by merely $1,000$.

Suppose the vicinity of an optimum resembles a singular, symmetric hillside. Then the angle of equal or better solutions within your immediate range is gradually increasing when moving towards the hill (we are assuming an ideal hill); when you are far away, it is negligible and might only allow for a move in no other but the direction of the hill. When approaching the hill, its left and right extensions are still equal or better than your position, therefore you could move straight on or you could deviate to the left or right.

The analysis above does not support such a notion of the vicinity of good solutions. In particular, we can see that the movements to improve the current solution become less and less when considering gradually improving solutions.

By choosing only from the best $X\%$, we might have cut off important solutions on the way. The relationship between the percentage of eligible solutions and the final solution quality is shown in figure 4.12.

There is clearly not much difference between choosing 0.1 and 0.2. For higher percentages, our naïve random choice of new incumbent solutions may distort the course; additional weighting might alleviate this steep increase.

Keeping graph 4.11 in mind it may appear tempting to choose 0.2 as the percentage of choice. However, 4.12 shows that there is no important difference to be found

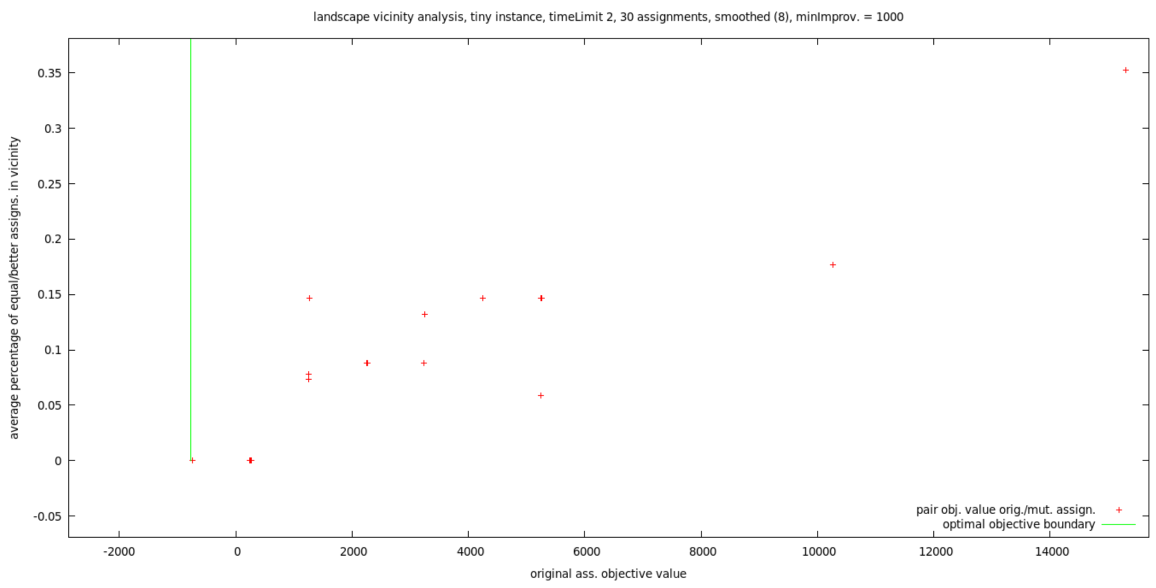


Figure 4.10: Percentage of vicinity solutions improving tangibly, i.e. by at least 1,000 points. The green vertical line indicates the optimal objective for the tiny instance.

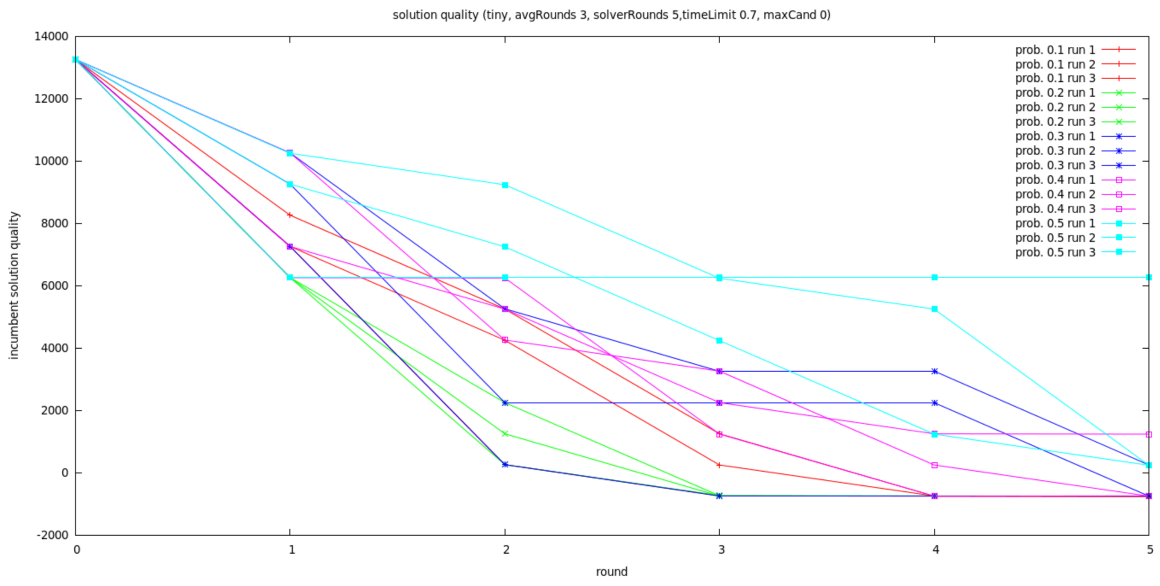


Figure 4.11: Course of objective value with MutationSolver

between both. Note that 4.11 does consider multiple assignments, whereas 4.12 does not. Furthermore choosing 0.1 as percentage does only incur half the load. Therefore, 0.1 is clearly a reasonable choice.

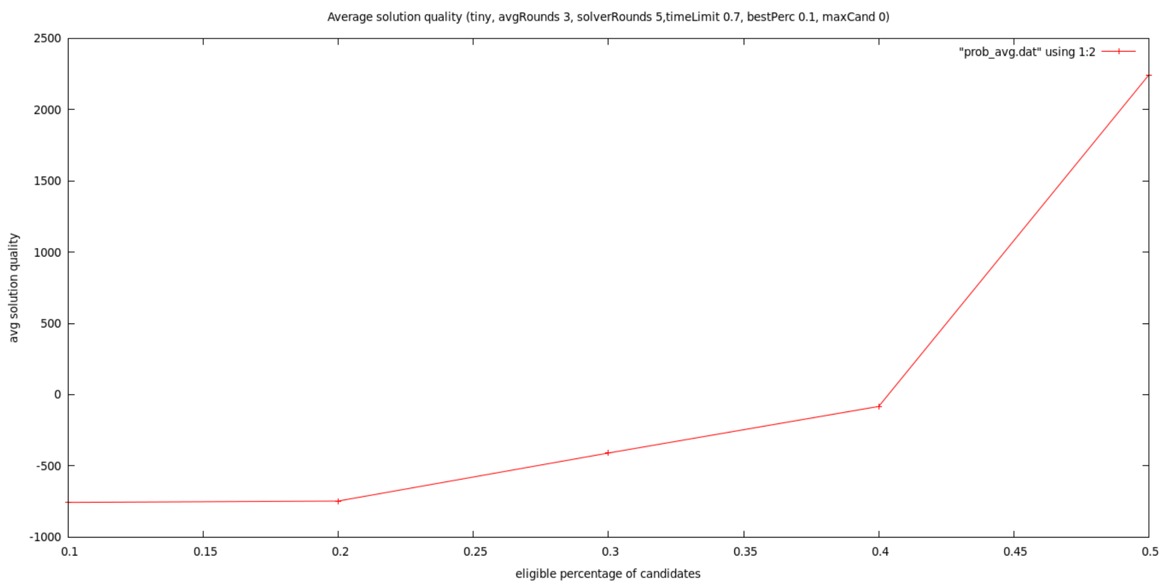


Figure 4.12: Solution quality dependency on chosen percentage

Furthermore we see that the fitness landscape is heavily rugged but scattered with small islands; consider the following excerpt from the very left of the distribution, as shown above in 4.5, in figure 4.13.
This, along with the global objective value distribution, justifies the steep decline in options we noticed in 4.10. Note that the analysis of the vicinity is *exhaustive*; this is only possible because we limited the analysis to a *local* analysis.

The graph 4.14 shows the algorithm, in the following referred to as *MutationSolver*, we developed above at work.
The *RandomSolver* is a reference solver. He merely draws random assignments and assesses them; if smoothing is enabled, he also smoothes them. It is hardly surprising that the random solver did exhibit very erratic behavior. However the above comparison averaged over 5 different randomly drawn resource assignments.
As we can clearly see, the random solver is *almost always* superior to the Mutation-Solver.
The courses have not been plotted; the time stated above reflects the time each solver needed to reach a solution within 80% of the optimal objective value. While the *MutationSolver* did exhibit a very steady course to the optimal solution, it was quite slow in doing so. This is mainly because we chose to draw so many solutions that we can be sure to improve by at least 1,000 points. [10]. We could reduce the number of assessed solutions to speed up the MutationSolver; however the graph above shows very clearly that we could just use the MutationSolver as well.

[10]The detailed explanation was given in the previous chapter

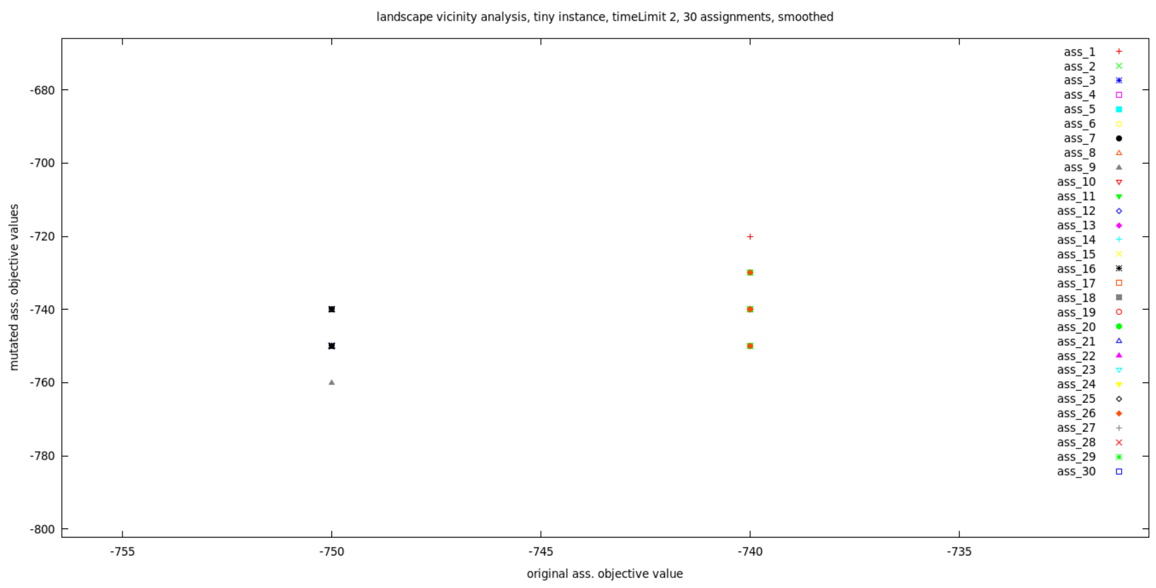


Figure 4.13: Excerpt of the close vicinity (with respect to objective values) of an assignment. Whereas the fitness landscape is heavily rugged on the global level, the smaller scale shows small islands. This is due to the weighting of the constraints at hand; the global scale is dominated by the LimitWorkloadConstraint relaxation.

Reducing the assessed assignments at that magnitude would nullify the probabilistic guarantees we have on the MutationSolver's behavior.
We deduce that while *steady* descend to near optimal solutions is possible within reasonable time [11], the RandomSolver is much faster and still has acceptable odds. While the time consumption of the RandomSolver was always higher than the ILP solver's consumption, the gap was small and did not rise very much when scaling up instances.

4.5 Correlation by Weinberger

Finally we want to analyze whether the second Weinberger conjecture (2.1) does apply to solving using solely the ILP solver.
Although we would prefer a linear correlation between the correlation coefficient and solution time (hardness), there is no other obvious relation either. Therefore this conjecture does not apply to this sub problem. The instance difficulty for the ILP solver using this model cannot be estimated using this method. As we have seen in the analysis above, particularly in the wake of the instance scaling analysis, unconstrained ILP solution time and preassigned solution time are correlated. Therefore

[11] for the analyzed instances

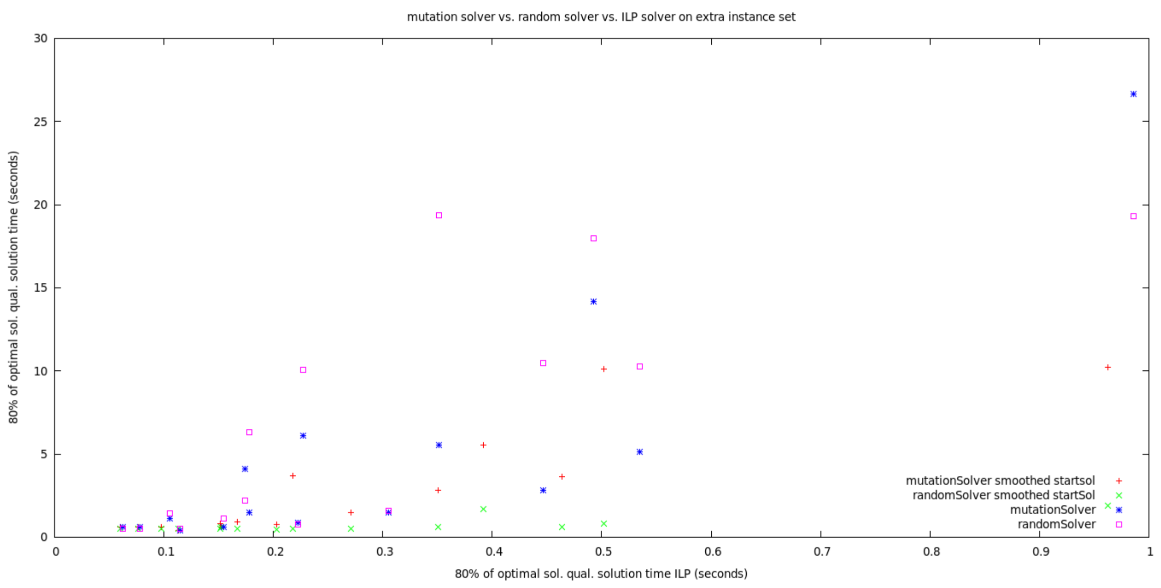


Figure 4.14: Comparing MutationSolver with RandomSolver and ILP solver; the instance set has been generated for this purpose. The MutationSolver is clearly inferior with respect to the averaged solution time.

we can deduce that this conjecture does neither apply to a bare ILP solver backed solution nor to our genetic multi stage approach.

4.6 Résumé

The first part of this thesis showed that ILP components cannot be assumed to behave “well”. Specifically the performance of essentially the same problem does heavily depend on the formulation at hand. Furthermore the ILP formulation may behave badly for instances where other solution techniques such as heuristics do not exhibit any peculiarities.

The second part revealed some unintuitive properties of the solution process and the fitness landscape. Insights into the behavior with respect to incumbent solutions, in particular the computation time for incumbent solutions and the objective value course can be leveraged in multi-stage approaches as the one at hand. The analysis of constraint impacts on computational complexity does yet again back up existing research. Furthermore it allows to dramatically (i.e. decreasing the magnitude of the steep rise when scaling up instances) cut down on computation times while retaining solution quality of the same time. Finally this thesis shed some light on the general complexity of TT fitness landscapes, in particular on the landscape of the teacher assignment sub-problem.

Those results are not only helpful to genetic approaches, but also to any number of

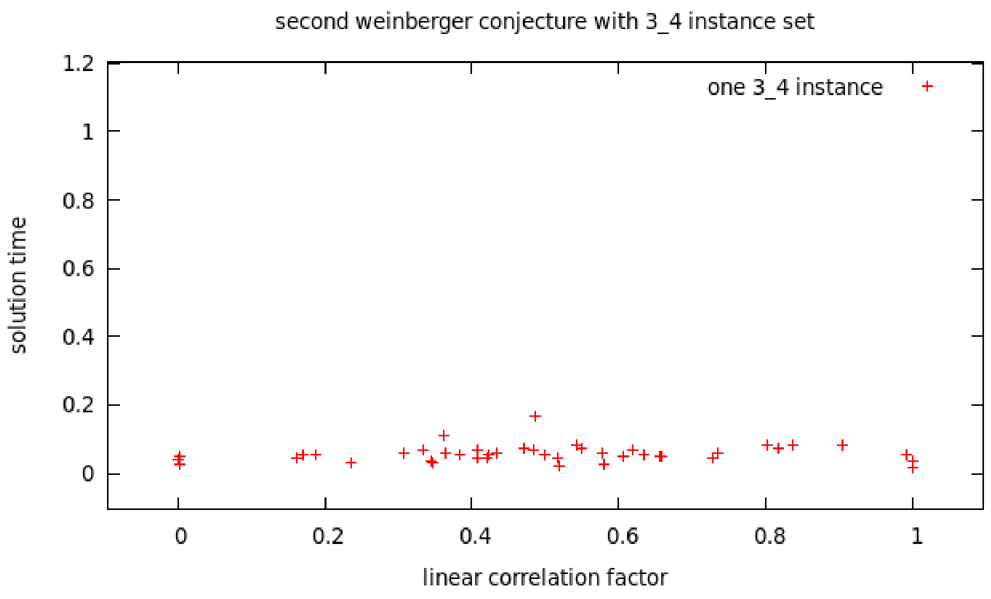


Figure 4.15: Weinberger Correlation for 3_4 instances; 80% solution quality as solution time

heuristics which specialize on TT problems.
Much groundwork has been done for this thesis - the developed tools can easily be adapted to suit other research needs. In addition to two genetic layers, this thesis provides:

- a C++ wrapper to the KHE library
- the first freely available ILP *implementation* of the XHSTT format
- a converter from the Roepstorff JSON format to the XHSTT format
- an instance generator for the Roepstorff format
- a host of instance landscape analysis tools for the Roepstorff format

4.7 Prospects

As opposed to the ILP formulation, the KHE solver shipped with the library did produce optimal results within an acceptable amount of time. Therefore, heuristics for the *second* stage (replacing the ILP formulation) might prove to be more successful for the XHSTT model. [12] This would then be considered to be the more

[12]The KHE library actually already ships such a heuristic algorithm. Furthermore, the heuristic solver may be decomposed into nearly arbitrarily small sub problems. For details, please refer to the KHE documentation; the API is already there.

genetic variant of *memetic algorithms*.

Furthermore another notion of movement on the solution space could be considered; our notion of movement did prove to yield rugged fitness landscapes and did not support the notion of singular hills. In particular the *recombination* operator might be worth exploring. Due to the delay with the first model in this thesis, this operator could not be researched.
The instance generator's properties still remain largely in the mist. Particularly the degree of resemblance of its output and real world instances is an interesting option to explore. There is still limited knowledge as to what distinguishes synthetic from real world instances, though real world instances have been observed to yield more cliques with respect to the graph coloring sub problem than synthetic generators.

Finally, algorithms interacting with the ILP solver might be worth exploring. Hooking into the solver's internals allows for fine-grained branch control during the solution process. This in turn allows to apply problem-specific knowledge to speed up the solver's incumbent solution discovery. As shown in the thesis, the incumbent solutions generally have a very short improvement course which is subsequently not improved by the solver. Sophisticated algorithms might use problem-specific knowledge to cut down on the later phase where no new incumbent solutions are found (e.g. by adding cut-off constraints) or yield to more incumbent solutions in the early solution process.

Bibliography

[ADSV07] Pasquale Avella, Bernardo D'Auria, Saverio Salerno, and Igor Vasil'ev. A computational study of local search algorithms for italian high-school timetabling. *J. Heuristics*, 13(6):543–556, 2007.

[AG08] K. M. Ng Aldy Gunawan, H. L. Ong. A genetic algorithm for the teacher assignment problem for a university in indonesia. 19 no. 1 p. 1-16, 2008.

[Akk72] E. A. Akkoyunlu. A linear algorithm for computing the optimum university timetable. 1972.

[AZ00] Eric Angel and Vassilis Zissimopoulos. On the classification of np-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99(1-3):261–277, 2000.

[BC98] Edmund K. Burke and Michael W. Carter, editors. *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT'97, Toronto, Canada, August 20-22, 1997, Selected Papers*, volume 1408 of *Lecture Notes in Computer Science*. Springer, 1998.

[BKN$^+$03] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In Fred Glover and GaryA. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 457–474. Springer US, 2003.

[BMKL08] Grigorios N. Beligiannis, Charalampos N. Moschopoulos, Georgios P. Kaperonis, and Spiridon D. Likothanassis. Applying evolutionary computation to the school timetabling problem: The greek case. *Computers & OR*, 35(4):1265–1280, 2008.

[CDB09] Patrick De Causmaecker, Peter Demeester, and Greet Vanden Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem. *European Journal of Operational Research*, 195(1):307–318, 2009.

[Che10] Chen. *Applied integer programming modeling and solution.* John Wiley and Sons, Hoboken, N.J, 2010.

[CK95] Tim B. Cooper and Jeffrey H. Kingston. The complexity of timetable construction problems. In *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, pages 283–295, 1995.

[CS03] Marco Chiarandini and Thomas Stützle. A landscape analysis for a hybrid approximate algorithm on a timetabling problem. TU Darmstadt Technical Report, AIDA-03-05, 2003.

[dBBB$^+$13] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[ED94] Weinberger ED. Local properties of kauffman's nk model: a tunably rugged energy landscape. Physical Review A 1991;44(10):6399–413, 1994.

[GJBP95] Christelle Guéret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, pages 130–145, 1995.

[GNP12] Aldy Gunawan, Kien Ming Ng, and Kim-Leng Poh. A hybridized lagrangian relaxation and simulated annealing method for the course timetabling problem. *Computers & OR*, 39(12):3074–3088, 2012.

[HHLS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 2009.

[HS07] Christopher Head and Sami Shaban. A heuristic approach to simultaneous course/student timetabling. *Computers & OR*, 34(4):919–933, 2007.

[HSLH14] Frank Hutter, Thomas Stützle, Kevin Leyton-Brown, and Holger H. Hoos. Paramils: An automatic algorithm configuration framework. *CoRR*, abs/1401.3492, 2014.

[IDw13] Nigel Stanton Ian David wilson, Ross Davies. A genetic algorithm based solution to the teaching assignmente problem. 81- No. 19, 2013.

[ITC] International timetabling competition. `http://www.utwente.nl/ctit/hstt/`. Accessed: 20.04.2015.

[JD12] Jörg Homberger Jonathan Domrös. An evolutionary algorithm for high school timetabling. Practice and Theory of Automated Timetabling (PATAT 2012):29–31, 2012.

[Lov10] April Lin Lovelace. On the complexity of scheduling university courses. 2010.

[LT15] R. Lewis and J. Thompson. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, 240(3):637–648, 2015.

[MSK12] Mohammed Saniee Abadeh Meysam Shahvali Kohshori. Hybrid genetic algorithms for university course timetabling. vol. 9 Issue 2 No. 2, 2012.

[MSP^{+}10] Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhydian Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

[PAD^{+}12] Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, and David Ranson. An XML format for benchmarks in high school timetabling. *Annals OR*, 194(1):385–397, 2012.

[Pas10] Vangelis Paschos. *Concepts of combinatorial optimization.* Wiley-ISTE, London, 2010.

[QA13] Danial Qaurooni and Mohammad R. Akbarzadeh-Totonchi. Course timetabling using evolutionary operators. *Appl. Soft Comput.*, 13(5):2504–2514, 2013.

[Rö13] Stefan Röpstorff. Efficient optimization of school timetables. 2013.

[Sie02] Gerard Sierksma. *Linear and integer programming : theory and practice.* Marcel Dekker, New York, 2002.

[SK13] thomas Riis Stidsen Simon Kristiansen. A comprehensive study of educational timeteabling, a survey. Report 8.22013, 2013.

[Sk14] Thomas R. Stidsen Simon kristiansen, Matias Sørensen. Integer programming for the generalized high school timetabling problem. Springer Science+Business Media, 2014.

[SL12] Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers and OR*, 39(5):875–889, 2012.

[SOS^{+}14] Jorge Alberto Soria-Alcaraz, Gabriela Ochoa, Jerry Swan, Juan Martín Carpio, Héctor Puga, and Edmund K. Burke. Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research*, 238(1):77–86, 2014.

[Suy10] Suyanto. An informed genetic algorithm for university course and student timetabling problems. In *Artifical Intelligence and Soft Computing, 10th International Conference, ICAISC 2010, Zakopane, Poland, June 13-17, 2010, Part II*, pages 229–236, 2010.

[TB12a] Ioannis X. Tassopoulos and Grigorios N. Beligiannis. A hybrid particle swarm optimization based algorithm for high school timetabling problems. *Appl. Soft Comput.*, 12(11):3472–3489, 2012.

[TB12b] Ioannis X. Tassopoulos and Grigorios N. Beligiannis. Solving effectively the school timetabling problem using particle swarm optimization. *Expert Syst. Appl.*, 39(5):6029–6040, 2012.

[THH97] Domingos M. Cardoso Tim H. Hultberg. The teacher assignment problem: A special case of the fixed charge transportation problem. European Journal of Operational Research(101):463–73, 1997.

[vdBH12] J. J. J. van den Broek and Cor A. J. Hurkens. An ip-based heuristic for the post enrolment course timetabling problem of the ITC2007. *Annals OR*, 194(1):439–454, 2012.

[Wan02] Yen-Zen Wang. An application of genetic algorithm methods for teacher assignment problems. *Expert Syst. Appl.*, 22(4):295–302, 2002.