10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A Criteria Transformation Approach to Timetabling based on Non-Linear Parameter Optimization

Christian John · Dietmar Tutsch · Reinhard Möller · Thomas Lepich · Bernard Beitz

Abstract This paper presents a concept for timetabling based on a parameter optimization system for approximative numerical calculation of some parameter combination under soft and hard constraints. The concept uses a non-linear parameter optimization method with an iterative variation of parameters. The paper focuses on the transformation process to migrate problem-domain specific criteria into optimization-compatible objects suitable for a standardized parameter optimization procedure. A framework for use in other problem domains is presented. The method is applicable to a wide range of timetabling problems due to its dynamically parametrized restrictions. Timetabling is integrated in educational context in various university and scholastic scopes.

Keywords Timetabling · Non-Linear Parameter · Optimization

1 Introduction

Timetabling is used in a vast variety of applications, such as in examination, curricula and courses, scheduling and assigning rooms or in general time management applications. This paper describes a new concept for deriving algorithmic criteria from generalized timetabling "stories", the set of fuzzily given conditions and constraints of a timetabling task, in order to calculate appropriate time slots under complex conditions. The general idea is to use a

Christian John
Bergische Universität Wuppertal
Faculty E Rainer-Gruenter-Strasse 21
D-42119 Wuppertal
Building FC, Room 2.07
Tel.: +49-202-4391186
Fax: +49-202-4391944
E-mail: chjohn@uni-wuppertal.de

probabilistic, iterative algorithm avoiding the well-known issues and problems of commonly used timetabling algorithms. Thus, the transformation of general problem-describing "criteria" to "parameters", "constraints", and "conditions" in terms of optimization is in our special focus. Today's timetabling solutions are mostly based on "syntactic", "statistic", or "structural" approaches. There are several disadvantages like preprocessing tasks, stochastic methods with deterministic tasks, fuzziness, clipping, exhaustive search, etc. [1–5]. Our concept is to avoid most of these disadvantages and to develop a more general and extensible solution for timetabling. The conceptual idea is based on a non-linear parameter optimization method, as considered by many authors before, consisting of an objective or target function, restrictions and constraints, and a set of parameters describing the problem domain [6–14].

In addition to existing approaches of this kind, our focus is on conveniently pre-processing the conditions and constraints.

2 Foundation

While typical combinatorial problems can easily be described in form of spoken or written text, their solution depends on the qualified extraction of the criteria governing the problem. Thinking of well-known optimization algorithms, we have to provide parameters, boundary conditions, constraints and an appropriate evaluator to be used as target or objective function. To glue these two sides together, we have to translate some informal criteria description into a syntactically correct formal expression with scalar parameters and explicit boundary conditions such as restrictions and constraints. These expressions can then be used to iteratively find a best-fitting solution - to achieve a goal, meaning to minimize or maximize the evaluation function based on some set of parameters while taking the boundary conditions into account [15,16].

Here the problem arises to transform problem-specific criteria descendent from some real-life problem domain into terms of optimization. Regarding the problem-specific criteria as issues or dimensions in an n-dimensional euclidean space, we have to find an isomorphic transformation of the form:

$$\Pi \approx \Omega$$

with Π being the domain-specific problem space and Ω being the optimization search space.

Fig. 1 shows this bijective transformation with the problem space criteria C_j and the search space parameters P_i and restrictions R_{j-i}.

This transformation mirrors the need for numerical solution of a non-linear system of equations. Several specialized approaches can be found to solve specific optimization problems (Fig. 2). The most general and non-limiting approach is to think of optimization in terms of *non-linear parameter optimization*.

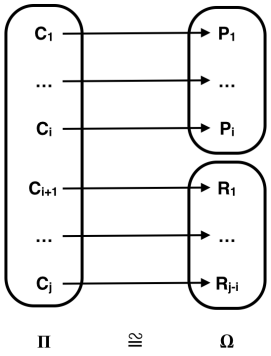

Fig. 1 Bijective criteria transformation

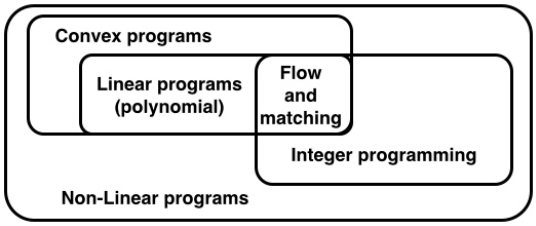

Fig. 2 Optimization problem classes (corresponding to [16])

Non-linear parameter optimization works by an iterative variation of parameters [16–19]. The variation may follow a stochastic and/or deterministic approach and normally comprises some sort of step-width control. While a parameter combination describes the problem domain of the specific given problem, the target function is expressed as *f(parameters)* and gives an estimation value for the grade of optimum fitting and thus the approach of the solution in question. Boundary conditions act as constraints that limit the valid parameter combinations. The optimum may be described as a minimum or maximum of the target function [20–22].

In general, parameter optimization in terms of a target function can be expressed as

$$X = \{x \in \Omega | \forall x' \in \Omega : f(x) \geq f(x') | f : \Omega \to \mathbb{R}, \geq \in \{<, >\}\}$$

with

Ω	the search space
$\geq \in \{<, >\}$	the comparative relation
$X \subseteq \Omega$	the set of global optima

where x is the n-dimensional set of scalar parameters with values fitting into the n-dimensional search space Ω and X is the best-fitting parameter set as the result of the optimization. We are free in the decision to choose the minimum or maximum approach for optimization, here we have chosen the minimum approach.

The iterative variation of parameters can be done in a stochastic or a deterministic manner or a mix of both. Deterministic parameter variation tries to find the best way towards the optimum in search by taking target function differentiations into account, while for stochastic parameter variation the need for a continuously differentiable target function does not arise. Typical representations for deterministic optimization are any forms of gradient procedures, whereas several stochastic procedures have been developed in the past, the evolutionary procedure being one of the most prominent ones.

The decision for deterministic or stochastic procedures in general or for some gradient procedure or evolutionary procedure in detail will have some impact on the overall performance of the optimization, but the main concepts are the same in both cases, so the decision for one of these algorhithms does not infringe the general timetabling procedure described below.

3 Timetabling as a Multi-Criteria Story

Encountering a real-life timetabling problem we are normally confronted with a diffuse informal description of conditions and goals to be met with the solution. We call this the "timetabling story". The first solution step is the need for semantically understanding the story details, rather than an algorithmic question. The meaning of understanding the timetabling story is to identify the events, limitations, desires and the time line. Then we transform these characteristic parts into formal definitions of prarameters, restrictions, constraints, the target function and the definition area. Some parts (see Fig. 3) have to be configured before the optimization is started.

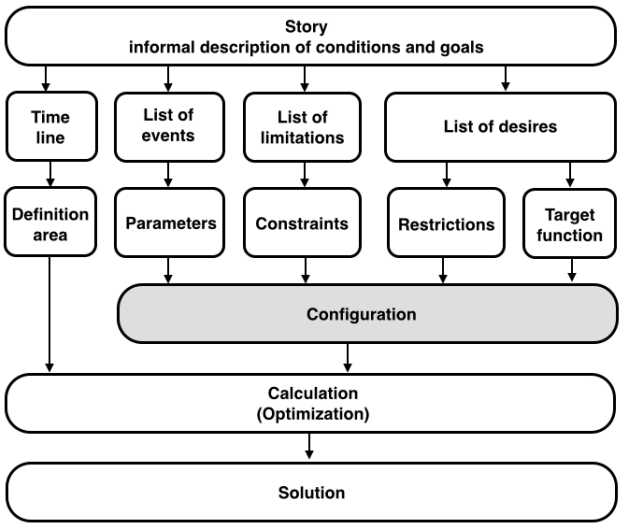

Fig. 3 Timetabling story transformation

Giving a definition of the term *timetabling*, we think of timetabling as some kind of time management for events, determining ideal time slot combinations or structured schedule based on a time line and restricted to a time interval. A timetable consists of distinct time slots, each having a start and end time, often denoted as "arrival" and "departure", and they need to be arranged or combined on the time line. The timeline is given as a one-dimensional definition region of time, restricted by a distinct start and end timestamp. Unfortunately the given time space is perforated by unavailable periods, e.g. weekends, holidays and night times. The remaining free time spaces then can be filled with the time slots.

While these aspects are concerning the timeline as a whole, more criteria have to be derived from the story, concerning inner relations and dependencies between time slots. Time slots may have a prescribed order, some or all time slots must not overlap, distance preferences may occur as well as daytime preferences.

Additionally, in almost all cases we have to ensure that all time slots find accommodation in the timeline because time slots must not be skipped.

In this paper, our goal is to create a generalized approach to all of these timetabling issues - or at least to a vast majority of them. Looking at the timetabling tasks in a more abstract way, we can name a number of general principles that can be found in all of the above-mentioned conditions:

- definition of time line, start and end time
- definition of blocked time spans
- recognition of free time spans
- observing relations, distances and dependencies
- observing allowed or forbidden overlapping
- observing orders and preferences.

These principles lead to the generalized criteria:

- time and duration
- relations and grouping
- delay and distance
- order and preferences
- overlapping

Other criteria may arise as well. Thus timetabling can be described as a multi-criteria problem. While some standard approach might think of criteria in terms of unilateral, bilateral and multilateral event relations, our solution handles these criteria classes (and any other criteria) in a unified manner. Even real-world knowledge may be integrated into this approach by defining customized optimization conditions.

Our generalized timetabling solution is based on a non-linear parameter optimization procedure (as described above), while the different criteria are implemented as sets of parameters and restrictions.

Fig. 4 shows the schematic user-interface for the "configuration" part in Fig. 3 where arbitary criteria may be added and parametrized by selecting predefined criteria classes from a given criteria catalogue and assigning appropiate values.

Constraints	Values		+	-
TimeSpanBlocking	12	18		
WeekEndBlocking	all			
HolidayBlocking	20			
NoCollision	all			
TimeSpanBlocking	23	43		
HolidayBlocking	50			

Fig. 4 Schematic User-Interface for Configuration and value assignment

Fig. 5 shows an example of a specific dynamically configured non-linear parameter optimization application with a combination of several different criteria principles, a target function, a set of restrictions and parameters. Thus the concept yields the following advantages: a dynamic selection of criteria, parametrizable constraint values, reusable restrictions, and enabling or disabling of conditions. As a result, we obtain a highly reconfigurable general solution system.

While this is the conceptual approach, the question arises, how to implement the named timetabling principles in a real-life dynamically configurable software application, allowing for adding any principle known and upcoming.

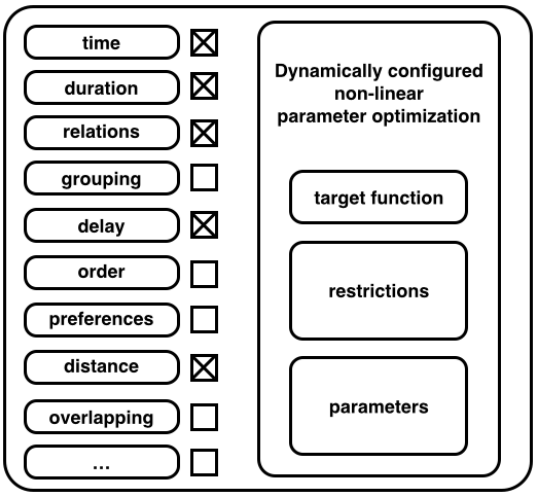

Fig. 5 Dynamically configured non-linear parameter optimization

4 Dynamic Multi-Criteria Transformation in OOP

After describing the conceptual structure of our problem domain the implementation starts with the technical realization in form of a three-layer OOP system (see Fig. 6).

In the top-down approach for analyzing the timetabling problem domain described above we have generalized typical timetabling questions, including event relations and real-world knowledge, into abstract criteria. The Object-Oriented Programming (OOP) design principle is the best-fitting paradigm to implement generalizable problem structures with concrete algorithm implementations. Following the standard OOP nomenclature, object classes are ruled by interfaces to encapsulate implementation details and realize inheritance and polymorphy.

We use OOP with classes and interfaces to implement the basic non-linear parameter optimization [16, 21] as our general problem solver and we provide interfaces to embed concrete parameters, restrictions, constraints, and target functions to meet the above-mentioned timetabling requirements.

Following this flexible and expandable design, we have structured our solution into a three-layer scheme, of which the first two layers contain the predefined parts of the software solution, while the third layer is the place for the implementation details of the particular problem (which, in our case, is the timetabling problem). This expandable layer has to be implemented individually with program code to match the problem's requirements.

4.1 First layer: Optimization

This is the basic layer forming the general foundation for our solution: the optimization layer. It generalizes the non-linear parameter optimization, defining the interfaces *IOptimizer* (for the optimizing algorithm as such) and a suitable *IVariator* (implementing the parameter variations and step-width control). The *IOptimizer* interface in particular is the hook for the real optimization algorithm's implementation. In general, we can differentiate between stochastic and deterministic optimization algorithms. In our implementation, a simple gradient program represents the deterministic algorithm class, where we decided to use a simple evolution program as representation of a stochastic algorithm. The variators have been developed accordingly. The optimizers in this basic layer may be used "as-is" by the concrete optimization task, but more sophisticated optimizers could be hooked in if estimated necessary, promising possible performance gains.

4.2 Second layer: Interface

The second layer is situated between the optimization and the real-life task. It defines the interfaces for linking the other layers together. The interface

layer provides several pre-conditions for the real-life task implementation to be fulfilled.

- *IParameter*: As described above, parameter optimization is equivalent to finding the best-fitting parameter combination according to a given evaluation or target function. The *IParameter* interface defines the abstract concepts for optimization parameters to be integrated into an optimization procedure.
- *IConstraint*: A constraint works as a boundary condition, hard-restricting parameter values to represent a valid solution by ensuring parameters to be contained in a valid range of values. Parameters violating a constraint must not be used in further calculations, they will be rejected and a new parameter variation will be triggered.
- *IRestriction*: A restriction is similar to a constraint, but it is a weak constraint: it is allowed to be violated, but violation will be punished with bad target values. Ideally, a restriction is implemented in a way such that violating the restriction will force the optimization converge against the optimum or at least against better values, expressed in parameter combinations. It will not enforce a new parameter set. Complying the restriction is a positive quality statement for the result - a valid and potentially better parameter combination.
- *ITarget*: This is the target function, evaluating and rating the current parameter combination. The target is the central place for defining the optimization goal. It is important to state that for deterministic algorithms the target function must be continuously differentiable, at least locally, i. e. it must be possible to calculate a numeric differentiation for the current parameter combination. Non-deterministic and stochastic algorithms do not rely on this precondition of differentiability.
- *ITask*: This is the overall container defining the concrete problem and binding the other parts together.

4.3 Third layer: Real-life task

This layer is the concrete implementation of all problem-specific parameters, constraints, restrictions as well as the target function (see Fig. 7). To build up a new problem we first define a Task and identify and implement the following details:

- The target function as implementation of *ITarget*
- Identify the problem's parameters and implement as set of *IParameter*, while initializing with estimated start parameter values
- Identify the constraints and implement as IConstraint subclasses
- Identify the restrictions and their penalty behavior, implemented as set of *IRestriction*

Now we can start the calculation. The optimization will begin with the start parameter set and iteratively vary the parameters and rate the current

result by evaluating the target function. The calculation will stop as soon as some termination condition is met. Using the evolutionary procedure [22], termination may simply be defined in terms of iteration steps, thus increasing the number of steps can give the chance for better results while deteriorating the runtime needed. In a more general approach, optimization procedures may have more sophisticated termination conditions in form of convergence criteria, promising more effective termination conditions. In this case, convergence has to be defined in the context of the problem domain. For deterministic optimization procedures, the target function's differentiability can help finding convergence criteria while running the risk of being stuck in some local optimum. For non-deterministic and stochastic procedures, the local optimum problem is reduced, but convergence criteria must be defined in some supplementary way.

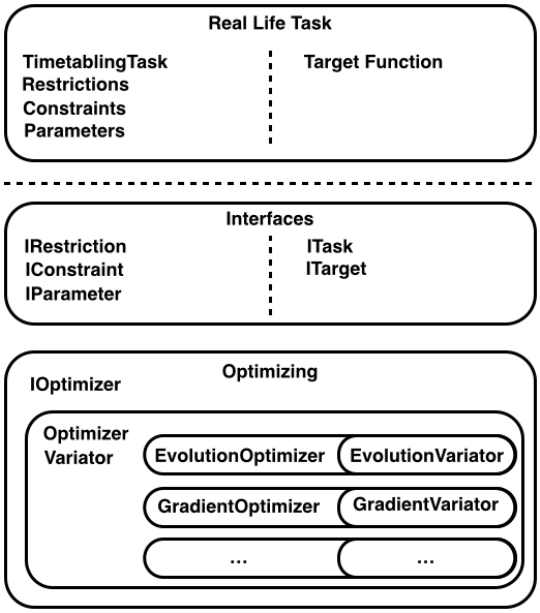

Fig. 6 Full three layer scheme

The whole three-layer scheme is outlined in Fig. 6, showing the OOP transformation of the dynamic multi-criteria timetabling algorithm.

4.4 Multi-Criteria Framework

We have seen that for solving a concrete task using this generalized OOP and optimization approach at least some programming is involved, implementing the scheme's interfaces appropriately. To facilitate this we have created a framework outlined below (see Fig. 7).

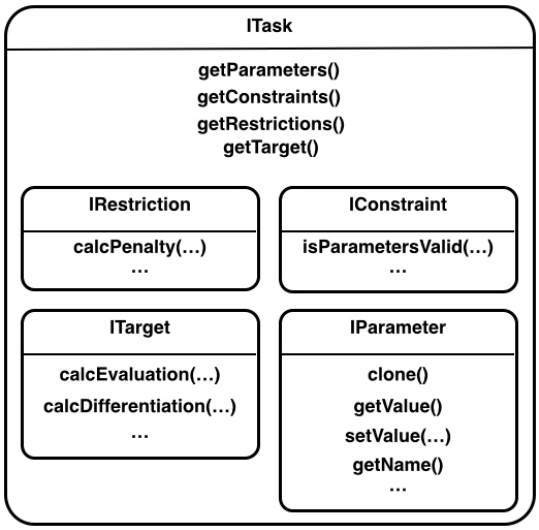

Fig. 7 Class Model

5 Scaling Effects in Transformation

Transforming the n-dimensional problem domain criteria into an optimization-based numerical solution, the problem of different parameter dimensions in terms of measure arises. In general, we can distinguish parameters in the dimensions

- time and date
- order and relation
- length and angle
- force, mass and weight
- color and brightness
- elasticity
- temperature
- and potentially many other dimensions.

Especially for timetabling, the time and order dimensions will become important. Both types of parameters have different definition ranges and cannot be handled equally without normalization. It is clear that those parameters must be submitted to normalization using their specific definition range. As a result, all normalized parameters may be treated equivalently in variation during optimization. We apply normalization in a strictly local manner, such that the *IParameters* internally encapsulate their true values while parameter variations apply normalized variation steps to them.

While *IConstraints*, if violated, decline our parameter set as a whole, *IRestrictions* provide a calculated penalty value representing the grade of non-compliance of the corresponding condition. A simple restriction may calculate its penalty value using a linear function. The linear function's slope can be

used as a weight factor for restrictions with different significance. More sophisticated penalty functions may be formulated as exponential functions.

As a result, we have to observe that

- parameters have to be normalized according their definition range
- restrictions have to be provided with an appropriate rating factor.

6 Example: Examination Planning at BUW

Paradigmatically, we show the conceptual approach to probabilistic timetabling for our institute at BUW Bergische Universität Wuppertal.

In the examination period we set our exams time slots, taking into consideration all constraints and relations between exams.

First we have to define a start-time constraint and an end-time constraint reflecting the examination period as a whole. Then we add more constraints:

- No exam should be written on a weekend - a *WeekendConstraint* arises
- Furthermore no exam should be written in an inconvenient time, for example in the night or too early in the morning - a *NightConstraint*
- Holidays are inconvenient as well as exam date - giving a *HolidayConstraint*

The constraints' goal is to find valid exam dates - our parameter set - in the resulting free time slots. Fig. 8 shows the constraints for the given timeline, limited to the constraints declared. Zooming into the timeline focusses a section of the timeline, in this example it is a typical week. Further zooming leads into the monday section. Obviously, applying all constraints gives a valid time span between 8:00 AM and 4:00 PM. In this idle time period on monday the algorithm might place five exams belonging to three different exam groups and thus partially overlapping. As we only consider the constraints at this point, the exam time slot distribution shown is valid, but not yet necessarily optimal. Further optimization steps will variate the time slot positions in the gaps of idle times, until eventually the optimum distribution is found.

7 OOP Transformation Framework

The OOP transformation implies the identification of the parameters describing this example task, the constraints and restrictions to be applied and the target function definition.

Running the optimization procedure, it will hopefully give a solution in form of the best-fitting parameter combination for our target function after n iterations. Here it would be a parameter set of t_i for the start of each time span. The number n of iterations needed to find the solution depends on several conditions: the arbitrary distribution of the exams start points, the kind of step-width control, the local-vs.-global optimum bias, and the selected fitting tolerance of the target function (the latter is equivalent to the algorithm's termination condition).

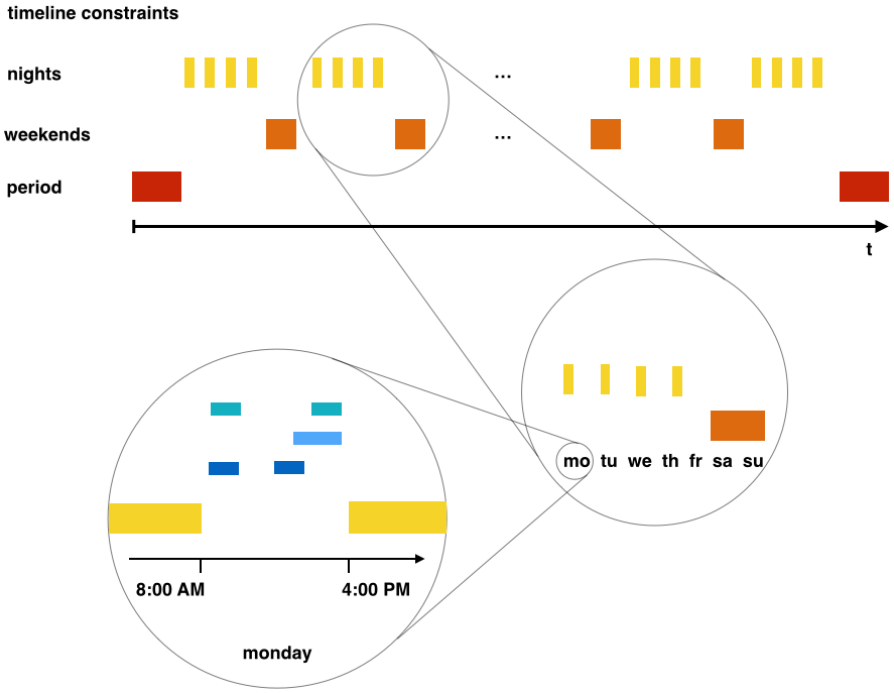

Fig. 8 Overwiew of the timeline constraints and free time spans

The number n of iterations can be estimated as being far lower than the steps we would need for an exhaustive search or any Monte-Carlo approach, but we can expect performance loss in comparison to conventional, direct timetabling algorithms. The performance loss will be compensated by the ability to dynamically parametrize and alter the set of constraints and restrictions.

8 Constraints and Restrictions

According to the previous examination timetabling example we limit the timeline by the period's start and end time defining these limits as two constraints: a *startConstraint* for the start or departure and an *endConstraint* for the end or arrival. The idea of constraints is to describe general conditions for a problem forcibly limiting the allowed parameter values. In this case, we have two boundary constraints because the time region has two limiting sides. E.g., the left side constraint checks the current parameter t against the region limit on the left side (see implementation: Listing 1). The result of these tests mark the actual given parameters to be valid or invalid. In case of invalid parameters the optimization procedure will trigger a new parameter variation, after which the parameters will again be submitted to constraint checks. In case of valid

parameters the target function will be evaluated using the current parameter set.

```
public class StartConstraint implements IConstraint
{
        public boolean isParametersValid(ParamSet params)
        {
                for (IParameter : params)
                        if (p.value < startTime)
                                return false;
                return true;
        }
}
```

Listing 1 Start constraint

While we understand constraints to be mandatory limitations for the problem's parameters, we describe restrictions to be weak constraints. Violating a restriction results in a bad target function value due to calculated penalty values from the restriction. In the context of our example the condition "two exams have to be a distance of seven days apart" would be a restriction (see implementation: Listing 2).

```
public class DistanceRestriction extends TimeRestriction
{
        private double dist;
        private int i1;
        private int i2;
        private double factor = 1.0;

        public DistanceRestriction(double factor, double dist,
                                   int i1, int i2)
        {
                super(factor);
                this.dist = dist;
                this.i1 = i1;
                this.i2 = i2;
        }

        public double calcPenalty(ParamSet params)
        {
                double t1 = params.get(i1).value;
                double t2 = params.get(i2).value;
                double delta = Math.abs(t1-t2);
                if (delta >= dist)
                        return 0.0;
                return factor * (dist - delta);
        }
}
```

Listing 2 Restriction for distance check

9 Target Function

Regarding the requirements described so far, our target function must fulfill two goals:

1. penalty values must lead back into the allowed time slots
2. when all restrictions are fulfilled, the time-span distribution must be submitted to a rating function, so that the optimization can converge towards the desired ideal distribution.

While goal (1) is handled by the restriction's penalty function values, fulfillment of goal (2) depends on programming the rating. The actual rating is formulated in terms of "distribute all time spans equally", "prefer gathering the time spans at the beginning of the period" or any other goal for the specific problem given.

While the start conditions

- set of start parameters
- start step width of the variation (i.e. search radius)

are crucial for a converging algorithm, the target function must reflect the optimization goal by rating the current parameter values according to the selected goal.

10 Dynamic configuration

The perception we have experienced by working with the timetabling is that a principal solution appears to be generalizable but specific timetabling tasks need specific ways of solution in detail. So we decided to provide a dynamically configurable solution process to represent the solution details.

Fig. 8 shows a configurable timetabling task and appropriate target function with optional settings for management of constraints and restrictions; we can turn on and off constraints and restrictions as needed, and in case of a missing condition, we can simply define a new constraint and/or restriction.

According to Fig. 9, adding a holiday would result in adding a new time-locking constraint for this day. As an example for a completely new condition we could introduce the idea of two time spans requiring to be in a defined order. In this case we implement a new *IRestriction* class and insert it into the timetabling task.

Applying the described procedure to real-life timetabling problems, short-term occurrences (e.g. teacher's illness) are omitted because they are not predictable.

11 Implementation

The software has been implemented in Java using the Eclipse RCP framework. The RCP framework and its plug-in mechanism allow to conveniently

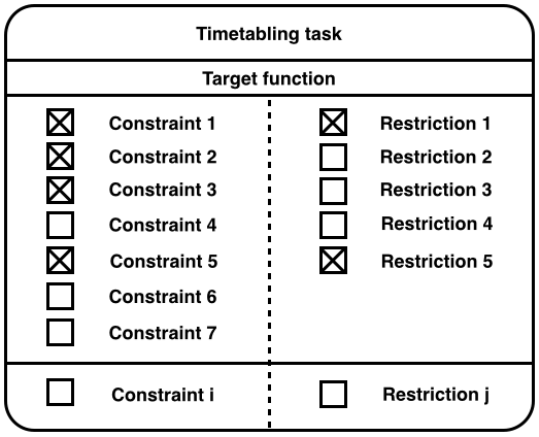

Fig. 9 Dynamic configuration of algorithm

expand the software for adapting to the current specific problem domain by implementing the basic framework interfaces. The core optimization algorithm is implemented as a basic plug-in (OSGi bundle) using the well-known evolutionary procedure. For performance enhancement purposes other optimization procedures may be implemented as well, no matter if following the stochastic or the deterministic approach. Any constraints, restrictions and target functions can be realized as additional plug-ins, thus making the whole package very flexible and expandable for other optimization domains.

12 Real-Life Applications

The main aspect for this work descended from the need for examination planning. Different kinds of exam constraints and exam restrictions lead to a non-linear parameter optimization approach. Before we built up this system, the planning was performed manually and with rather high error rate. The idea arose to create an automatic exam planning system and avoiding error rates.

We used the optimized exam dates for an extended application for room management. This includes a room list together with the room seat capacities and assigns exams to rooms, taking into consideration the expected numbers of participants.

After calculating the exam dates and appropriate rooms we provided this informations for another project: We developed an application for helping students and visitors navigating through and locating at the Wuppertal University Campus. Our students will use their smartphone and its integrated camera to take a picture from their current environment, and a server-based application will identify their position and locate them on the campus. Together with the timetabling information, students and exam candidates can be guided to their target exam rooms using smartphones and web browsers. This procedure is an extension of our eCampus project [eCampus] [23].

13 Conclusion

A new, flexible and configurable framework for transforming timetabling criteria stories was presented, using standardized condition classes for an iterative approach via non-linear parameter optimization. The concepts for transforming the different timetabling criteria into objects to be applied in an optimization procedure were shown and a simple framework for creating optimization-oriented subclasses and end-user applications has been presented. Scaling effects for parameters and restrictions have been discussed. The method uses constraints and restrictions to integrate arbitrary boundary conditions to represent timetabling specifics for a wide range of applications. An example demonstrated the basic ideas behind the transformation method and discussed some oncoming problems and their solutions.

References

1. Tomáš Müller, Real-life Examination Timetabling, in MISTA 2013 - Proceedings of the 6th Multidisciplinary International Scheduling Conference, 2013.
2. Tomáš Müller, H. Rudová, Real-life Curriculum-based Timetabling In PATAT 2012 - Proceedings of the 9th international conference on the Practice And Theory of Automated Timetabling, 2012.
3. Rui Li, Michael T.M. Emmerich et al.: Mixed Integer Evolution Strategies for Parameter Optimization. MIT 2013.
4. Alberto Colorni , Marco Dorigo , Vittorio Maniezzo, A Genetic Algorithm To Solve The Timetable Problem, Computational Optimization and Applications Journal, 2013
5. Tanguy Lapègue, Damien Prot, Odile Bellenguez-Morineau, A Tour Scheduling Problem with Fixed Jobs: use of Constraint Programming, Practice and Theory of Automated Timetabling, 2012.
6. Moritz Mühlenthaler, Rolf Wanka, Fairness in Academic Course Timetabling, Practice and Theory of Automated Timetabling, 2012.
7. Gilles Pesant, A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues, Timetabling, Practice and Theory of Automated Timetabling, 2012.
8. Benny Raphael, Ian F. C. Smith, Engineering Informatics: Fundamentals of Computer-Aided Engineering, Second Edition, John Wiley and Sons, 2013
9. Wil Michiels, Emile Aarts,Jan Korst, Theoretical Aspects of Local Search, Springer 2007
10. Francesca Rossi; Peter Van Beek; Toby Walsh, Handbook of constraint programming. Elsevier 2006
11. Burke E.K., Landa Silva J.D., Soubeiga E., Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, In Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference (MIC 2003), pp 129-158, 2005
12. Kalyanmoy Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley Paperback 2009
13. Kaisa Miettinen, Nonlinear Multiobjective Optimization, Springer 2012
14. Matthias Ehrgott, Multicriteria Optimization, Springer 2010
15. Christian John, Reinhard Möller, A Probabilistic Approach to Pattern-Matching Based on a Dynamic Rule-Driven System, 2013 IEEE GHTCE, Shenzhen 18.11.2013
16. Christos H. Papadimitriou, Kenneth Steiglitz, Combinatorial Optimization - Algorithms and Complexity. 1998 Dover, New Jersey
17. Jörn Schmidt, Christina Klüver, Jürgen Klüver: Programmierung naturanaloger Verfahren. Vieweg+Teubner. Wiesbaden 2010.
18. Juraj Hromkovic. Randomisierte Algorithmen. Teubner. Wiesbaden 2004.

19. Rolf Wanka. Approximationsalgorithmen. Teubner. Wiesbaden 2006.
20. Christian John, Thomas Lepich, Bernhard Beitz, Reinhard Möller, Dietmar Tutsch, A Probabilistic Approach to Pattern-Matching Based on Non-Linear Parameter Optimization, 2014 IEEE WCCAIS ICCIS, Sousse 17.02.2014
21. Walter Alt, Nichtlineare Optimierung, 2011, Vieweg+Teubner Verlag, Wiesbaden
22. Karsten Weicker: Evolutionre Algorithmen.Vieweg, Berlin 2007.
23. http://www.gds.uni-wuppertal.de/gds/service/veranstaltungen/projekt-icampus.html (2013-07-02,15:00)