University of British Columbia
Department of Computer Science
Technical Report TR-2009-15

An Automatically Configured Modular Algorithm for Post Enrollment Course Timetabling

Chris Fawcett
University of British Columbia
Computer Science Department
Vancouver, BC, Canada
fawcettc@cs.ubc.ca

Holger H. Hoos
University of British Columbia
Computer Science Department
Vancouver, BC, Canada
hoos@cs.ubc.ca

Marco Chiarandini
University of Southern Denmark
Department of Mathematics and Computer Science
Odense DK-5230, Denmark
marco@imada.sdu.dk

June 16, 2009

Abstract

Timetabling tasks form a widely studied type of resource scheduling problem, with important real-world applications in schools, universities and other educational settings. In this work, we focus on post-enrollment course timetabling, the problem that was covered by Track 2 of the recent 2nd International Timetabling Competition (ITC2007). Following an approach that makes strong use of automated exploration of a large design space of modular and highly parameterised stochastic local search algorithms for this problem, we produced a solver that placed third in Track 2 of ITC2007. In subsequent work, we further improved both the solver framework and the automated algorithm design procedure, and obtained a solver that achieves consistently better performance than the top-ranked solver from the competition and represents a substantial improvement in the state of the art for post-enrollment course timetabling.

Keywords: post enrollment course timetabling, stochastic local search, automated algorithm design, parameter tuning, algorithm configuration

1 Introduction

Course and examination timetabling is a resource constrained scheduling problem encountered by universities and other educational institutions. It involves scheduling a set of events (such as classes) into given rooms and timeslots. The schedule is subject to resource and feasibility constraints derived from the availability of rooms, student enrollments in the events, precedence relations between events, and student or teacher preferences. While the feasibility constraints must be strictly satisfied, giving rise to a satisfaction problem, preferences should not be violated whenever possible, which gives rise to an optimization problem. The presence of such hard and soft constraints is typical for many real-world constraint optimisation problems.

In this work, we present a new state-of-the-art solver for a particular timetabling problem, the Post-Enrollment Course Timetabling Problem (PECTP) [22], as considered in Track 2 of the recent 2nd International Timetabling Competition 2007 (ITC2007) and described in more detail in Section 2. Our contribution lies in the solver itself, which represents a substantial improvement over the previously best algorithm for the PECTP(the solver by Cambazard et al. that won Track 2 of ITC2007 [7] – see also Section 3), and in the approach used for designing this solver.

That approach is heavily based on a powerful automated algorithm configuration method, which allows us to search for a performance-optimised design within a large space of candidate solvers (see also Section 4). The same fundamental automated algorithm design approach has been recently used to obtain substantial improvements in the state of the art for solving a prominent abstract protein folding problem [27] as well as for solving various types of SAT instances [14, 18] (where the latter piece of work has been undertaken in parallel and to a large degree independently of the work presented here). The differences between those studies and the work presented here lies in some important details of the automated algorithm procedure used and in the way it was applied.

The space of algorithms for the PECTP considered in this work is defined by a modular solver framework that is based on stochastic local search (SLS) methods [11] and builds on several ideas from the timetabling and graph coloring literature. The key idea behind this framework is to first solve the feasibility problem and then the optimization problem by restricting the search to only feasible timetables. Different solution representations and neighbourhoods are interleaved during the search in the two phases, and randomization together with tabu search and simulated annealing concepts are combined in a flexible and novel way (details are provided in Section 5).

The search for performance-optimising instantiations of this modular and highly parameterised framework was performed using FocusedILS, a recent automated algorithm configuration procedure [16, 15] that can effectively deal with a large number of parameters. Based on the multi-phase architecture underlying our solver, we applied FocusedILS in four phases (described in Section 7) to first optimise the hard constraint solver (phases 1 and 2) and then the soft constraint solver (phases 3 and 4). For the last two phases, we used newly developed performance optimisation objectives, and in the last phase, we used a new version of FocusedILS, developed in the context of this work (described in Section 6).

The solver obtained at the end of phase 1 was submitted to Track 2 of ITC2007, where it ranked third.[1] The subsequent development phases resulted in additional, substantial performance improvements, and the final version of our solver obtained at the end of phase 4 finds consistently feasible timetables for all 24 PECTP instances from ITC2007 within 600 CPU seconds each and outperforms the previous state of the art solver by Cambazard et al. on 20 of the 24 instances in terms of soft constraint violations (see Section 8).

2 The Post-Enrolment Timetabling Problem

The following formalisation was defined in the context of the 2nd International Timetabling Competition (ITC2007)[2] and is described in detail in [22]; it is an extension of the problem used in the first International Timetabling Competition (ITC2003) and defined by Ben Paechter.[3] This model of university course timetabling involves scheduling a set of weekly recurring classes into timeslots and rooms. There are 45 non-overlapping timeslots, nine on each of five weekdays and a complement of rooms, each of which has a certain student capacity as well as specific "features", such as a projector or dry-erase boards. Finally, a set of classes is given, each of which involves a set of students, who may attend multiple classes during the week; each class needs to be scheduled in a room of sufficient capacity, but in addition may require specific room features. Additionally, each class can only be scheduled into a given subset of "available" timeslots,

[1] This part of the our work was outlined in an extended abstract that appeared in the proceedings of PATAT'08 [9].

[2] Second International Timetabling Competition, 2007. `http://www.cs.qub.ac.uk/itc2007/index.htm` Visited April 2009.

[3] First International Timetabling Competition, 2003. `http://www.idsia.ch/Files/ttcomp2002/`. Visited April 2009.

and precedences between classes of the form "class A must be scheduled in an earlier timeslot than class B" may be specified.

A *timetable* associates each of a number of classes with a timeslot and a room; each class that is associated in this way is said to be *scheduled* to the respective timeslot and room. A *complete timetable* is one that schedules all given classes, while a *partial timetable* may leave some classes unscheduled. A partial timetable is *valid*, if the following *hard constraints* are satisfied by all scheduled classes:

1. No student has to attend more than one class at the same time.
2. Only one class is assigned for any given room and timeslot.
3. All classes are assigned to suitable rooms, that is, rooms that meet the capacity and feature requirements of the class.
4. No class is assigned to a timeslot for which it is declared not available.
5. No class violates the pre-defined lecture precedences in the week.

A timetable is said to be *feasible* if it is complete and if it does not violate any hard constraint.

Valid and feasible timetables should also be fair for the students by meeting the following *soft constraints*:

1. Students should not have to attend only one lecture during any given day.
2. Students should not have to attend three or more classes in consecutive timeslots in a day.
3. Students should not have to attend a class in the last timeslot of a day.

Given a specific instance of the PECTP, the objective is to find a timetable that is feasible and that minimizes the occurrence of soft constraint violations. The quality of a valid timetable is measured by the *distance to feasibility* (DTF) and by *the cost of soft constraint violations* (CSCV). The DTF is defined as the total number of students that attend unscheduled classes. The CSCV is given by the sum of (i) the number of occurrences of a student having just one class on a day, (ii) a score for each occurrence of a student having more than two classes consecutively (the score is determined by the number of consecutive classes exceeding two), and (iii) the number of occurrences of a student having a class in the last timeslot of a day. The minimization of the DTF has higher priority than the minimization of CSCV. According to the rules of ITC2007, different timetables are ranked according to DTF, with ties broken according to CSCV. In practice, it is possible to associate with any given timetable a value $X := W \cdot DTF + CSCV$, where W is at least as large as the largest possible value for $CSCV$. Then a ranking of k timetables is obtained by sorting the associated values $X^{(1)} < X^{(2)} < \ldots < X^{(k)}$ and assigning rank values between 1 and k to the corresponding timetables (if there is a tie, rank values are averaged).

The PECTPas defined here and in ITC2007 differs from the earlier formalisation used in ITC2003 only by the newly added hard constraints 4 and 5. The instances used in both competitions have been synthetically generated by an automated instance generator developed by Ben Paechter and it is known that each instance admits an optimal solution of zero cost. Given the instance structure and considering the two additional constraints in ITC2007, the instances of the second competition are generally perceived to be much harder.

3 Related Work

At ITC2003, the best results were achieved by Kostuch et al. [19] with a three phase heuristic approach consisting of a graph coloring based construction of a valid timetable, a local search improvement phase

to reach feasibility and a simulated annealing phase for minimizing the soft constraint violations. Many others among the best solution approaches used heuristic algorithms and similar ideas (see `http://www.idsia.ch/Files/ttcomp2002/results.htm` for details).

At ITC2007 (Track 2), the solver that ranked first is a local search approach by Cambazard et al. [7]. This solver has many elements similar to the approach described in [8]. A matrix representation is used to define tentative timetables. The position in the matrix of an event determines the timeslot and the room uniquely. The solver starts with a randomly generated timetable and tries to make it first feasible and then better through the application of a combination of five neighbourhood operators. The neighborhood operators are then embedded in stochastic local search methods with plateau investigation, tabu search and simulated annealing strategies.

Ibaraki [17] encodes the PECTPinto an optimization version of the constraints satisfaction problem. He then uses a general-purpose algorithm based on an efficient tabu search to solve the model. The solver uses binary variables indicating the assignment of a value to a variable and basic one-exchange and swap neighborhoods. The method ranked second in the 2007 competition in spite of its generality and its apparent simplicity.

Müller [25] developed a Constraint Solver Library based on local search techniques which has applicability to several timetabling problems. This solver, much the same as other approaches, treats hard and soft constraints in two different phases. However, differently from [7] the hard constraint phase is solved working with valid partial timetables while soft constraints are solved later on a complete timetable. The room and precedence constraints are relaxed and treated as soft constraints. The solver is a combination of stochastic local search methods such as iterative improvement, tabu search and simulated annealing (although named differently). A set of trivial neighborhoods are used with equal probability at each iteration.

Another competitive approach at ITC2007 is due to Mayer et al. [23] who developed an ant colony method substantially different from the implementation proposed in the context of ITC2003 by Socha et al. [26] and previous versions. The peculiarity is the use of two pheromone matrices, one for the desirability of lecture-timeslot assignments and another for the desirability of lecture-room assignments. The construction phase considers classes in random order and for each of them timeslots and rooms are scanned in the order indicated by the pheromone values. The first timeslot and room that maintain the timetable valid are assigned. Once a feasible timeslot has been found, an improvement phase that reschedules classes causing large numbers of violations into new timeslots takes care of the soft constraints.

All the methods that ranked in the first positions in ITC2007, as well as in ITC2003, belong to the family of stochastic local search algorithms. A complete constraint programming approach is shown, most recently by Cambazard et al. [7] to be still not competitive. Integer programming approaches are instead more promising and Lach and Lübbecke [21, 20] report very good results on a different but easier timetabling problem.

4 Automated Algorithm Configuration

In recent years there has been a considerable amount of methodological research devoted to the issue of configuring the components and tuning the parameters of optimization algorithms, and especially of heuristic algorithms. Contrary to the traditional approach of trying to minimise the number of user-configurable algorithm parameters, these methods embrace the idea of parameterising as much algorithm functionality as possible [12].

Previous work in the area of automated algorithm configuration includes CALIBRA [1], the F-Race algorithm (and variants) [5, 3], model-based approaches such as sequential kriging optimisation (SKO) [13] and sequential parameter optimisation (SPO) [4], and a recent framework based on stochastic local search techniques, FocusedILS [16, 15].

FocusedILS [16] is a tool that performs local search in the space of parameter configurations for a given algorithm in order to achieve optimal (or near-optimal) performance on a set of problem instances. It currently supports parameters with discrete, finite domains; these parameters can be either numerical or categorical, where categorical parameters are used primarily for choices between alternative options or components, such as search heuristics. FocusedILS also supports nested parameters and can hence deal with situations in which the choice of one component introduces additional parameters of this component into the configuration process, as happens often in the design of complex stochastic local search algorithms. The algorithm to be configured is considered to be a black box procedure, and FocusedILS does not need any specific information about how it operates or about the problem it is solving.

FocusedILS has been successfully applied previously to automatically tune the parameters of SPEAR, a high performance DPLL-style SAT solver for formal verification problems, achieving state-of-the-art performance on that problem [14]. Concurrently with the work presented in this paper, FocusedILS has also been used to automatically configure SATenstein, a high-performance stochastic local search algorithm for SAT, by choosing among many modular components in order to instantiate a general algorithm framework [18]. This is the only current work of this type that approaches the flexibility and modularity of the approach presented in this paper.

We decided to use FocusedILS in this work because it is the only automated algorithm configuration procedure we are aware of that has been demonstrated to be effective in dealing with very large, highly discrete design spaces.

5 Solver Design

Our solver consists of two main procedures: a hard constraint solver and a soft constraint violation minimizer. All parameters exposed by this version of our solver are documented in Table 1.

5.1 Hard Constraint Solver

The *hard constraint solver* consists of a constructive phase followed by a stochastic local search phase. The constructive phase generates feasible partial assignments of classes to time slots and rooms by using an approach similar to that of [2]. The heuristic assigns classes to time slots in such a way that the five hard constraints are satisfied. Specifically, first a topological order of the digraph representing the precedence constraints is constructed and the classes are sorted accordingly with ties broken randomly. The classes are then considered in that sorted order and each class is placed into a time slot that is both feasible and compatible with the fewest classes that are left to be scheduled. The feasibility of the insertion of a lecture into a time slot with respect to room assignment is checked by computing the exact matching of the classes and rooms scheduled so far in that time slot. Lectures without a feasible insertion point are left in a list of unscheduled classes for possible insertion in the local search phase.

The construction is repeated a number of times based on the parameter *numConstructionSolutions*, taking advantage of the existence of multiple topological orderings to vary the insertion order. For easy problem instances, this procedure is often sufficient to obtain a complete feasible assignment. In general, however, we may be left with some unassigned classes. We choose the constructed timetable with lowest DTF value and improve it in a second search phase, which uses a tabu search procedure based on the PARTIALCOL algorithm [6]. At each iteration of this procedure, an unscheduled lecture is inserted into the best non-tabu time slot for it, and all classes breaking any hard constraint as a result of the insertion are moved into a list of currently unscheduled classes. The selection of the lecture to be inserted at each step is guided by evaluating the number of students attending each unscheduled lecture, which is the contribution of that lecture to the distance-to-feasibility measure. After a number of non-improving iterations equal to the number of events in the problem multiplied by *hardTSIdleItersMultiplier*, the best partial feasible solution is passed to the

first-improvement local search of the soft constraint solver (see Section 5.2). This procedure perturbs the current timetable in a way that is favourable to the hard constraint solver.

If after *maxNonImprovingIterations* rounds of this alternation between tabu search and soft constraint optimisation an improving move has still not been made, a number of classes are chosen at random and removed from the timetable into the list of unscheduled classes. The number to remove begins at *howManyIncrement* and is incremented by *howManyIncrement* each time the procedure is required, reduced modulo *howMany-Modulus* multiplied by the number of events in the problem. This ensures that the number of events removed varies over time and does not grow without bound. The parameter *resetHowManyAfterImprovement* dictates whether or not the number to remove is reset to the beginning value of *howManyIncrement* after an improving move is finally found.

We continue alternating tabu search on the unscheduled classes with the two levels of solution perturbation until a feasible solution is found or a provided time limit is reached. On instances for which feasibility is reached very quickly, we additionally repeat the entire procedure *numConstructionSolutions* times before selecting the feasible assignment with minimum soft constraint violations.

5.2 Soft Constraint Solver

The *soft constraint violation minimizer* is applied to the assignment returned by the hard constraint solver. The solver starts first with a first-improvement local search until a local minima is found for all of the 1-exchange, 2-exchange, swap-of-time-slots and Kempe chain neighborhoods. These neighbourhoods were previously described in [8], and are applied here in the same order as in that work. Next, a modified version of the simulated annealing solver presented in [8] is applied until the time limit is reached or a schedule with zero soft constraint violations is located. This solver has been essentially reimplemented since that work, primarily to be more modular, and to support additional parameters. In order to determine the initial simulated annealing temperature, *saSamplesForInitialTemperature* random moves are performed in the 2-exchange neighbourhood. The initial temperature is then set to the average change in soft constraint violation from the random moves multiplied by *saIniTempFactor*.

At each step of the search, and until the time limit is reached, we attempt to find an acceptable move in either the 2-exchange or swap-of-time-slots neighbourhoods, with the choice determined by *saSTSSelectionProbability*. Moves in these neighbourhoods are acceptable if they improve CSCV or with probability $p = e^{\frac{\Delta}{T}}$ if they are non-improving, where Δ is the change in CSCV produced by the move and T is the current temperature. Each of the two neighbourhoods can exit early before attempting every possible move, after a percentage of the total moves determined by *stsIdleCountForEarlyExit* or *twoexIdleCountForEarlyExit* are not accepted. This allows the search to avoid wasting time and adjust the temperature as needed earlier than it would have otherwise.

After the neighbourhood terminates, having found an acceptable move or not, we move to a temperature adjustment phase. T is cooled to $T * saAlpha$ after *saIterationCutoffCooling* attempts have been made to make a move at the current temperature, regardless of whether there was an improvement of CSCV. If *saStagnationStrategy* is not *no_reheating*, then after *saIdleIterationCutoffReheating* rounds of temperature changes with no improvement in CSCV we heat T to $T + T * saGeometricBeta$ or $T + T_0 * saAdditiveBeta$ depending on the value of *saStagnationStrategy*, where T_0 is the initial temperature.

6 Extensions to FocusedILS

During the development of our solver, we implemented two extensions to FocusedILS. [4] The first of these provides support for parameters whose impact on the performance of the algorithm that is being optimised

[4] This new version 2.4 of FocusedILS is available from the project webpage, http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/

Parameter Name	Type	Default	Domain Values
numConstructionSolutions	Hard	10	{1, 5, **10**, 25, 50, 100}
hardTSTabuDelta	Hard	2.0	{1.0, 1.2, **1.5**, 2.0, 2.5, 3.0}
maxNonImprovingIterations	Hard	10000	{1, **10**, 50, 100, 500, 1000, 5000, 10000, 100000, 1000000}
hardTSIdleItersMultiplier	Hard	50	{1, 2, 5, **10**, 25, 50, 75, 100}
resetHowManyAfterImprovement	Hard	disabled	{enabled, **disabled**}
howManyIncrement	Hard	2	{1, 2, 3, 4, 5, **6**, 7, 10, 15, 20}
howManyModulus	Hard	0.2	{0.01, 0.02, 0.05, 0.1, **0.2**, 0.3, 0.4, 0.5}
saAlpha	Soft	0.99	{0.90, **0.91**, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.0}
saIniTempFactor	Soft	0.40	{0.20, 0.40, 0.60, 0.80, 1.00, 1.20, 1.40, 1.60, 1.80, 2.00, 2.20, 2.40, 2.60, 3.00, 3.25, **3.50**, 3.75, 4.00, 4.25, 4.50}
saSamplesForInitialTemperature	Soft	100	{1, 10, 50, **100**, 150, 200}
saIterationCutoffCooling	Soft	50	{1, 10, 50, 100, 1000, 3000, 5000, 7000, 9000, 11000, 13000, 15000, 17000, 19000, 21000, 23000, 25000, 27000, 29000, 31000, 33000, 35000, **37000**, 39000, 41000, 43000, 45000}
saIdleIterationCutoffReheating	Soft	25	{1, 2, 5, 10, 15, 20, 25, 30, 35, 40, 45, **50**, 55, 60, 65, 70, 75}
saStagnationStrategy	Soft	reheat_additive	{**reheat_additive**, reheat_geometric, no_reheating}
saAdditiveBeta	Soft	0.50	{0.1, 0.25, 0.50, 0.75, 1.0, 1.25, **1.50**, 1.75, 2.00, 2.25}
saGeometricBeta	Soft	N/A	{0.01, 0.05, 0.10, 0.20, 0.50, 0.70, 0.90, 1.00}
saSTSSelectionProbability	Soft	0.1	{0.0, **0.1**, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
stsIdleCountForEarlyExit	Soft	1.0	{0.001, 0.005, **0.01**, 0.05, 0.10, 0.25, 0.50, 0.75, 1.0}
twoexIdleCountForEarlyExit	Soft	1.0	{0.001, 0.005, 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, **1.0**}

Table 1: The eighteen exposed parameters in the current version of our solver. "Hard" indicates that the parameter is used in the hard constraint solver, while "Soft" indicates that the parameter was used in the soft constraint solver. All parameters other than *resetHowManyAfterImprovement* and *saStagnationStrategy* were marked as unimodal when tuning with FocusedILS. The parameter values of the final tuned parameter configuration are indicated in bold.

is characterised by a unimodal function, while the second supports a new performance metric that is useful when attempting to match or beat the performance of an existing solver.

6.1 Support for Unimodal Parameter Response Curves

A response curve for a numeric algorithm parameter is a function that maps the values of that parameter to the performance of the algorithm when all other parameter values remain fixed. Since we found the response curves observed for many numerical parameters encountered in our algorithm design approach to be unimodal, we extended FocusedILS to support parameters explicitly marked to have unimodal response curves. For such parameters, local search steps in FocusedILS are restricted to only consider values that are directly adjacent to the current value. We note that using this method does not change the theoretical guarantee that FocusedILS, when run sufficiently long, will always find an optimal parameter configuration [16], even if it is applied to parameters whose response curves are *not* unimodal. However, when used inappropriately, it may slow down FocusedILS's progress towards better configurations.

6.2 The p-Value Performance Metric

In order to evaluate the performance of an algorithm (the *challenger*, C) against another stochastic algorithm (the *incumbent*, I), we introduce the *p-value performance metric*, ppm. Let $q_{I,1}, q_{I,2}, \ldots q_{I,n}$ be a sample of solution qualities found by I in a set of independent runs on a given instance, and let $q_{I,max}$ denote the maximum value of this sample. With every result q_C obtained by a single run of C we associate a value ppm defined as follows. If $q_C > q_{I,max}$ we set $ppm := q_C/q_{I,max}$, if $q_{I,max} = 0$ we set $ppm := q$, otherwise we set ppm equal to the p-value of q_C in the empirical solution quality distribution (SQD) given by the sample $q_{I,1}, q_{I,2}, \ldots q_{I,n}$. This definition is designed to provide search-based algorithm design approaches

like FocusedILS with guidance towards finding better challengers. When evaluating multiple runs of C on multiple instances (for each of which we have an SQD of I), ppm is calculated as the average over the ppm values for the individual runs and instances. It is this average ppm value over runs and instances that we use to evaluate a point in the search space of FocusedILS.

7 Experimental Setup and Protocol

All of our automated configuration runs using FocusedILS, as well as all evaluations of our solvers and those of other finalists were performed on a cluster of machines, each running Suse Linux 10.1, kernel 2.6.16.27-0.9-smp, with 2GB of RAM and a dual-core Intel Xeon 3.20Ghz CPU with 2MB of L2 cache per core. Each independent run of FocusedILS used only one machine and processor core, and the cluster was used to perform these independent runs in parallel. The process of design and configuration for our solver was done in four phases, each described in the subsections below.

7.1 Original Submission to ITC2007

The entire iterative process of designing and testing the solver used for our submission to ITC2007 took approximately one month, during which a total of 1000 CPU hours were used for FocusedILS runs. The hard constraint solver and its parameters were at that point identical to the version described in this paper, although the parameters had reduced domains compared to those given in Table 1. Although there were four exposed parameters for the soft constraint solver in this phase, none of them were modified from their values in the submission of Chiarandini et al. to ITC2003. In this phase we used FocusedILS 2.2 to configure the seven hard constraint solver parameters to minimise mean CSCV over 384 CPU second runs on all of sixteen ITC2007 instances available during the competition (the 'public' instances).

7.2 Optimising the Hard Constraint Solver for Runtime

After our submission, we tried to push the performance of our hard constraint solver as far as possible. Given that the hard constraint solver parameters and soft constraint solver parameters of our algorithm are effectively decoupled, and that in general we want the hard constraint solver to find a solution that is feasible as quickly as possible, we ran FocusedILS 2.3 on a version of our algorithm that terminated as soon as a feasible solution was found. The parameters and domains used were those marked "hard" in Table 1, with each FocusedILS run starting from a default configuration corresponding to our ITC2007 submission. We used FocusedILS to optimise this version of our algorithm for mean runtime required to generate a feasibile solution, using ten independent runs of FocusedILS with an algorithm runtime cutoff of ten CPU minutes and 120 CPU hours allocated per FocusedILS run. Each independent run of FocusedILS used a different training set of size 64, four runs on each of the 16 public ITC2007 instances, ordered randomly. Optimising for runtime in this manner allows FocusedILS to explore more configurations than when tuning for solution quality due to the pruning mechanism that is only available when optimising for runtime.

7.3 Exploratory Soft Constraint Solver Optimisation

After the results of ITC2007 were made available, we undertook a set of experiments with the aim of improving our soft constraint solver to a state competitive with the solver of Cambazard et al. that ranked first in our track of the competition. We performed a set of five exploratory runs using FocusedILS 2.3 to configure our original (smaller) set of soft constraint parameters to minimise the CSCV value over the public ITC2007 instances. These runs were allocated 120 CPU hours each with an algorithm runtime cutoff of 600 CPU seconds. This 600 second runtime is longer than the 384 second timelimit for our machines

used during the competition, and was chosen because at lower runtimes the solver of Cambazard et al. often failed to find a feasible solution, making comparison of soft constraint solver performance difficult. This particular cutoff was chosen to achieve a balance between trying to ensure that feasible schedules were always located and trying to stay close to short runtimes that are desireable in a real-world timetabling scenario.

Using the results of this preliminary experiment as a guide, we performed a substantial reimplementation, modularisation and parameter domain expansion of our original solver. Our original four soft constraint parameters were expanded to the eleven present in Table 1, although at that point they did not have the same domains. We then used FocusedILS 2.3 to configure this solver to minimise the runtime required to reach the median CSCV value of the version resulting from the previous design phase, using the previous best parameter configuration as a starting point. This set of five FocusedILS runs was also allocated 120 CPU hours per run with a runtime cutoff of 1200 seconds to allow for poorer configurations to reach the median CSCV and provide guidance to the search. This resulted in small improvements to our solver performance, but not enough to change our standing relative to the other ITC2007 finalists or to bring us substantially closer to the performance of the winning solver by Cambazard et al. The major benefit of those configuration experiments was that they allowed us to realise that our intuitions about appropriate domains for our parameters were incorrect. In both experiments, FocusedILS repeatedly selected parameter values at the boundaries of their given domains. The domains of these parameters were therefore aggressively expanded, yielding the domains indicated in Table 1. The parameter configuration resulting from these tuning runs was used as a starting point for our subsequent and final design phase.

7.4 p-Value Performance Metric Optimisation of the Soft Constraint Solver

Given our stated goal of matching or exceeding the solver performance of Cambazard et al., our final development phase used FocusedILS 2.4 to optimise the soft constraint solver parameters using the p-value performance metric. We generated an empirical SQD for each of the sixteen public ITC2007 instances by performing twenty runs of Cambazard et al.'s solver, with ten CPU minutes of runtime each. By minimizing the mean of the p-value performance metric values of runs using our solver on these instances, our goal was to find algorithm parameter configurations that resulted in performance far in the left tail of the Cambazard et al. SQDs. Our intuition was that this would result in a solver configuration that also performed better than the solver of Cambazard et al. according to other performance criteria, in particular mean relative solution quality and the ranking metric used in ITC2007.

The parameters and domains used were those marked "soft" in Table 1, with each FocusedILS run starting from a default configuration corresponding to the values obtained from the experiments in Section 7.3. The training sets used for FocusedILS were the same as those used in Section 7.2, and we again used ten independent FocusedILS runs of 120 CPU hours each with an algorithm runtime cutoff of ten CPU minutes.

8 Experimental Results

The top three finalists of track 2 of ITC2007, Cambazard et al., Atsuta et al. and our original solver, achieved mean ranks of 13.904, 24.427 and 28.340, respectively, using the ITC2007 scoring method. The relative performance of these solvers across the full set of instances ('public' and 'private', where the latter are the ones that were used as part of the official ranking in ITC2007) is illustrated in more detail in Figure 1. From this data, it is evident that our ITC2007 solver performed very well with respect to finding feasible timetables (in fact, it arguably outperformed all other finalists in that respect). This result is consistent with the fact that the design of our solver, and in particular the automated configuration process performed on it, had been focussed on the hard constraint solver. At the same time, our ITC2007 solver was considerably less successful than its competitors in minimising soft constraint violations. It was this observation that motivated to a large extent the subsequent work on our solver reported in this paper.

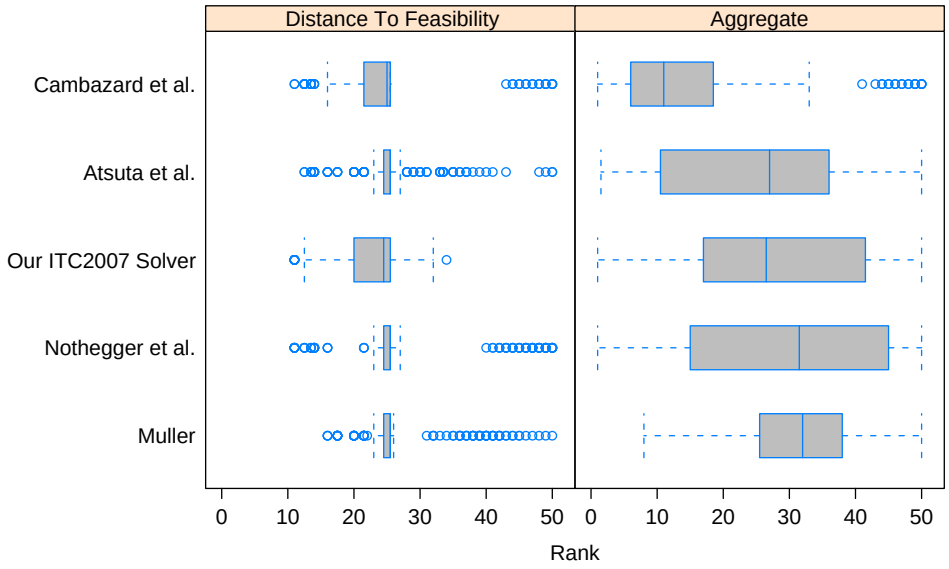

Figure 1: Performance results for the 5 finalists of ITC2007. Generated using the published results of the competition, which used 10 independent runs of each solver on each of the 16 public instances as well as 8 private instances that had not been previously available. These box plots were made using the distribution of ranks for each solver, based on two criteria. The distance to feasibility portion of this plot ranked these 50 runs per instance using only the DTF measure, with ties broken by assigning the average rank of all runs having the same value. The aggregate plot ranked the runs first by DTF, with ties broken by CSCV. Runs having identical DTF and CSCV were again assigned the average rank of all runs having the same value. It is clear that our solver had very good performance by the DTF measure, with only 4 of the 240 runs failing to find a feasibile solution compared to 18 of 240 by the solver of Cambazard et al.

The solver configuration with the best training quality after the hard constraint optimisation (see Section 7.2) for runtime is annotated in Table 1. The corresponding solver achieved a training quality (average runtime to reach feasibility across 4 runs on each of the 16 public ITC2007 instances) of 10.096 seconds. When evaluated using a larger set of 12 runs on each of the same 16 public instances, the mean runtime to reach feasibility was found to be 13.99 seconds – a substantial improvement compared to the mean runtime of 33.94 seconds on the same set of 192 runs measured for our ITC2007 solver. In addition, while our ITC2007 solver took over 400 seconds to find feasible timetables for some instances, the maximum time for finding a feasible schedule on any of the public instances observed in any run of our new solver was merely 97 seconds.

In the next development phase, we achieved substantial further performance improvements, based on p-Value performance metric optimisation of the soft constraint solver. The resulting solver, whose parameter configuration can be seen from Table 1, achieved a training quality of 0.3281, which indicates that the mean solution quality it reached after 600 CPU sec exceeded that reached by about 67% of the runs of the solver of Cambazard et al. within the same time. As can be seen from Table 2, this new solver performed overall substantially better than the solver by Cambazard et al., which had won ITC2007; in particular, our solver tends to find better timetables for all 16 public instances (which had been utilized during the design of both competing solvers) as well as for 5 of the 8 private instances (which had not been used during any design stage). As can also be seen in Table 2, on four instances, the median quality of the timetables produced by our solver is better by more than one order of magnitude than that of the winner of ITC2007, while the converse holds for only two instances. Additionally, there are four instances on which our solver finds timetables without any soft constraint violations, while the solver by Cambazard et al. does not, and one instance on which the latter does not even find a feasible timetable within 600

	Cambazard et al.			Our Solver		
Instance ID	q0.25	median	q0.75	q0.25	median	q0.75
1	563.25	681.50	835.50	**192.00**	**348.00**	**421.50**
2	707.00	1074.50	1264.75	**19.75**	**27.50**	**244.00**
3	207.50	228.50	262.50	**169.75**	**188.50**	**202.00**
4	350.50	383.00	426.00	**286.00**	**312.00**	**346.25**
5	2.00	3.00	4.00	**0.00**	**0.00**	**3.25**
6	**0.00**	**0.00**	**1.25**	**0.00**	**0.00**	38.75
7	7.00	8.00	8.00	**5.00**	**5.00**	**6.00**
8	**0.00**	**0.00**	0.00	**0.00**	**0.00**	**0.00**
9	1410.75	1526.00	1681.50	**27.75**	**137.00**	**483.75**
10	1551.50	1850.50	2273.75	**30.00**	**44.00**	**68.25**
11	255.75	290.50	347.25	**168.75**	**203.50**	**243.00**
12	322.50	390.00	465.75	**213.50**	**283.50**	**351.75**
13	32.00	114.50	177.75	**0.00**	**46.00**	**115.00**
14	0.00	1.00	1.00	**0.00**	**0.00**	**0.00**
15	**0.00**	**0.00**	**152.50**	**0.00**	**0.00**	228.25
16	11.00	62.00	114.50	**2.00**	**10.0**	**74.0**
17	2.00	8.00	17.00	**0.00**	**0.00**	**1.00**
18	**0.00**	6.50	77.25	**0.00**	**0.00**	**0.00**
19	1793.00	2044.50	2262.00	**3.0**	**4.0**	**416.75**
20	**529.00**	**551.50**	**602.25**	647.75	683.00	718.00
21	**0.00**	**0.00**	**1.25**	7.00	13.00	47.25
22	N/A	N/A	N/A	**493.25**	**948.50**	**1702.75**
23	1437.50	1611.00	1788.25	**735.25**	**890.50**	**1135.25**
24	**19.75**	**27.00**	**46.25**	451.25	555.50	694.75

Table 2: For both the solver of Cambazard et al. and our automatically configured solver, 100 runs of 600 CPU seconds were performed on the public and private ITC2007 instances (numbered 1–16 and 17–24, respectively). The table shows the median and both upper and lower quartiles for CSCV on each instance. For instance 22, the solver of Cambazard et al. found a feasible solution on only 18 of the 100 runs performed. This also occurred in three runs on instance 10 and four runs on instance 19. Of the 18 successful runs on instance 22, the median and quartiles were 2349, 2071.5 and 2402.75. Numbers given in bold are those with the best performance for the given statistic on that instance.

CPU sec. When using the ITC2007 ranking method with only these two solvers competing for ranks, over ten randomly selected 600 second runs for each solver, our solver achieves a mean rank of 8.746 while the solver of Cambazard et al. achieves a mean rank of 12.254. These results clearly demonstrate the substantial performance improvement achieved by our solver compared to the previous state of the art. We also note that considering the differences in relative performance across problem instances illustrated in Table 2, a solver that outperforms the winner of ITC2007 on every problem instance considered here in terms of solution quality (at the price of increased overall CPU time) can be easily obtained by running both our new solver and that of Cambazard et al. and reporting the best solution found by either of them.

9 Conclusions and Future Work

We have presented a novel, highly modular and parameterised algorithm for solving the PECTP introduced in ITC2007 and demonstrated that it substantially improves the state of the art for solving this problem. We obtained this algorithm by making strong use of automated algorithm design techniques. In particular, we used FocusedILS to search for a performance-optimising design in the huge space of candidate algorithms spanned by our parametric solver framework. In order to achieve the results reported in this work, we optimised the performance of the hard and soft constraint solving procedures within our framework during separate phases, using different optimisation objectives, including the newly developed p-value performance metric. We believe that this metric, as well as the general multi-phased algorithm design approach followed here, will prove to be useful in the automated design of high-performance solvers for other constraint optimisation problems.

It would be particularly interesting to use these tools in combination with an appropriately expanded version of our solver framework to tackle the two other timetabling problems used in ITC2007 (an examination and a curriculum-based timetabling problem [24, 10]). We have also recently begun working towards applying our timetabling algorithm to real-world timetabling problems at the University of British Columbia.

Acknowledgments.

The authors would like to thank Hadrien Cambazard for providing source code for the solver presented in [7], as well as valuable email feedback.

References

[1] Adenso-Díaz, B. and Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114.

[2] Arntzen, H. and Løkketangen, A. (2003). A tabu search heuristic for a university timetabling problem. In *Proceedings of the Fifth Metaheuristics International Conference*, Kyoto, Japan.

[3] Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Aguilera, M. J. B., Blum, C., Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer.

[4] Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing Series. springer-berl.

[5] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M., Schultz, A., Miller, J., Burke, E., and Jonoska, N., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18, New York. Morgan Kaufmann Publishers.

[6] Blöchliger, I. and Zufferey, N. (2008). A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975.

[7] Cambazard, H., Hebrard, E., OSullivan, B., and Papadopoulos, A. (2008). Local search and constraint programming for the post enrolment-based course timetabling problem. In Burke, E. K. and Gendreau, M., editors, *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, CA. Université de Montréal.

[8] Chiarandini, M., Birattari, M., Socha, K., and Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432.

[9] Chiarandini, M., Fawcett, C., and Hoos, H. (2008). A modular multiphase heuristic solver for post enrollment course timetabling (extended abstract). In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*.

[10] Gaspero, L. D., McCollum, B., and Schaerf, A. (2007). The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, Belfast, United Kingdom.

[11] Hoos, H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

[12] Hoos, H. H. (2008). Computer-aided design of high-performance algorithms. Technical Report TR-2008-16, University of British Columbia, Department of Computer Science.

[13] Huang, D., Allen, T., Notz, W., and Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466.

[14] Hutter, F., Babić, D., Hoos, H. H., and Hu, A. J. (2007a). Boosting Verification by Automatic Tuning of Decision Procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, pages 27–34, Washington, DC, USA. IEEE Computer Society.

[15] Hutter, F., Hoos, H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. Technical Report TR-2009-01, University of British Columbia.

[16] Hutter, F., Hoos, H. H., and Stützle, T. (2007b). Automatic algorithm configuration based on local search. In *Proc. of the Twenty-Second Conference on Artificial Intelligence (AAAI '07)*, pages 1152–1157.

[17] Ibaraki, T. (2008). Problem solving by general purpose solvers. In *Proceedings of the Seventh International Symposium on Operations Research and its Applications (ISORA 2008), Lijiang, China*, Lecture Notes in Operations Research. Asia-Pacific Operations Research Center (APORC).

[18] KhudaBukhsh, A., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2009). Satenstein: Automatically building local search sat solvers from components. Accepted for publication.

[19] Kostuch, P. (2005). The university course timetabling problem with a three-phase approach. In Burke, E. and Trick, M., editors, *PATAT*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125. Springer Verlag, Berlin, Germany.

[20] Lach, G. and Lübbecke, M. (2008). Curriculum based course timetabling: Optimal solutions to the udine benchmark instances. Technical Report 9, TU Berlin, Institut für Mathematik.

[21] Lach, G. and Lübbecke, M. (2008). Optimal university course timetables and the partial transversal polytope. In McGeoch, C., editor, *7th International Workshop on Efficient and Experimental Algorithms (WEA08)*, volume 5038 of *LNCS*, pages 235–248, Berlin. Springer.

[22] Lewis, R., Paechter, B., and McCollum, B. (2007). Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University.

[23] Mayer, A., Nothegger, C., Chwatal, A., and Raidl, G. (2008). Solving the post enrolment course timetabling problem by ant colony optimization. In Burke, E. K. and Gendreau, M., editors, *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, CA. Université de Montréal.

[24] McCollum, B., McMullan, P., Burke, E. K., Parkes, A. J., and Qu, R. (2007). The second International Timetabling Competition: Examination timetabling track. Technical Report QUB/IEEE/Tech/ITC2007/-Exam/v4.0/17, Queen's University, Belfast.

[25] Müller, T. (2008). ITC2007 solver description: A hybrid approach. In Burke, E. K. and Gendreau, M., editors, *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008*, Montreal, CA. Université de Montréal.

[26] Socha, K., Sampels, M., and Manfrin, M. (2003). Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In Cagnoni, S., Cardalda, J. R., Corne, D., Gottlieb, J., Guillot, A., Hart, E., Johnson, C., Marchiori, E., Meyer, J.-A., Middendorf, M., and Raidl, G., editors, *Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. springer-lncs.

[27] Thachuk, C., Shmygelska, A., and Hoos, H. (2007). A replica exchange monte carlo algorithm for protein folding in the hp model. *BMC Bioinformatics*, 8(342).