Charles University in Prague
Faculty of Mathematics and Physics

Constraint-based Timetabling

Summary of Ph.D. Thesis

Prague, 2005

Tomáš Müller

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Rozvrhování s omezujícími podmínkami

Autoreferát doktorandské disertační práce
k získání akademicko-vědeckého titulu doktor.

Obor I-1: Teoretická informatika

Praha, 2005 **Tomáš Müller**

Disertační práce byla vypracována v rámci doktorandského studia, které uchazeč absolvoval na katedře teoretické informatiky Univerzity Karlovy v letech 2001-2005.

Uchazeč: RNDr. Ing. Tomáš Müller

Školitel: Doc. RNDr. Roman Barták, Ph.D.
Katedra teoretické informatiky MFF UK,
Malostranské náměstí 2/25, 118 00 Praha 1

Školící pracoviště: Katedra teoretické informatiky MFF UK,
Malostranské náměstí 2/25, 118 00 Praha 1

Oponenti:

Prof. Gilles Pesant
Centre de recherche sur les transports, Université de Montréal
C.P. 6128, succ. Centre-ville
H3C 3J7 Montreal, Canada

Prof. RNDr. Peter Vojtáš, DrSc.
Katedra softwarového inženýrství MFF UK
Malostranské nám. 25, 118 00 Praha 1

Autoreferát byl rozeslán dne: ....

Obhajoba se koná dne .... v .... hod. před komisí pro obhajoby disertačních prací oboru I-1 na MFF UK, Ke Karlovu 3, Praha 2, v místnosti č. 105

S doktorandskou disertační prací je možno se seznámit na studijním oddělení pro doktorské studium MFF UK, Ke Karlovu 3, 120 00 Praha 2.

Prof. RNDr. Petr Štěpánek, DrSc.
předseda komise pro obhajoby
disertačních prací v oboru teoretická informatika
Katedra teoretické informatiky a matematické logiky MFF UK
Malostranské nám. 25, 118 00 Praha 1

1. Introduction

Constraint programming is a natural tool for describing as well as solving a lot of problems from various areas. Its major advantage is its capability of precise declarative description of a problem using relations between variables. It is based on a strong theoretical basis and it has wide practical applications in areas of evaluation, modelling, and optimisation.

Timetabling is one of the typical examples of constraint programming application. The task is to allocate activities in time and space respecting various constraints and to satisfy as nearly as possible a set of desirable objectives. A typical constraint is the request that activities which are using the same resource (e.g., a room, a machine, an operator, …) can not overlap in time or that a resource is of a certain capacity, restricting e.g. how many activities can use it at the same time. In addition, there are usually relations between activities and constraints restricting what resources an activity should or can use.

There are a lot of timetabling problems from various areas, for example, there is course, examination, transport, workforce, sport timetabling etc. In this thesis we will concentrate on course timetabling.

There are two major objectives of this Ph.D. thesis: We would like to find, describe and experimentally verify a constraint-based algorithm which is applicable to course timetabling problems as well as to other constraint satisfaction and optimisation problems. Moreover, with such an algorithm, we would like to tackle a real-life large scale timetabling problem. The whole Ph.D. work was motivated by this possibility to create an algorithm which is able to solve a given real-life problem and which can produce a solution fully acceptable by the users.

2. Overview

Many real-life industrial and engineering problems can be modelled as finite constraint satisfaction problems (CSP) [Tsa93]. A CSP consists of a set of variables associated with finite domains and a set of constraints restricting the values that the variables can simultaneously take. In a complete solution of a CSP, a value is assigned to every variable from the variable's domain, in such a way that every constraint is satisfied.

Most algorithms for solving CSPs search systematically through the possible assignments of values to variables. Such algorithms are guaranteed to find a solution, if one exists, or to prove that the problem has no solution. They start from an empty solution (no variable is assigned) that is extended towards a complete solution satisfying all the constraints in the problem. Backtracking occurs when a dead-end is reached. The biggest problem of such backtrack-based algorithms is that they typically make early mistakes in the search, i.e., a wrong early assignment can cause a whole subtree to be explored with no success. There are several ways of improving standard chronological backtracking. Look-back enhancements exploit information about the search which has already been performed, e.g., backmarking or backjumping [DF02]. Look-ahead enhancements exploit information about the remaining search space via filtering techniques (e.g., via maintaining arc consistency described in [BR97, BR01]) or variable and value ordering heuristics [MF00]. The last group of enhancements is trying to refine the search tree during the search process, e.g., dynamic backtracking [Gin93].

Local search algorithms [MF00] (e.g., min-conflict [MJP92] or tabu search [GH97]) perform an incomplete exploration of the search space by repairing an infeasible complete assignment. Unlike systematic search algorithms, local search algorithms move from one complete (but infeasible) assignment to another, typically in a non-deterministic manner, guided by heuristics. In general, local search algorithms are incomplete, they do not guarantee finding a complete solution satisfying all the constraints. However, these algorithms may be far more efficient (wrt. response time) than systematic ones in finding a solution. For optimisation problems, they can reach a far better quality in a given time frame.

There are several other approaches which try to combine local search methods together with backtracking based algorithms. For example, the decision repair algorithm presented in [JL02] repeatedly extends a set of assignments (called decisions) satisfying all the constraints, like in backtrack-based algorithms. It performs a local search to repair these assignments when a dead-end is reached (i.e., these decisions become inconsistent). After these decisions are repaired, the construction of the solution continues to the next dead-end. A similar approach is used in the algorithm presented in [Sch97] as well.

2.1. Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) is a triple $\Theta = (V,D,C)$, where

- $V = \{v_1,v_2,\ldots,v_n\}$ is a finite set of variables,
- $D =\{D_1,D_2,\ldots,D_n\}$ is a set of domains (i.e., D_i is a set of possible values for the variable v_i),
- $C = \{c_1,c_2,\ldots,c_m\}$ is a finite set of constraints restricting the values that the variables can simultaneously take.
- A solution to the constraint satisfaction problem Θ is a complete assignment of the variables from V that satisfies all the constraints.

For many constraint satisfaction problems it is hard or even impossible to find a solution in the above sense. For example, for over-constrained problems [FW92], there does not exist any complete assignment satisfying all the constraints. Therefore other definitions of problem solution like Partial Constraint Satisfaction were introduced [FW92]. In paper [BMR04], we proposed a new view of the problem solution based on a new notion of maximal consistent assignment. This approach is strongly motivated by the university timetabling problem but we believe that it is generally applicable. The basic idea behind is to assign as many variables as possible while still keeping the rest of the problem “consistent”. It means that the user may later relax some constraints in the problem (typically some of the constraints among the non-assigned variables that cause conflicts) so that after this change the assignment can be extended to other variables.

Formally, let Θ be a CSP and ζ be a consistency technique (for example arc consistency). We say that the constraint satisfaction problem is consistent if the consistency technique deduces no conflict (e.g., for arc consistency, the conflict is indicated by emptying some domain). We denote $\zeta(\Theta)$ the result of the consistency test which could be either true, if the problem Θ is ζ consistent, or false otherwise. Let Θ be a CSP and σ be a (partial) assignment of variables, then we denote $\Theta\sigma$ application of the assignment σ to the problem Θ, i.e., the domains of the variables in σ are reduced to a singleton value defined by the assignment. Finally, we say that a partial assignment σ is consistent with respect to some consistency technique ζ iff $\zeta(\Theta\sigma)$. Note that a complete consistent assignment is a solution of the problem. Note also that backtracking-based solving techniques typically extend a partial consistent assignment towards a complete (consistent) assignment.

As we already mentioned, for some problems there does not exist any complete consistent assignment; these problems are called over-constrained. In such a case, we propose to look for the maximal consistent assignment. We say that the consistent assignment is maximal for a given CSP if there is no consistent assignment with a larger number of assigned variables. We can also define a weaker version, so called locally maximal consistent assignment. Locally maximal consistent assignment is a consistent assignment that cannot be extended to another variable(s). Notice the difference between the above two notions. The maximal consistent assignment is defined using the cardinality of the assignment (the number of assigned variables) so it has a global meaning while the locally maximal consistent assignment is defined using a subset relation, i.e., it is not possible to assign an additional variable without getting inconsistency. It is pretty easy (fast) to extend any consistent assignment to a locally maximal consistent assignment. In fact, every branch of the search tree defines such a locally maximal consistent assignment. Visibly, the maximal consistent assignment is the largest (using cardinality) locally maximal consistent assignment.

2.2. Minimal Perturbation Problem

Most existing solvers are designed for static problems. These problems can be expressed, solved by appropriate means, and the solution applied without any change to the problem statement. Many real-life problems [Koc02, VJ03, SW00, Ian04], however, are subject to change. Additional input requirements produce a new problem derived from the original problem. The dynamics of such a problem may require changes during the solution process, or even after a solution is generated. In many real situations, it is necessary to alter the solution process so that the dynamic aspects of the problem definition are taken into account.

Problem changes may result from changes to environmental variables, such as broken machines, delayed flights, or other unexpected events. Users may also specify new properties based on the solution found so far. The goal is to find an improved solution for the user. Naturally, the problem solving process should continue as smoothly as possible after any change in the problem formulation. In particular, the solution of the altered problem should not differ significantly from the solution found for the original problem.

There are several reasons to keep a new solution as close as possible to the existing solution. If the solution has already been published, such as the assignment of gates to flights, frequent changes would confuse passengers. Moreover, changes to a published solution may necessitate other changes if initially satisfied wishes of users are violated. This may create an avalanche reaction.

Dynamic problems appear frequently in real-life planning and scheduling applications where the task is to “minimally reconfigure schedules in response to a changing environment" [SW00]. Dynamic changes in the context of timetabling problems have started to be studied at [EGJ03]. Issues of interactive timetabling which needs to handle dynamic aspects of the problem were discussed in [CDJD04, PMM04, MB02]. A survey of existing approaches to dynamic scheduling can be found in [Koc02]. In an annotated bibliography on dynamic constraint solving [VJ03], it is notable that only four papers were devoted to the problem of minimal changes.

In paper [BMR04], we presented a new formal model of the minimal perturbation problem that is applicable to over-constrained problems as well as to problems where finding a complete solution is hard. Recall that the idea of MPP is to define a solution of the altered problem in such a way that this solution is as close as possible to the (partial) solution of the original problem.

We define a minimal perturbation problem (MPP) as a quadruple $\Pi = (\Theta, \Theta', F, \alpha)$, where:

- Θ, Θ' are two CSPs called an *initial problem* and a *changed problem*,
- F is a mapping of the variables from Θ to Θ' (see below for details), and
- α is a (locally) maximal consistent assignment for Θ called *initial assignment.*

The function F defines how the problem Θ is changed in terms of variables. It is (almost) one-to-one mapping of the variables from Θ to the variables from Θ'. For some variables v from Θ, the function F might not be defined which means that the variable v is removed from the problem. However, if the function F is defined[1] then it is unique (it is a one-to-one mapping), i.e., $(v \neq u \ \& \ !F(v) \ \& \ !F(u)) \Rightarrow F(v) \neq F(u)$. Also, for some variables v' from Θ', the origin might not be defined (i.e., there is no variable v such that F(v) = v'), which means that the variable v' is added to the problem. Notice also that the constraints and domains can be changed arbitrarily when going from Θ to Θ'. We do not need to capture such changes using the mapping functions like F because we are concerned primarily about the variable assignments.

Let σ be a (partial) assignment for Θ and γ be a (partial) assignment for Θ'. Then we define $W_\Pi(\sigma,\gamma)$ as a set of variables v from Θ such that the assignment of v in σ is different from F(v) in γ, i.e., $W_\Pi(\sigma,\gamma) = \{v \in \Theta \mid v/h \in \sigma \ \& \ F(v)/h' \in \gamma \ \& \ h \neq h'\}$[2]. We call $W_\Pi(\sigma,\gamma)$ a distance set for σ and γ in Π and the elements of the set are called perturbations.

A solution to the minimal perturbation problem $\Pi = (\Theta, \Theta', F, \alpha)$ is a (locally) maximal consistent assignment β for Θ' such that the size of the distance set $W_\Pi(\alpha,\beta)$ is minimal. The idea behind the solution of MPP is apparent – the task is to find the best possible assignment of the variables for the new problem in such a way that it differs minimally from the existing variable assignment of the initial problem.

Let us summarize now the two criteria used when solving MPP: the first criterion is maximizing the number of assigned variables, the second criterion is minimizing the number of perturbations between the resultant solution and the previous (initial) solution. These criteria are combined lexicographically to get an objective function.

[1] Denoted !F(v), meaning F is defined for variable v

[2] For simplicity reasons we write $v \in \Theta$ which actually means $v \in V$, where $\Theta = (V,D,C)$.

3. Timetabling

A *timetable* shows when particular events are to take place. It does not necessarily imply an allocation of resources. Thus a published bus or train timetable shows when journeys are to be made on a particular route or routes. It does not tell us which vehicles or drivers are to be assigned to particular journeys. The allocation of vehicles and drivers is part of the scheduling process. Although timetabling is strictly the design of the pattern of journeys, this pattern may be devised as part of a process which bears in mind whether it is likely that an efficient schedule may be fitted to the resulting journey pattern.

A.Wren defines timetabling [Wren96] as follows:

Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives.

Timetabling has long been known to belong to the class of problems called NP-complete [CK96], i.e., no method of solving it in a reasonable (polynomial) amount of time is known.

3.1. Purdue Timetabling Problem

Our work is motivated by the class timetabling problem at Purdue University [RM03, MR04]. Here a timetable for large lecture classes is constructed by a central scheduling office in order to balance the requirements of many departments offering large classes that serve students from across the university. Smaller classes, usually focused on students in a single discipline, are timetabled by "schedule deputies" in the individual departments. Such a complex timetabling process, including subsequent student registration, takes a rather long time. Initial timetables are generated about half a year before the semester starts. The importance of creating a solver for a dynamic problem increases with the length of this time period and the need to incorporate various changes that arise.

As for Fall 2004 semester, this problem consists of about 830 classes (forming almost 1800 meetings) having a high density of interaction that must fit within 50 lecture rooms with capacities up to 474 students. Room availability is a major constraint for Purdue. Overall utilization of the time available in rooms exceeds 78%; moreover, it is around 94% for the four largest rooms. About 90,000 course requests by almost 30,000 students must also be considered. 8.4% of class pairs have at least one student enrolment in common.

The timetable maps classes (students, instructors) to meeting locations and times. A major objective in developing an automated system is to minimize the number of potential student course conflicts which occur during this process. This requirement substantially influences the automated timetable generation process since there are many specific course requirements in most programs of study offered by the University.

To minimize potential time conflicts, Purdue has historically subscribed to a set of standard meeting patterns. With few exceptions, 1 hour $\times$ 3 day per week classes meet on Monday, Wednesday, and Friday at the half hour (7:30, 8:30, 9:30, ...). 1.5 hour $\times$ 2 day per week classes meet on Tuesday and Thursday during set time blocks. 2 or 3 hours $\times$ 1day per week classes must also fit within specific blocks, etc. Generally, all meetings of a class should be taught in the same location. Such meeting patterns are of interest to the problem solution as they allow easier changes between classes having the same or similar meeting patterns.

Another aspect of the timetabling problem that must be considered is the need to perform student sectioning. Most of the classes in the large lecture problem (about 75%) correspond to single-section courses. Here we have exact information about all students who wish to attend a specific class. The remaining courses are divided into multiple sections. In this case, it is necessary to divide the students enrolled in each course into sections that will constitute the classes.

Currently, the timetable for Purdue University is constructed manually. We have proposed an automated timetabling system to solve the initial as well as the minimal perturbation problem in [MR04, MRB05]. This solution is based on the iterative forward search algorithm described in the following chapters.

Problem Representation

Due to the set of standardized time patterns and administrative rules enforced at the university, it is generally possible to represent all meetings of a class by a single variable. This tying together of meetings considerably simplifies the problem constraints. Most classes have all meetings taught in the same room, by the same instructor, at the same time of day. Only the day of week differs. Moreover, these days and times are mapped together with the help of meeting patterns, e.g., a 2 hours $\times$ 3 day per week class can be taught only on Monday, Wednesday, Friday, beginning at 5 possible times.

In addition, all valid placements of a course in the timetable have a one-to-one mapping with values in the variable's domain. This domain can be seen as a subset of the Cartesian product of the possible starting times, rooms, etc. for a class represented by these values. Therefore, each value encodes the selected time pattern (some alternatives may occur, e.g., 1.5 hour $\times$ 2 day per week may be an alternative to 1 hour $\times$ 3 day per week), selected days (e.g., a two meeting course can be taught in Monday+Wednesday, Tuesday+Thursday, Wednesday+Friday), and possible starting times. A value also encodes the instructor and selected meeting room. Each such placement also encodes its preferences (soft constraints), combined from the preference for time, room, building and the room's available equipment. Only placements with valid times and rooms are present in a class's domain. For example, when a computer (classroom equipment) is required, only placements in a room containing a computer are present. Also, only rooms large enough to accommodate all the enrolled students can be present in valid class placements. Similarly, if a time slice is prohibited, no placement containing this time slice is in the class's domain.

The variable and value encodings described above leave us with only two types of hard constraints to be implemented: resource constraints (expressing that only one course can be taught by an instructor or in a particular room at the same time), and group constraints (expressing relations between several classes, e.g., that two sections of the same lecture can not be taught at the same time, or that some classes have to be taught one immediately after another).

There are three types of soft constraints in this problem. First, there are soft requirements on possible times, buildings, rooms, and classroom equipment (e.g., a computer or a projector). These preferences are expressed as integers:

- -2 … strongly preferred
- -1 … preferred
- 0 … neutral (no preference)
- 1 … discouraged
- 2 … strongly discouraged

As mentioned above, each value, besides encoding a class's placement (time, room, instructor), also contains information about the preference for the given time and room. Room preference is a combination of preferences on the choice of building, room, and classroom equipment. The second group of soft constraints is formed by student requirements. Each student can enrol in several classes, so the aim is to minimize the total number of student conflicts among these classes. Such conflicts occur if the student cannot attend two classes to which he or she has enrolled because these classes have overlapping times. Finally, there are some group constraints (additional relations between two or more classes). These may either be hard (required or prohibited), or soft (preferred), similar to the time and room preferences (from -2 to 2).

4. Iterative Forward Search Algorithm

The iterative forward search (IFS) algorithm that we propose here is based on ideas of local search methods [MF00]. However, in contrast to classical local search techniques, it operates over feasible, though not necessarily complete solutions. In such a solution, some variables can be left unassigned. Still all hard constraints on assigned variables must be satisfied. Similarly to backtracking based algorithms, this means that there are no violations of hard constraints.

Working with feasible incomplete assignments has several advantages compared to the complete infeasible assignments that usually occur in local search techniques. For example, when the solver is not able to find a solution (i.e., a complete feasible assignment), a largest feasible partial assignment (using cardinality) can be returned. Especially in interactive timetabling applications, such assignments are much easier to visualize, even during the search, since no hard constraints are violated. For instance, two lectures never use a single resource (e.g., a classroom) at the same time. Moreover, because of the iterative character of the search, the algorithm can easily start, stop, or continue from any feasible assignment, either complete or incomplete.

Iterative forward search works in iterations (see Figure 4.1. for algorithm). During each step, a variable A is initially selected. Typically an unassigned variable is chosen like in backtracking-based search. An assigned variable may be selected when all variables are assigned but the solution found so far is not good enough (for example, when there are still many violations of soft constraints). Once a variable A is selected, a value *a* from its domain D_A is chosen for assignment. Even if the best value is selected (whatever “best” means), its assignment to the selected variable may cause some hard conflicts with already assigned variables. Such conflicting assignments are removed from the solution and become unassigned. Finally, the selected value is assigned to the selected variable.

The algorithm attempts to move from one (partial) feasible solution σ to another via repetitive assignment of a selected value *a* to a selected variable A. During this search, the feasibility of all hard constraints in each iteration step is enforced by unassigning the conflicting assignments η (computed by function *conflicts*). The search is terminated when the requested solution is found or when there is a timeout expressed, for example, as a maximal number of iterations or available time being reached. The best solution found is then returned.

```
procedure ifs(V,D,C,α) // an initial assignment α is the parameter
   σ = α;          // current assignment
   β = α;          // best assignmen
   while canContinue(σ) do          // CSP problem Φ=(V,D,C) is
      A = selectVariable(σ);         // a global parameter
      a = selectValue(σ, A);         // for all used functions
      η = conflicts(σ, A, a); //conflicting assignments
      σ = (σ - η) ∪ {A/a};    //next assignment
      if better(σ, β) then β = σ;
   end while
   return β;
end procedure
```

Fig. 4.1. The iterative forward search algorithm

The above algorithm schema is parameterized by several functions, namely

- the termination condition (function *canContinue*),
- the solution comparator (function *better*),
- the variable selection (function *selectVariable*) and
- the value selection (function *selectValue*).

4.1.1. Termination Condition

The termination condition determines when the algorithm should finish. For example, the solver should terminate when the maximal number of iterations or some other given timeout value is reached. Moreover, it can stop the search process when the current assignment is good enough, e.g., all variables are assigned and/or some other solution parameters are in the required ranges. For example, the solver can stop when all variables are assigned and less than 10% of the soft constraints are violated. Termination of the process by the user can also be a part of the termination condition.

4.1.2. Solution Comparator

The solution comparator compares two assignments: the current assignment and the best assignment found. This comparison can be based on several criteria. For example, it can lexicographically order assignments according to the number of unassigned variables (a smaller number is better) and the number of violated soft constraints.

4.1.3. Variable Selection

As mentioned above, the presented algorithm requires a function that selects a variable to be (re)assigned during the current iteration step. This function is equivalent to a variable selection criterion in constraint programming. There are several guidelines for selecting a variable [Dech03]. In local search, the variable participating in the largest number of violations is usually selected first. In backtracking-based algorithms, the first-fail principle is often used, i.e., a variable whose instantiation is most complicated is selected first. This could be the variable involved in the largest set of constraints or the variable with the smallest domain, etc.

We can split the variable selection criterion into two cases. If some variables remain unassigned, the "worst" variable among them is selected, i.e., first-fail principle is applied. This may be, for example, the variable with the smallest domain or with the highest number of hard and/or soft constraints.

The second case occurs when all variables are assigned. Because the algorithm does not need to stop when a complete feasible assignment is found, the variable selection criterion for such case has to be considered as well. Here all variables are assigned but the assignment is not good enough, e.g., in the sense of violated soft constraints. We choose a variable whose change of a value can introduce the best improvement of the assignment. It may, for example, be a variable whose value violates the highest number of soft constraints.

It is possible for the assignment to become incomplete again after such an iteration because a value which is not consistent with all hard constraints can be selected in the value selection criterion. This can be also taken into account in the variable selection heuristics.

4.1.4. Value Selection

After a variable is selected, we need to find a value to be assigned to the variable. This problem is usually called "value selection" in constraint programming [Dech03]. Typically, the most useful advice is to select the best-fit value. So, we are looking for a value which is the most preferred for the variable and which causes the least trouble as well. This means that we need to find a value with the minimal potential for future conflicts with other variables. For example, a value which violates the smallest number of soft constraints can be selected among those with the smallest number of hard conflicts.

To avoid cycling, it is possible to randomize the value selection procedure. For example, it is possible to select the N best values for the variable and choose one of them randomly. Or, it is possible to select a set of values so that the heuristic evaluation for the worst value in this group is maximally p percent higher than the heuristic evaluation of the best value (where smaller value means better evaluation). Again, the value is selected randomly from this group. This second rule inhibits randomness if there is a single very good value.

4.2. IFS for Minimal Perturbation Problem

Let us first describe the meaning of perturbation in our approach. The changed problem differs from the initial problem by input perturbations. An input perturbation means that a variable must have different values in the initial and changed problem because of some input changes (e.g., a course must be scheduled at a different time in the changed problem).

The solution to the minimal perturbation problem (MPP) [SW00, BMR03, BMR04] can be evaluated by the number of additional perturbations. They are given by subtraction of the final number of perturbations and the number of input perturbations. An alternative approach is to consider variables in the initial and in the new problem which were assigned differently [RRH02, BMR03, BMR04]. As before, we need to minimize the number of such differently assigned variables.

Despite the local search nature of the algorithm, there are some adjustments needed to be able to effectively solve the MPP. The purpose of these adjustments is to minimize the number of additional perturbations. The easiest way to do this is to adopt variable and value selection heuristics which prefer the previous assignments (but not all the time, to avoid cycling).

For example, value selection heuristics can be adopted to select the initial value (if it exists) randomly with a probability P (it can be rather high, e.g., between 50-90%). If the initial value is not selected, the original value selection can be executed. Also, if there is the initial value in the set of best-fit values (e.g., among values with the minimal number of hard and soft conflicts), the initial value can be preferred as well. Otherwise, a value can be selected randomly from the constructed set of best-fit values. A disadvantage of such selection is that the probability P has to be selected carefully: if it is too small, the search can easily move away and the number of additional perturbations will grow during the search. If it is too high, the search will stick too much with the initial solution and, if there is no solution with a small amount of additional perturbations it will be hard to find a feasible solution.

Another approach is to limit the number of additional perturbations during the search. Furthermore, like in branch and bound, such a limit can be decreased when a feasible solution with the given number of perturbations is found. For example, if the number of additional perturbations is equal to or greater than the limit, the initial value has to be selected. Otherwise, if the number of additional perturbations is below the limit, the original value selection strategy is followed. The number of additional perturbations can also include variables that are not assigned yet whose initial values cause a hard conflict with the current assignments.

The above approaches can also be combined together, which can help to divide their influence during the search.

Variable selection heuristics can also be adopted to find a solution with a small number of perturbations. For example, when all variables are assigned, a variable that has an initial value but such value is not assigned to it should be selected, e.g., randomly among all variables that have not the initial value assigned, and that participate in the highest number of violated soft constraints.

4.3. Conflict-based Statistics

Value ordering heuristics play an important role in solving various problems. They allow choosing suitable values for particular variables to compute a complete and/or optimal solution. Problem-specific heuristics are usually applied because problem-independent heuristics are computationally expensive. Here we propose an efficient problem-independent approach to value selection whose aim is to recognize good and poor values.

Methods similar to conflict-based statistics (CBS) were successfully applied in earlier works [DF02, JL02]. In the weighting-conflict heuristics presented in [JL02], a weight is associated with each decision (assignment of a value to a variable). It characterizes the number of times that the decision has appeared in any conflict. It also takes the arity of a conflict into account. Each time a conflict is found, the weight of its decision constraints (i.e., assignments which are in the conflict) is increased by $1/r$ where r is the arity of the conflicting constraint. These weights are used for selections of decisions which are negated when a dead-end is reached.

In our approach, the conflict-based statistics works as an advice in the value selection criterion. It helps to avoid repetitive, unsuitable assignments of the same value to a variable. In particular, conflicts caused by this assignment in the past are memorized. In contrast to the weighting-conflict heuristics proposed in [JL02], conflict assignments are memorized together with the assignment which caused them. Also, we propose our statistics to be unlimited, to prevent short-term as well as long-term cycles.

The main idea behind conflict-based statistics is to memorize conflicts and discourage their future repetition. For instance, when a value is assigned to a variable, conflicts with some other assigned variables may occur. This means that there are one or more constraints which prohibit the applied assignment together with the existing assignments. A counter, tracking how many times such an event occurred in the past, is stored in memory. If a variable is selected for an assignment (or reassignment) again, the stored information about repetition of past conflicts is taken into account.

Conflict-based statistics is a data structure that memorizes hard conflicts which have occurred during the search together with their frequency and assignments which caused them. More precisely, it is an array

$$CBS[V_a = v_a \rightarrow \neg\, V_b = v_b] = c_{ab.}$$

It means that the assignment $V_a = v_a$ caused c_{ab} times a hard conflict with the assignment V_b=v_b in the past. Note that it does not imply that these assignments $V_a = v_a$ and $V_b = v_b$ cannot be used together in case of non-binary constraints. The proposed conflict-based statistics does not actually work with any constraints. It only memorizes the conflict assignments together with the assignment which caused them. This helps us capture similar cases as well, e.g., when the applied assignment violates a constraint different from the past ones, but some of the created conflicts are the same. It also reduces the total space allocated by the statistics.

The conflict-based statistics can be implemented as a hash table. Such structure is empty in the beginning. During computation, the structure contains only non-zero counters. A counter is maintained for a tuple $[A = a \rightarrow \neg\, B = b]$ in case that the value a was selected for the variable A and this assignment $A = a$ caused a conflict with an existing assignment $B = b$. An example of this structure

$$A = a \quad \rightarrow \quad 3 \times \neg\, B = b, \quad 4 \times \neg\, B = c, \quad 2 \times \neg\, C = a, \quad 120 \times \neg\, D = a$$

expresses that variable B conflicts three times with its assignment *b* and four times with its assignment *c*, variable C conflicts two times with its assignment *a* and D conflicts 120 times with its assignment *a*, all because of later selections of value *a* for variable A. This structure can be used in the value selection heuristics to evaluate conflicts with the assigned variables. For example, if there is a variable A selected and its value *a* is in conflict with an assignment $B = b$, we know that a similar problem has already occurred 3 times in the past, and the conflict $A = a$ can be weighted with the number 3.

The conflict-based statistics is being used in the value selection criterion. A trivial min-conflict value selection criterion selects a value with the minimal number of conflicts with the existing assignments. This heuristics can be easily adapted to a weighted min-conflict criterion. Here the value with the smallest sum of the number of conflicts multiplied by their frequencies is selected. Stated in another way, the weighted min-conflict approach helps to select a certain value that might cause more conflicts than another value. The point is that these conflicts are not so frequent, and therefore they have a lower weighted sum. Our hope is that it can considerably help the search to get out of a local minimum.

4.4. Summary

In this section, we have presented the iterative forward search algorithm which is a mixture of systematic search and the local search approach. In the following section, we will discuss some of its extensions and later on we will present some computational results of this algorithm used on a CSP problem as well as on the large lecture timetabling problem on Purdue University.

The very first version of this algorithm was presented in [MB01] and in the diploma thesis [Mul01] as an ad-hoc solution for the iterative lecture timetabling problem. Its application on lecture timetabling problem on Mathematics and Physics Faculty of Charles University was presented in [MB02]. The applicability of this algorithm on the n-queens problem was presented in [Mul02].

The iterative forward search algorithm in the form as it is presented in this chapter for solving of a general CSP with various extensions (conflict-based heuristics, maintenance of arc consistency, its extension towards dynamic backtracking) was presented in [MBR04]. Its application to the minimal perturbation problem of Purdue University timetabling was presented in [MR04, MRB05]. Also, we used this algorithm in comparison with a branch&bound algorithm designed for solving MPP problem in [BMR04] where it was better than the proposed branch&bound algorithm.

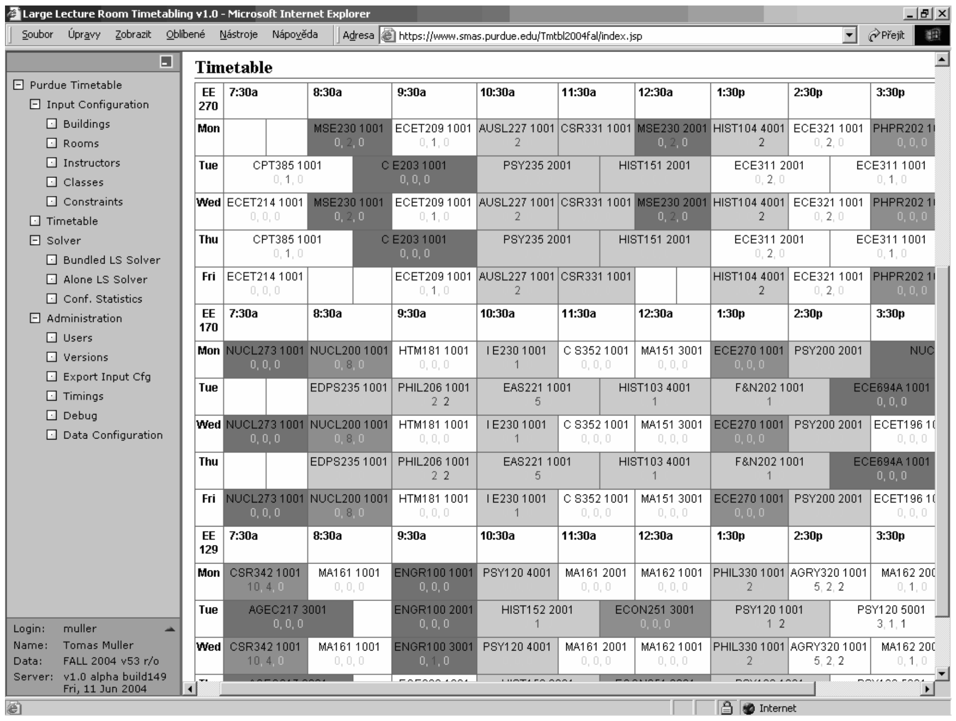
5. Experimental Results

The iterative forward search has been implemented in Java. It contains a general implementation of the iterative search algorithm. The general solver operates over abstract variables and values with a selection of available extensions, basic general heuristics, solution comparators, and termination functions. It may be customized to fit a particular problem (e.g., as it has been extended for Purdue University timetabling) by implementing variable and value definitions, adding hard and soft constraints, and extending the parametric functions of the algorithm.

In this chapter, we present capabilities of iterative forward search on the real-life course timetabling problem of Purdue University (see chapter 3.1 for the description of the problem). Results from solving both initial as well as minimal perturbation problems are presented.

The following experiments were performed on the complete Fall 2004 data set, including 830 classes to be placed in 50 classrooms. The classes included represent 89,677 course requirements for 29,808 students. We have achieved similar results with Fall 2001, Spring 2005 and Fall 2005 data sets as well, even though they are quite different in the number of requirements (Fall 2004 is the most constrained one out of these four data sets).

Besides the discussed IFS solver, the timetabling application for Purdue University also contains a web-based graphical user interface (written using Java Server Pages) which allows management of several versions of the data sets (input requirements, solutions, changes, etc.), browsing the resultant solutions (see Figure 5.1), and tracking and managing changes between them.


Large Lecture Room Timetabling v1.0 - Microsoft Internet Explorer
Soubor Úpravy Zobrazit Oblíbené Nástroje Nápověda
Adresa https://www.smas.purdue.edu/Tmtbl2004fal/index.jsp
Přejít
Internet

Purdue Timetable
- Input Configuration
  - Buildings
  - Rooms
  - Instructors
  - Classes
  - Constraints
- Timetable
- Solver
  - Bundled LS Solver
  - Alone LS Solver
  - Conf. Statistics
- Administration
  - Users
  - Versions
  - Export Input Cfg
  - Timings
  - Debug
  - Data Configuration

Login: muller
Name: Tomas Muller
Data: FALL 2004 v53 r/o
Server: v1.0 alpha build149 Fri, 11 Jun 2004

Timetable

EE 270	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon		MSE230 1001 0, 2, 0	ECET209 1001 0, 1, 0	AUSL227 1001 2	CSR331 1001	MSE230 2001 0, 2, 0	HIST104 4001 2	ECE321 1001 0, 2, 0	PHPR202 1 0, 0, 0
Tue	CPT385 1001 0, 1, 0	C E203 1001 0, 0, 0		PSY235 2001	HIST151 2001		ECE311 2001 0, 2, 0	ECE311 1001 0, 1, 0	
Wed	ECET214 1001 0, 0, 0	MSE230 1001 0, 2, 0	ECET209 1001 0, 1, 0	AUSL227 1001 2	CSR331 1001	MSE230 2001 0, 2, 0	HIST104 4001 2	ECE321 1001 0, 2, 0	PHPR202 1 0, 0, 0
Thu	CPT385 1001 0, 1, 0	C E203 1001 0, 0, 0		PSY235 2001	HIST151 2001		ECE311 2001 0, 2, 0	ECE311 1001 0, 1, 0	
Fri	ECET214 1001 0, 0, 0		ECET209 1001 0, 1, 0	AUSL227 1001 2	CSR331 1001		HIST104 4001 2	ECE321 1001 0, 2, 0	PHPR202 1 0, 0, 0

EE 170	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 1	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001	NUC
Tue		EDPS235 1001	PHIL206 1001 2 2	EAS221 1001 5	HIST103 4001 1		F&N202 1001 1	ECE694A 1001 0, 0, 0	
Wed	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 1	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001	ECET196 1 0, 0, 0
Thu		EDPS235 1001	PHIL206 1001 2 2	EAS221 1001 5	HIST103 4001 1		F&N202 1001 1	ECE694A 1001 0, 0, 0	
Fri	NUCL273 1001 0, 0, 0	NUCL200 1001 0, 8, 0	HTM181 1001 0, 0, 0	I E230 1001 1	C S352 1001 0, 0, 0	MA151 3001 0, 0, 0	ECE270 1001 0, 0, 0	PSY200 2001	ECET196 1 0, 0, 0

EE 129	7:30a	8:30a	9:30a	10:30a	11:30a	12:30a	1:30p	2:30p	3:30p
Mon	CSR342 1001 10, 4, 0	MA161 1001 0, 0, 0	ENGR100 1001 0, 0, 0	PSY120 4001	MA161 2001 0, 0, 0	MA162 1001 0, 0, 0	PHIL330 1001 2	AGRY320 1001 5, 2, 2	MA162 200 0, 1, 0
Tue	AGEC217 3001 0, 0, 0		ENGR100 2001 0, 0, 0	HIST152 2001 1	ECON251 3001 0, 0, 0		PSY120 1001 1 2	PSY120 5001 3, 1, 1	
Wed	CSR342 1001 10, 4, 0	MA161 1001 0, 0, 0	ENGR100 3001 0, 1, 0	PSY120 4001	MA161 2001 0, 0, 0	MA162 1001 0, 0, 0	PHIL330 1001 2	AGRY320 1001 5, 2, 2	MA162 200 0, 1, 0

Fig. 6.1. Generated timetable in web-based graphical user interface.

Search Algorithm

The quality of a solution is expressed as a weighted sum combining soft time and classroom preferences, satisfied soft group constraints and the total number of student conflicts. This allows us to express the importance of different types of soft constraints. The following weights are considered in the sum:

- $W_{student}$ … weight of a student conflict,
- W_{time} … weight of a time preference of a placement,
- W_{room} … weight of a classroom preference of a placement,
- W_{constr} … weight of a preference of a satisfied soft group constraint,

Note that preferences of all time, classroom and group soft constraints go from -2 (strongly preferred) to 2 (strongly discouraged). So, for instance, the value of the weighted sum is increased when there is a discouraged time or room selected or a discouraged group constraint satisfied. Therefore, if there are two solutions, the better solution of them has the lower weighted sum of the above criteria. Moreover, additional solution parameters can be included in this comparison as well. For instance, we can also discourage empty half-hour time segments between classes (such half-hours cannot be used since all events require at least one hour) or usage of classrooms that are too large (having more than 50% excess seats).

The termination condition stops the search when the solution is complete and good enough (expressed by the solution quality described above and, in case of minimal perturbation problem also by the number of allowed perturbations). It also allows for the solver to be stopped by the user. Characteristics of the current and the best achieved solution, describing the number of assigned variables, time and classroom preferences, the total number of student conflicts, etc., are visible to the user during the search.

The solution comparator prefers a more complete solution (with a smaller number of unassigned variables). In case of minimal perturbation problem, a solution with a smaller number of perturbations among solutions with the same number of unassigned variables is preferred. If both solutions have the same number of unassigned variables (and perturbations), the solution of better quality is selected.

If there are one or more variables unassigned, the variable selection criterion picks one of them randomly. We have tried several approaches using domain sizes, number of previous assignments, numbers of constraints in which the variable participates, etc., but there was no significant improvement in this timetabling problem towards the random selection of an unassigned variable. The reason is, that it is easy to go back when a wrong variable is picked - such a variable is unassigned when there is a conflict with it in some of the subsequent iterations.

When all variables are assigned, an evaluation is made for each variable according to the above described weights. The variable with the worst evaluation is selected. This variable promises the best improvement in optimisation.

We have implemented a hierarchical handling of the value selection criteria. There are three levels of comparison. At each level a weighted sum of the criteria described below is computed. Only solutions with the smallest sum are considered in the next level. The weights express how quickly a complete solution should be found. Only hard constraints are satisfied in the first level sum. Distance from the initial solution (MPP), and a weighting of major preferences (including time, classroom requirements and student conflicts), are considered in the next level. In the third level, other minor criteria are considered. In general, a criterion can be used in more than one level, e.g., with different weights.

The above sums order the values lexicographically: the best value having the smallest first level sum, the smallest second level sum among values with the smallest first level sum, and the smallest third level sum among these values. As mentioned above, this allows diversification between the importance of individual criteria. In general, there can be more than three levels of these weighted sums, however three of them seem to be sufficient for spreading weights of various criteria for our problem.

The value selection heuristics also allow for random selection of a value with a given probability P_{rw} (random walk, e.g., 2%) and, in the case of MPP, to select the initial value (if it exists) with a given probability P_{init} (e.g., 70%).

Criteria used in the value selection heuristics can be divided into two sets. Criteria in the first set are intended to generate a complete assignment:

- Number of hard conflicts, weighted by $V_{conf,1}$ in the first level, $V_{conf,2}$ in the second level and $V_{conf,3}$ in the third level.
- Number of hard conflicts, weighted by their previous occurrences (see section 4.3 about conflict-based statistics) and by $V_{wconf,1..3}$.

Additional criteria allow better results to be achieved during optimisation:

- Number of student conflicts caused by the value if it is assigned to the variable, weighted by $V_{student,1..3}$.
- Soft time conflicts caused by a value if it is assigned to the variable, weighted by $V_{time,1..3}$.
- Soft classroom conflicts caused by a value if it is assigned to the variable (combination of the placement's building, room, and classroom equipment compared with preferences), weighted by $V_{room,1..3}$.

- Preferences of satisfied soft group constraints caused by the value if it is assigned to the variable, weighted by $V_{constr,1..3}$.
- Difference in the number of assigned initial values if the value is assigned to the variable (weighted by $V_{\Delta init,1..3}$): -1 if the value is initial, 0 otherwise, increased by the number of initial values assigned to variables with hard conflicts with the value.

Let us emphasize that the criteria from the second group are needed for optimisation only, i.e., they are not needed to find a feasible solution. Furthermore, assigning a different weight to a particular criteria influences the value of the corresponding objective function (e.g., see Figure 5.3 with comparison for optimisation criteria $V_{student,1..3}$ and $V_{time,1..3}$). The solver returns good results in reasonable time (e.g., in 30 minutes time limit) when the total sum of the weights used in additional criteria in the first level corresponds to one half of the weight $V_{wconf,1}$. The weights in the second level usually correspond to the weights used for the solution quality comparison ($W_{student}$, W_{time}, W_{room}, W_{constr}).

Below, we present two types of experiments. The first investigates finding an initial solution (e.g., when all requirements are placed in the system). This is followed by experiments on the minimal perturbation problem (e.g., where there is an existing solution plus a set of changes to be applied to it). Solving an initial problem can be seen as a special case of MPP where all variables are new and therefore have no initial values.

If not stated otherwise, the solution quality weights $W_{student}$, W_{time}, W_{room} and W_{constr} in the solution quality weighted sum are set to zero in the following experiments. First level weight for the weighted hard conflicts $V_{wconf,1}$ is set to 1, all other weights in the value selection criterion are set to zero. Also, there is no random value selection (P_{rw}=0). This way, by default, only the hard constraints are considered during the search. We will show how the other weights influence the search process and the overall solution quality. Also, if not stated otherwise, distances between buildings are not considered and department balancing is not used. The results presented in the following chapters were computed on 1GHz Pentium 3 PC running Windows 2000, with 512 MB RAM and JDK 1.4.2.

5.1. Initial Problem

The experiments in Figure 5.2 present the behaviour of the solver wrt. various settings of weights for particular criteria (the student conflicts, violated time preferences, and violated room preferences). It is important to see that the weights for particular criteria can be easily adjusted. It allows to emphasize or suppress particular optimization criteria and it results in the corresponding change of the solution quality.

Time refers to the amount of time required by the solver to find the presented solution. *Satisfied enrolments* gives the percentage of satisfied requirements for courses chosen by students. *Preferred time* and *preferred room* correspond to the satisfaction of time and room preferences respectively. 100% corresponds to a case when all classes are placed in their most preferred times or rooms, 0% means a case when the least preferred locations are used. Preferences of soft group constraints are not presented, since there are no such constraints in the Fall 2004 data set (all group constraints are either required or prohibited).

Test case	No preference	Students	Time	Room	Students + Time	Students + Time + Rooms	No CBS
Assigned variables [%]	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	98.42 ± 0.20
Time [min]	0.16 ± 0.03	8.45 ± 4.40	18.68 ± 6.50	0.17 ± 0.01	19.96 ± 5.34	14.79 ± 4.87	24.08 ± 4.42
Satisfied enrolments [%]	98.26 ± 0.15	**99.74 ± 0.02**	98.20 ± 0.13	98.18 ± 0.24	**99.61 ± 0.03**	**99.79 ± 0.01**	99.52 ± 0.06
Preferred time [%]	62.54 ± 1.19	65.33 ± 1.45	**98.75 ± 0.13**	62.14 ± 0.94	**95.70 ± 0.32**	**95.02 ± 0.37**	94.62 ± 0.43
Preferred room [%]	63.64 ± 2.29	62.60 ± 1.66	62.82 ± 2.07	**98.58 ± 0.29**	62.68 ± 2.23	**75.30 ± 2.30**	62.82 ± 2.07

Fig. 6.2. Solutions of the initial problem

A complete solution was found on every run of all experiments. Average values together with their RMS (root-mean-square) variances of the best achieved solutions from 10 different runs found within 30 minute time limit are presented.

The experiment marked *No preference* presents average solutions obtained without any preferences on the soft constraints. All solution quality weights W and value selection weights V are set to zero, except of the weight $V_{wconf,1}=1$ (weight of the weighted hard conflicts in the first level of the value selection).

The following five experiments marked *Students*, *Time* and *Rooms* are minimizing just one of the criteria: the student conflicts, violated time preferences and violated room preferences. *Students* experiment uses the same weights as *No preference* experiment, but student weights are the following: $V_{student,1}=0.5$, $V_{student,2}=W_{student}=1$. Similarly, *Time* experiment uses weights $V_{time,1}=0.5$, $V_{time,2}=W_{time}=1$ and *Rooms* experiment weights $V_{room,1}=0.5$, $V_{room,2}=W_{room}=1$.

The experiment marked *Students + Time* equally combines student conflicts with time preferences, weights are $V_{student,1}=V_{time,1}=0.25$, $V_{student,2}=V_{time,2}=W_{student}=W_{time}=1$.

The next experiment (marked *Students + Time + Rooms*) most closely corresponds to reality. Here all the soft preferences are considered. Student conflicts and time preferences are weighted equally, room preferences are considered much less important. Weights of student conflicts and time preferences are the same as in the previous experiment (marked *Students + Time*). Moreover, the weights on room preferences are $V_{room,2}=W_{room}=0.2$. Note that rooms are not considered in the first level of the value selection criteria.

Finally, the last experiment (marked *No CBS*) presents average solutions obtained from the solver without conflict-based statistics. The weights on soft constraints are the same as in the previous experiment. But there is $V_{conf,1}=1$ (weight of a hard conflict) instead of $V_{wconf,1}=1$ (weight of a hard conflict weighted by CBS). $V_{wconf,1}$ is set to zero. The solver was not able to find a complete solution within the given 30 minute time limit, not even when 2% random walk selection was used $P_{rw}=0.02$) to avoid cycling. Furthermore, there were at least 5 unassigned classes after 3 hours of running time.

On the Purdue Timetabling Problem, conflict-based statistics proved itself not only as a technique which can improve the solution quality, but as a technique which can help us to find a complete feasible solution.

Figure 5.3 compares several experiments giving different stress on student conflicts and time preferences. Average values from the best solutions of 10 different runs found within 30 minute time limit are presented. Only student conflicts or time preferences are considered in the border experiments marked *students* and *time* respectively. In the middle (experiment marked 1:1), student conflicts and time preferences are equally weighted. The experiment marked 3:1 prefers student conflicts three times as much as time preferences (i.e., weights of student conflicts are three times higher than weights of time preferences) and vice versa. For instance, the experiment marked 1:2 has the following weights: $V_{wconf,1}$=1, $V_{student,1}$=0.2, $V_{time,1}$=0.4, $V_{student,2}$=$W_{student}$=1, $V_{time,2}$=W_{time}=2.

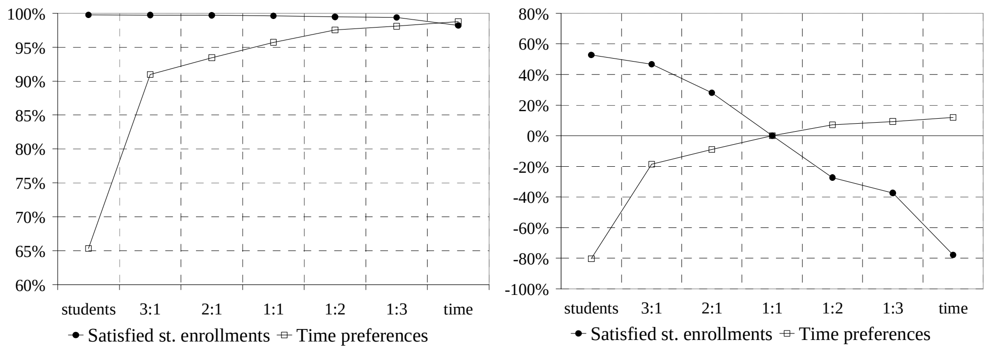


Fig. 6.3. Comparison of satisfied student enrolments and time preferences: average quality of the solution (left), improvement of the solution in terms of percentage of the 1:1 solution (right).

5.1.1. Minimal Perturbation Problem

The following experiments were conducted on one of the complete initial solutions computed in the previous set of experiments (column marked *Students + Time + Room* in Figure 5.2). Input perturbations were generated such that a given number of randomly selected variables were not allowed to retain the values they were assigned in the initial solution. Therefore, these classes can not be scheduled to the same placement as in the initial solution (either room or starting time must be different). Only variables with more than one value in their domains were used. For each number of input perturbations, ten different sets of input perturbations (i.e., variables with initial values prohibited) were generated. The following figures show the average parameter values of the best solutions found within 10 minutes.

The aim of the first set of experiments is to find a suitable setting for P_{init} (probability of selection of an initial value) and $V_{\Delta init,1..3}$ (difference in the number of assigned initial values). In each experiment, we have executed 10 tests for each of 10, 20, 30, ... 100 input perturbations respectively (100 runs in total). The average numbers of assigned variables together with the average numbers of additional perturbations are presented in Figure 5.3. One or a combination of the criteria is used in each experiment. The second column refers to the set of criteria described in Figure 5.4.

Let us explain the contents of this table. For instance, the expression $0.25_{s=1}$, $1.0_{s=2}$ in the column marked $V_{students,s}$ means that $V_{students,1}$ is set to 0.25 and $V_{students,2}$ is set to 1. The first case (Δinit=0) corresponds to the settings of the *Students + Time + Room* experiment. In remaining Δinit sets, we tried to decrease the importance of other value selection criteria in comparison with the $V_{\Delta init}$ criterion. For Δinit=1, the first level value selection criterion $V_{\Delta init,1}$ is used and the other optimisation criteria which were placed in the first level are disabled ($V_{student,1}$, $V_{time,1}$ are set to zero). And the third line Δinit=2 corresponds to a case when the second level value selection criterion $V_{\Delta init,2}$ is used and the other optimisation criteria from the second level ($V_{student,2}$, $V_{time,2}$, $V_{room,2}$) are moved to the third level.

Test case		Assigned variables [%]	Number of perturbations
P_{init}	Δinit		
0.5	0	100.00	13.83
0.6	0	99.98	13.48
0.7	0	99.96	13.33
0.8	0	99.95	12.94
0	2	100.00	31.40
0.6	2	99.99	13.26
0	1	100.00	13.70
0.6	**1**	**100.00**	**11.90**

Fig. 6.4. Comparison of several approaches to MPP.

Δinit	$V_{\Delta init,i}$	$V_{students,s}$	$V_{time,t}$	$V_{room,r}$
0	-	$0.25_{s=1}$, $1.0_{s=2}$	$0.25_{t=1}$, $1.0_{t=2}$	$0.2_{r=2}$
1	$0.5_{i=1}$	$1.0_{s=2}$	$1.0_{t=2}$	$0.2_{r=2}$
2	$1.0_{i=2}$	$0.25_{s=1}$, $1.0_{s=3}$	$0.25_{t=1}$, $1.0_{t=3}$	$0.2_{r=3}$

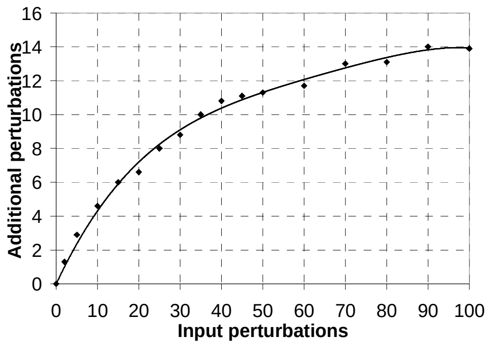
Fig. 6.5. Meaning of Δinit

Let us discuss particular experiments from Figure 5.4. In the first four experiments (marked P_{init}=0.5, ..., P_{init}=0.8), the minimal perturbation problem was solved only by changing the value selection criteria so that it selected an initial value with a given probability (50%, 60%, 70% and 80% respectively). Otherwise, it worked exactly as *Students + Time + Room* experiment, since all the other weights were the same. As the P_{init} probability is rising, we can see that the average number of additional perturbations is descending, but the algorithm is loosing the ability to find a complete solution in every run (in the given 10 minute time limit).

Similarly, we can see that using just the second level value selection criterion $V_{\Delta init,2}$ is able to find a complete solution all the time, but the average number of additional perturbations is too high. A combination with the 60% probability of an initial value selection helps to improve the average number of additional perturbations, but again, there were some cases where a complete solution was not found.

Using the first level value selection criteria $V_{\Delta init,1}$ seems to be very promising. With this criterion, we were able to find a complete solution to all the presented experiments. Moreover, the experiment marked P_{init}=0.6, Δinit=1 (combining $V_{\Delta init,1}$ with 60% initial value selection probability) gave us the best results from the above experiments, since the average number of additional perturbations was the lowest. The following results (Figures 5.6 and 5.7) were computed using the weights from this experiment.

Figure 5.6 presents the average number of additional perturbations (variables that were not assigned to their initial value though not prohibited). Additional perturbations are presented wrt. the absolute number of input perturbation (i.e., up to about 13.4% of input perturbations is considered). The best solution found within 10 minutes from each experiment is taken into account. The number of additional perturbations grows with the number of input perturbations.



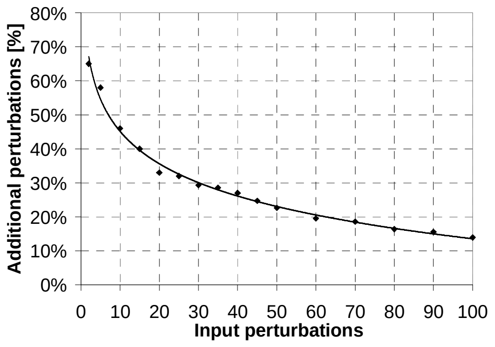
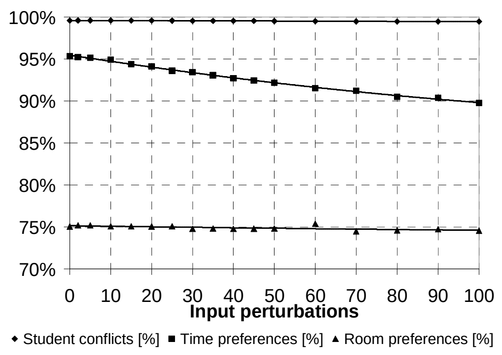


Fig. 6.6. Absolute number of average additional perturbations (left) and average additional perturbations in terms of percentage of the number of input perturbations (right).

The graph on Figure 5.7 (left) shows the average quality of the resulting solutions in the same manner as presented in Figure 5.2. Because the initial solution is (at least locally) optimal, and because the number of additional perturbations is the primary minimization criteria, it is not surprising that the quality of the solution declines with an increasing number of input perturbations. The weighting between time preferences, student conflicts, and other parameters considered in the optimisation can have a similar influence as seen in the initial solutions.



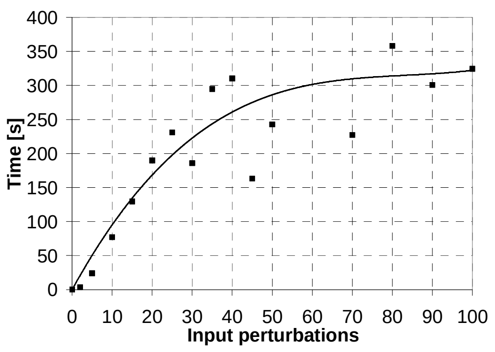


Fig. 6.7. Average solution quality (left), average time (rigt).

Finally, the graph in Figure 5.7 (right) presents the average time needed to find the best solution. Note that a 10 minutes time limit for finding the best solution was set. The influence of this limit is seen mostly on the right portion of the chart, where the number of input perturbations exceeds 50.

5.2. Summary

We have proposed and implemented a solution to a large scale university timetabling problem. Our proposal includes a new iterative forward search algorithm that is extended by conflict-based statistics which can be generalized to other search algorithms. Both ideas combined together suffice to solve the problem and the role of additional heuristics can be minimized. Our problem solver is able to construct a demand-driven timetable as well as incorporate dynamic aspects. The initial solution generated by our solver satisfies the course requests of more than 99% of students together with about 95% of time requirements. The automated search was able to find suitable times and classrooms for all classes. The experiments with a MPP give us very promising results as well. Within 10 minutes, the solver was able to find a complete, high quality solution with a small number of additional perturbations.

Moreover, the used heuristics can be tuned to maximally fulfil the user requirements, e.g., when there is a need of a trade-off between several objective functions. We have demonstrated this, for instance, in the experiment giving different stress on the satisfied student enrolments and time preferences for Purdue University timetabling problem (see figure 5.2).

6. Conclusion

In this thesis, we have presented an iterative forward search algorithm which is capable of solving various timetabling as well as general constraint satisfaction and optimisation problems. It is based on local search, but it works with partial feasible solutions, so it is capable of returning a (partial) solution any time during the search. This might be a very important feature, especially if the algorithm is used in an interactive manner. It can start from any (partial) solution and it can be used for both initial and minimal perturbation problem. We have also presented various extensions of this algorithm which can improve the quality of the returned solutions as well as applicability of the algorithm on various problems.

Also, the presented algorithm works well on the real-life large scale course timetabling problem at Purdue University. The generated solutions were very well accepted on Purdue University and they are going to use this solver in practice as of semester Spring 2006. Moreover, we are going to extend this solver to be used not only for the generation of the central timetable but also for all the departmental timetabling problems. These problems are of different structure and also there are some other constraints which need to be implemented.

The major contributions of this work are: We have defined a minimal perturbation problem. This definition is applicable on various dynamic problems where the task is to find a solution of a modified problem that is as near as possible to the solution of the original problem. Next, we have developed the iterative forward search algorithm which is capable as we believe of solving various constraint satisfaction and optimisation problems as well as minimal perturbation problems. We have also presented the conflict-based statistics which can be used in the framework of IFS or a local search algorithm and we have shown that it could dramatically improve the results especially when solving optimisation problems. Finally, we were able to solve Purdue university large lecture room timetabling problem and we are going to continue using the presented approaches for the departmental problems as well. Also, we have published four data sets (from four different semesters) of Purdue timetabling problem in a clear, anonymous form which can be used as an interesting timetabling benchmark.

7. Bibliography

BMR03 Roman Barták, Tomáš Müller, and Hana Rudová. *Minimal Perturbation Problem – A Formal View.* Neural Network World (2003), vol. 13, no. 5, p. 501-511.

BMR04 Roman Barták, Tomáš Müller, and Hana Rudová. *A new approach to modelling and solving minimal perturbation problems.* In Recent Advances in Constraints, pages 233–249. Springer Verlag LNAI 3010, 2004.

BR97 C. Bessière and J. C. Régin. *Arc consistency for general constraint networks: Preliminary results.* In Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 398–404, Nagoya, Japan, 1997.

BR01 C. Bessière and J. C. Régin. *Refining the basic constraint propagation algorithm.* In Proceedings IJCAI'01, pages 309–315, Seattle WA, 2001.

CDJD04 Hadrien Cambazard, Fabien Demazeau, Narendra Jussien, and Philippe David. *Interactively solving school timetabling problems using extensions of constraint programming.* In Edmund K. Burke and Michael Trick, editors, PATAT 2004 -Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pages 107–124, 2004.

CK96 T. B. Cooper and J.H. Kingston. *The Complexity of Timetable Construction Problems.* In the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, pp. 283-295, Springer-Verlag, 1996.

Dech03 Rina Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, 2003.

DF02 Rina Dechter and Daniel Frost. *Backjump-based backtracking for constraint satisfaction problems.* Artificial Intelligence, 136(2):147–188, 2002.

EGJ03 Abdallah Elkhyari, Christelle Guéret, and Narendra Jussien. *Solving dynamic timetabling problems as dynamic resource constrained project scheduling problems using new constraint programming tools.* In Edmund Burke and Patrick De Causmaecker, editors, Practice And Theory of Automated Timetabling, pages 39–59. Springer-Verlag LNCS 2740, 2003.

FW92 Freuder, E.C., Wallace R.J., *Partial Constraint Satisfaction*, Artificial Intelligence, 58:21-70, 1992.

GH97 Philippe Galinier and Jin-Kao Hao. *Tabu search for maximal constraint satisfaction problems.* In Proceedings 3rd International Conference on Principles and Practice of Constraint Programming, pages 196–208. Springer-Verlag LNCS 1330, 1997.

Gin93 Matthew L. Ginsberg. *Dynamic backtracking.* Journal of Artificial Intelligence Research, pages 23–46, 1993.

Ian04 Ian Miguel. *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning.* Springer, 2004.

JL02 Narendra Jussien and Olivier Lhomme. *Local search with constraint propagation and conflict-based heuristics.* Artificial Intelligence, 139(1):21–45, 2002.

Koc02 Waldemar Kocjan. *Dynamic scheduling: State of the art report.* Technical Report T2002:28, SICS, 2002.

MB01 T. Müller and R. Barták. *Interactive Timetabling.* In Proceedings of the ERCIM Workshop on Constraints, Prague, June 2001

MB02 Tomáš Müller and Roman Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results.* In Burke, Edmund; Causmaecker, Patrick De (eds.): Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT2002), Gent, 2002, pp. 58-72.

MBR04 T. Müller, R. Barták, H. Rudová. *Iterative Forward Search: Combining Local Search with Maintaining Arc Consistency and a Conflict-based Statistics.* In LSCS'04 - International Workshop on Local Search Techniques in Constraint Satisfaction, 2004.

MF00 Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics.* Springer, 2000.

MJP92 Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. *Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems.* Artificial Intelligence, 58:161–205, 1992.

MR04 Tomáš Müller and Hana Rudová. *Minimal Perturbation Problem in Course Timetabling.* In PATAT 2004 - Proceedings of the 5th international conference on the Practice And Theory of Automated Timetabling, pages 283-303, 2004.

MRB05 T. Müller, H. Rudová, R. Barták *Minimal Perturbation Problem in Course Timetabling.* Practice And Theory of Automated Timetabling, Selected Revised Papers, 2005. To appear.

Mul01 T. Müller. *Interactive Timetabling***,** Master Thesis, KTIML MFF UK, Prague, September 2001

Mul02 T. Müller. *Interactive Heuristic Search Algorithm.* In Proceedings of the CP'02 Conference - Doctoral Programme, Ithaca, September 2002. Springer-Verlag LNCS 2470, pp. 765, 2002.

PMM04 Sylvain Piechowiak, Jingxua Ma, and René Mandiau. *EDT-2004: An open interactive timetabling tool.* In Edmund K. Burke and Michael Trick, editors, PATAT 2004 - Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pages 305–321, 2004.

RM03 Hana Rudová and Keith Murray. *University course timetabling with soft constraints.* In Edmund Burke and Patrick De Causmaecker, editors, Practice And Theory of Automated Timetabling, Selected Revised Papers, pages 310–328. Springer-Verlag LNCS 2740, 2003.

RRH02	Yongping Ran, Nico Roos, and Jaap van den Herik. *Approaches to find a nearminimal change solution for dynamic CSPs.* In Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, pages 373–387, 2002.
Sch97	Andrea Schaerf. *Combining local search and look-ahead for scheduling and constraint satisfaction problems.* In Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 1254–1259, Nagoya, Japan, 1997.
SW00	Hani El Sakkout, Mark Wallace. *Probe backtrack search for minimal perturbation in dynamic scheduling.* CONSTRAINTS, 4(5):359–388, 2000.
Tsa93	E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
VJ03	Gérard Verfaillie and Narendra Jussien. *Dynamic constraint solving, 2003.* A tutorial including commented bibliography presented at CP 2003. See http://www.emn.fr/x-info/jussien/CP03tutorial/.
Wren96	A. Wren. *Scheduling, Timetabling and Rostering – A Special Relationship?* In the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, pp. 46-75, Springer-Verlag, 1996.