A New Approach to Modeling and Solving Minimal Perturbation Problems

Roman Barták[1], Tomáš Müller[1], Hana Rudová[2]

[1] Charles University, Faculty of Mathematics and Physics
Malostranské nám. 2/25, Prague, Czech Republic
{bartak, muller}@ktiml.mff.cuni.cz

[2] Masaryk University, Faculty of Informatics
Botanická 68a, Brno, Czech Republic
hanka@fi.muni.cz

Abstract. Formulation of many real-life problems evolves when the problem is being solved. For example, a change in the environment might appear after the initial problem specification and this change must be reflected in the solution. Such changes complicate usage of a traditionally static constraint satisfaction technology that requires the problem to be fully specified before the solving process starts. In this paper, we propose a new formal description of changes in the problem formulation called a minimal perturbation problem. This description focuses on the modification of the solution after a change in the problem specification. We also describe a new branch-and-bound like algorithm for solving such type of problems.

Introduction

Many real-life problems can be naturally stated as constraint satisfaction problems (CSP). In practice, the problem formulation is not static but it evolves in time [5,6,12,13,17]. In fact, it evolves even during solving the problem. Thus, to further spread up applicability of the constraint satisfaction technology in real-life applications, it is necessary to cover the dynamics of the real world. In particular, it is necessary to handle changes in the problem specification during the solving process.

The problem changes may result from the changes in the environment like broken machines, delayed flights, and other unexpected events. The users may also initiate some other changes that might specify new properties of the problem based on a (partial) solution found so far. The goal is to find a better solution for the users. 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 much from the solution found for the original problem.

There are several reasons to keep the new solution as close as possible to the existing solution. For example, if the solution has already been published like the assignment of gates to flights then it would not be convenient to change it frequently because it would confuse passengers. Moreover, the changes to the already published

solution might force other changes because the originally satisfied wishes of the users may be violated which raise an avalanche reaction.

Our work is motivated by a large scale timetabling problem at Purdue University. Once timetables are published there, they require many changes based on the additional user input. These changes should be reflected in the problem solution with a minimal impact on any previously generated solution. Thus, the primary focus of our work is to provide a support for making such changes to the generated timetable. The basic requirement is to keep the solution as close as possible to the published solution of the initial problem provided that the new solution is a solution of the altered problem. In terms of constraint satisfaction, it means that the minimal number of variable assignments is changed after the problem modification.

The paper studies the above type of dynamic problems called a minimal perturbation problem (MPP). The basic task is to find a solution of the altered problem in such a way that this new solution does not differ much from the solution of the original problem. In addition to the formal definition of the minimal perturbation problem, we also propose a branch-and-bound like algorithm to solve such problems. This algorithm can provide an approximate solution for over-constrained and hard-to-solve problems too.

The paper is organized as follows. We first give more details about the course timetabling problem that makes the main motivation of our research. Then, we survey the existing approaches to the minimal perturbation problem and to handling dynamic changes in CSP. The main part of the paper is dedicated to a formalization of the minimal perturbation problem and to a description of the solving algorithm for such a problem. We conclude with an experimental evaluation of the algorithm using random placement problems.

Motivation: A Course Timetabling Problem

The primary intent behind our work on minimal perturbation problems lies in the need to solve such a problem in the context of a real timetabling application for Purdue University (USA). The timetabling problem at Purdue University consists of allocating approximately 750 courses into 41 large lecture rooms with capacities up to 474 students. The courses are taught several times a week resulting in about 1,600 meetings to be scheduled. The space covered by all the meetings fills approximately 85% of the total available space.

There are special meeting patterns defined for each course that restrict possible time and location placement. For example, the valid combinations of days per course are given and all meetings of the same course must be taught at the same classroom and at the same time of the day. The classroom allocation must also respect the instructional requirements and the preferences of the faculty. Moreover, the instructors may have specific time requirements and preferences for each course.

The students select the courses that they wish to attend. The task is to schedule all the courses respecting given constraints and preferences while minimizing the number of potential student course conflicts for each of almost 29,000 students. The conflict appears when a student wishes to attend different courses that overlap in time.

The construction of a solution for the above problem was described in [15]. The main ideas behind the constraint model are as follows. Each meeting of a course is described using two domain variables: a time variable (starting time of the meeting in a week) and a classroom variable. The hard constraints ensure that two meetings will not be taught at the same classroom at once. The constraints also remove the values prohibited by the meeting patterns and by the requirements of instructors and faculties. The preferences on time and classroom placement together with the student course conflicts are modeled using soft constraints. The cost of a soft constraint ensuring that two courses with common students will not overlap is equal to the number of common students. The preferences of instructors and faculties are expressed using soft unary constraints.

The variable assignment is constructed using a new limited assignment number (LAN) search algorithm [18]. The LAN search algorithm is an incomplete iterative search algorithm where the standard backtracking is limited to an incomplete search of the linear complexity. The aim is to generate a partial solution with the maximal number of assigned variables. The algorithm runs in iterations where each iteration step explores some sub-tree of the search tree. The linear complexity is achieved by considering a limit on the number of assignments for each variable tried during each iteration step. The result of one iteration step is a partial assignment that is used as a guide in the next iterations. Special value and variable ordering heuristics have been proposed for this purpose.

It may still happen that the set of hard constraints is over-constrained. The user input can be used to resolve this problem by relaxing some constraints. Then, the LAN search algorithm can continue in the subsequent iterations with the problem definition changed. This approach is similar to the problem considered in this paper – we have a partial solution and we want to construct a new solution under the redefinition of the problem. However, the approach of the LAN search algorithm does not attempt to minimize the number of changes in the subsequent solutions. It rather maximizes the number of assigned variables. A solver for minimal perturbation problems should do both tasks.

The solver of the above timetabling problem was implemented using the `clpfd` library of SICStus Prolog [2] with the help of our soft constraints solver [16]. The initial implementation started with built-in backtracking of SICStus Prolog. However, the experiments with standard backtracking did not lead to a solution after 10 hours of CPU time because too many failed computations were repeated exploring the parts of the search tree with no solution. The LAN search algorithm was able to substantially improve on the initial partial solution. Starting from 33 courses, only one course remained unassigned after eight iterations of the LAN search algorithm. Assignment of this course was successfully completed with the help of the user.

The assignment generated by the LAN search algorithm introduces an initial solution of the timetabling problem. Once this solution is published, it requires many changes based on the additional input. These changes should be involved in the problem solution with a minimal impact on any previously generated solution. Note also, that it is not possible to limit the changes in the problem definition. Faculties and instructors may come with completely new courses to be scheduled and the requirements towards the original courses may change substantially. Moreover, some requirements or classes may be canceled which can help to find a more acceptable

solution. On the other hand, the new requirements usually make the problem harder since we are constrained in the number of allowed changes with respect to the original solution. The original problem, its solution, and the set of requested changes introduce the input for the minimal perturbation problem we are going to solve.

The LAN search algorithm represents the very first step in the direction towards solving the minimal perturbation problem because the partial solution generated in each iteration step is derived from the previous solution. As our experiments showed, the distance of these solutions is mostly acceptable if a small number of changes is introduced to the problem. However, the distance significantly enlarges when the number of problem changes increases. Other experiments also showed that a more informed heuristics may improve the quality of the solution substantially. Thus, the LAN Search seems to be a good base for solving the minimal perturbation problem.

Related Works

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" [7]. Therefore, handling dynamic changes of the problem formulation was studied in the constraint community for some time. Dechter and Dechter [4] proposed a notion of *Dynamic Constraint Satisfaction Problem* (DCSP) that is a sequence of CSPs, where every CSP is a result of changes in the preceding one. A difference between two consecutive CSPs is expressed by a set C_{add} of added constraints and a set C_{del} of constraints deleted from the problem.

There exist many criteria for evaluating the performance and quality of algorithms solving DCSP, like efficiency in finding a new solution or solution stability in the face of problem alternation. A minimal perturbation problem can occur here as a criterion of similarity or consistency of the solution – the smallest number of values should be different between any new and original solution.

Several algorithms have been proposed to search for a minimum change in the solution. For example, Ran, Roos, and Herik [14] proposed an algorithm for finding a near-minimal change solution that is a solution with the minimal or near minimal number of modified value assignments. Their algorithm is looking for a new solution by changing the assignment of one, two, tree variables, and so on until a feasible assignment is found – an approach similar to Limited Discrepancy Search [10].

A *minimal perturbation problem* was described formally by El Sakkout, Richards, and Wallace in [6] as a 5-tuple $(\Theta, \alpha, C_{add}, C_{del}, \delta)$, where Θ is a CSP, α is a solution to Θ, C_{add} and C_{del} are constraint removal and addition sets, and δ is a function that measures the distance between two complete assignments. Their solving method combining linear and constraint programming is looking for a complete assignment β that minimizes $\delta(\alpha,\beta)$ and that is a solution of the new problem arising from Θ by adding the constraints from C_{add} and deleting the constraints from C_{del}.

It is interesting to see that there is a lack of works concerned by search for a minimum change [14,7]. A commented bibliography from [17] refers only to four papers including an earlier version of this paper.

Looking to existing methods to handle dynamic problems, there are two main differences in our approach. First, we allow arbitrary changes of the problem formulation including addition and deletion of the variables. Note that variable deletion can be modeled in DCSP as a deletion of all the constraints containing the variable. Variable addition cannot be modeled directly in DCSP. Second, we are working with incomplete assignments to solve over-constrained and hard-to-solve problems. Other methods usually relax constraints in advance to solve such problems.

A Formal Model

In this section, we present a new formal model of the minimal perturbation problem (MPP) 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 solution of the original problem. We first survey the standard definitions of CSP and we introduce a new notion of a maximal consistent assignment. This new notion will help us to describe formally an approximate solution of the constraint satisfaction problem. In the second part, we define a minimal perturbation problem and a solution of the minimal perturbation problem.

Preliminaries

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, where 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, it does not exist any complete assignment satisfying all the constraints in over-constrained problems. Therefore other definitions of the problem solution, for example Partial Constraint Satisfaction [8], were introduced.

In this paper, we propose a new view of the problem solution based on a new notion of a maximal consistent assignment. This approach is strongly motivated by the course timetabling problem but we believe that it is generally applicable. The basic idea is to assign as many variables as possible while keeping the problem consistent. Then, the user may 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 partial assignment can be extended to other variables.

Formally, let Θ be a CSP and C be a consistency technique, for example arc consistency. We say that the *constraint satisfaction problem* is *C-consistent* if the consistency technique C deduces no conflict. For example, for arc consistency the conflict is indicated by emptying a domain of some variable. We denote C(Θ) the result of the consistency test which could be either *true*, if the problem Θ is C-consistent, or *false* otherwise. Let Θ be a CSP and σ be an assignment of some variables from Θ. Then, we denote Θσ an application of the assignment σ to the problem Θ. Θσ is a problem derived from Θ such that the domains of the variables from σ are reduced to a value defined by σ. Finally, we say that an *assignment s* is *C-consistent* with respect to some consistency technique C if and only if C(Θσ) is true. Note that a complete C-consistent assignment is a solution of the problem provided that the consistency technique C is able to check satisfaction of the constraints when the values of the variables are known. Backtracking-based solving techniques are typically extending a partial consistent assignment towards a complete consistent assignment where all the variables are assigned.

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 a solution defined using the notion of a maximal consistent assignment. We say that a *C-consistent assignment is maximal* for a given CSP if there is no C-consistent assignment with a larger number of assigned variables. We can also define a weaker notion of a so called *locally maximal C-consistent assignment.* A locally maximal C-consistent assignment is a C-consistent assignment that cannot be extended to another variable.

Notice the difference between the above two notions. The maximal C-consistent assignment has a global meaning because it is defined using the cardinality of the assignment that is the number of assigned variables is maximized there. The locally maximal C-consistent assignment is defined using a subset relation – it is not possible to extend the locally maximal C-consistent assignment by assigning any additional variable without getting inconsistency. Visibly, the maximal C-consistent assignment is the largest (using cardinality) locally maximal C-consistent assignment.

Example (maximal arc-consistent assignments):

Let V = {a,b,c,d,e} be a set of variables with domains D = {D_a={1,2}, D_b={1,2,3}, D_c={2,3}, D_d={2,3}, D_e={2,3}} and C = {a≠b, b≠c, c≠d, c≠e, d≠e} be a set of constraints. Assume that we use arc consistency as the technique for checking consistency of CSP Θ = (V,D,C). Then:

- σ = {a/1} is a locally maximal arc-consistent assignment for Θ which is not a maximal arc-consistent assignment (|σ|=1),
- γ = {a/2, b/1} is a maximal arc-consistent assignment for Θ (|γ|=2).

If a constraint satisfaction problem has a solution then any maximal C-consistent assignment is the solution provided that the consistency technique C is able to check satisfaction of the constraints when the values of the variables are known. If a constraint satisfaction problem has no solution – it is an over-constrained problem – then some maximal C-consistent assignment still exists. We propose to define the *solution of the (over-constrained) problem* as a maximal C-consistent assignment.

There is a strong real-life motivation for the above view of the solution for over-constrained problems. In scheduling and timetabling applications [13,15], the maximal C-consistent assignment usually corresponds to allocation of the largest number of activities to resources. Such an assignment is more informative than the answer "no" that indicates non-existence of a complete assignment of variables satisfying the constraints. Moreover, if the problem solution is defined as a maximal C-consistent assignment then some solution always exists. In particular, it is not necessary to know in advance that the problem is over-constrained. So, a maximal C-consistent assignment is as a generalization of the traditional solution. It covers a solution of CSP as well as a solution of over-constrained CSP.

Because the maximal C-consistent assignment is the largest among the locally maximal C-consistent assignments, a locally maximal C-consistent assignment can be seen as an *approximate solution*. The largest the assignment is, the better approximate solution we have. The constraint solver may return an approximate solution when the problem is hard-to-solve. In particular, the solver may return the largest locally maximal C-consistent that is possible to compute using given resources (e.g., time).

Notice finally that the solution is parameterized by a consistency technique C which gives users the flexibility to define the desired features of the solution. The consistency technique may check only validity of the constraints between the assigned variables. Then we get the assignment with the largest number of instantiated variables. This assignment cannot be extended to another variable without getting a constraint violation. If a stronger consistency technique is used, for example arc consistency, we may expect a shorter assignment as the solution. However, this assignment can be extended to another variable without getting a constraint violation. Instead, we get a failure of the arc-consistency test. In some sense, a solution defined using a stronger consistency level gives the user some advance for future extensions of the assignment.

A Minimal Perturbation Problem

Now we can formally define a *minimal perturbation problem* (MPP) as a triple $\Pi = (\Theta, \alpha, \delta)$, where:

- Θ is a CSP,
- α is a possibly partial assignment for Θ called an *initial assignment*,
- δ is a *distance function* defining a distance between any two assignments.

A *solution to the minimal perturbation problem* $\Pi = (\Theta, \alpha, \delta)$ is a solution β for Θ such that $\delta(\alpha,\beta)$ is minimal. Recall, that we define the solution as a maximal C-consistent assignment for some consistency technique C. The idea behind the solution of MPP is apparent – the task is to find the largest possible assignment of variables for the problem Θ in such a way that it differs minimally from the initial assignment.

Recall that the minimal perturbation problem should formalize handling changes in the problem formulation. So, one may ask where the original problem is in the above definition. Because the new solution is compared only to the solution of the original problem, it is not necessary to include neither the original problem nor the description of the problem change in the definition of MPP. This gives us the freedom to change

the problem arbitrarily, in particular to add and remove variables and constraints and to change variables' domains. The solution of the original problem can be mapped to an initial assignment in Π in the following way. Assume that an assignment σ is a solution of the original problem. Then the initial assignment α in the definition of MPP is defined in the following way:

$$\alpha = \{v/h \mid v/h \in \sigma \ \& \ v \in \Theta\}^1.$$

Note that α and σ are not necessarily identical because some variables may be removed from the problem. Thus, $\alpha \subseteq \sigma$ holds.

The distance function δ in the definition of MPP is specified by the user. For purposes of our timetabling problem, we use a specific distance function describing the number of differences between two assignments. Let σ and γ be two assignments for Θ. Then we define $W(\sigma,\gamma)$ as a set of variables v such that the assignment of v in σ is different from the assignment of v in γ:

$$W(\sigma,\gamma) = \{v \in \Theta \mid v/h \in \sigma \ \& \ v/h' \in \gamma \ \& \ h \neq h'\}.$$

We call $W(\sigma,\gamma)$ a *distance set* for σ and γ and the elements of the set are called *perturbations*. The distance function is then defined in the following way:

$$\delta(\sigma,\gamma) = |W(\sigma,\gamma)|.$$

If a metric is defined on the variables' domains then it is possible to specify other distance functions, for example:

$$\delta(\sigma,\gamma) = \max_v \{dist_v(h,h') \mid v/h \in \sigma \ \& \ v/h' \in \gamma\}, \text{ or}$$
$$\delta(\sigma,\gamma) = \sum_v \{dist_v(h,h') \mid v/h \in \sigma \ \& \ v/h' \in \gamma\}, \text{ or}$$
$$\delta(\sigma,\gamma) = \left(\sum_v \{dist_v^2(h,h') \mid v/h \in \sigma \ \& \ v/h' \in \gamma\}\right)^{1/2},$$

where $dist_v$ is a distance function (metric) on the domain of the variable v.

Notice that the above formulation of MPP generalizes the formulation from [6] by working with partial assignments rather than with complete assignments and by allowing arbitrary changes to the problem. Also, we reformulated the definition from [1] to be easier and more general while preserving the original meaning that is minimizing the number of changes in the solution after a problem change

Example:

Let α={b/3} be an initial assignment for a CSP Θ with variables {b,c,d}, domains {D_b={1,3}, D_c={1,2,3}, D_d={2,3}}, and constraints {b≠c, c≠d, d≠b}. Then the problem Θ has the following solutions (maximal arc-consistent assignments):

- β_1 = {b/1,c/2,d/3} ($W(\alpha,\beta_1)$ = {b}),
- β_2 = {b/1,c/3,d/2} ($W(\alpha,\beta_2)$ = {b}),
- β_3 = {b/3,c/1,d/2} ($W(\alpha,\beta_3)$ = {}),

but only the solution β_3 is a solution of MPP $\Pi = (\Theta, \alpha, |W|)$.

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

MPP Solver

A minimal perturbation problem is a type of optimization problem so it is natural to use optimization technology to solve it. In particular, we have modified a branch-and-bound algorithm for this purpose.

There is one specialty of MPP going beyond conventional objective functions, in particular the maximization of the number of assigned variables. Handling incomplete assignments is important for solving over-constrained problems. It also helps to produce an approximate solution for hard-to-solve problems. The proposed algorithm explores locally maximal consistent assignments and it keeps the largest assignment with a minimal distance from the initial assignment (row 4). We have developed a concept of variable locking to extend any partial consistent assignment to a locally maximal consistent assignment.

The algorithm is also requested to run in an interactive environment that is the solution must be produced in a reasonable time. Often a complete search space is too large. Therefore, we combine the optimization branch-and-bound algorithm with principles of the LAN search algorithm [18]. In particular, we propose a concept of variable expiration that decreases the size of the search space. Consequently, the algorithm finds an approximate solution as defined in the previous section.

```
   label(Variables,LockedVariables)
1     if validate_bound(Variables, Locked Variables) then
2        V <- select_variable(Variables, LockedVariables)
3        if V=nil then
4           save_best_solution(Variables)
5        else
6           Value <- nil
7           while Value <- select_value(V,Value) & non_expired(V) do
8              label(Variables,LockedVariables) under V=Value
9           end while
10          label(Variables,[V|LockedVariables])
11       end if
12    end if
   end label

   solve(Variables)
      label(Variables,[])
      return saved_best_solution
   end solve
```

Fig. 1. A labeling procedure for solving the minimal perturbation problem.

Figure 1 shows a skeleton of the proposed algorithm that is basically a branch-and-bound algorithm. The algorithm labels variables – assign values to the variables – until a solution is found. First, the algorithm checks whether the current bound is better than the bound of the so far best assignment, if any (row 1). In case of success, the algorithm continues by extending the current partial assignment (rows 2-11). Otherwise, search in this branch stops because the current partial assignment cannot be extended to the best assignment. In row 2, the algorithm selects a variable V to be labeled. If the variable selection fails (row 3), it means there is no variable to be assigned. Thus, the current solution is stored as the best one (row 4) and the current

branch is closed. In case of successful variable selection (rows 5-10), the values in the domain of V are tried to be assigned to V (rows 7-8). The algorithm is called recursively in an attempt to assign the rest of the variables (row 8). If it is not possible to select any value for the variable V then the variable is locked and the algorithm continues in labeling the remaining variables (row 10).

In the next sections, we will give details on variable locking and variable expiration, we will describe how to estimate the bound for the optimization algorithm (`validate_bound`), and we will describe the value (`select_value`) and variable (`select_variable`) ordering heuristics used in our solver.

Locked and Expired Variables for Partial Assignments

We have formulated a minimal perturbation problem in such a way that an incomplete assignment could be a solution to the problem. Traditionally, the depth-first search algorithms backtrack to the last assigned variable when all attempts to assign a value to the variable failed. Then they try to find another value for it. Notice that we do not get a locally maximal consistent assignment in such a case because it could still be possible to extend the partial assignment to another non-assigned variable. Therefore, we introduce here the concept of *variable locking*. The variable whose assignment failed is locked and the search algorithm proceeds to the remaining non-assigned variables (row 10). The locked variables still participate in constraint propagation so the above mechanism extends any partial consistent assignment of variables to locally maximal consistent assignment. Notice also that the locking mechanism is local – the variable is locked only in a given search sub-tree. As soon as the algorithm backtracks above the level, where the variable has been locked, the variable is unlocked and it can participate in labeling again.

Recall that our original motivation was to support interactive changes in the problem. It means that the solver returns a solution quickly after any change. Exploring a complete search space of the problem could be hard and a time consuming task. We propose to explore just a part of the search space by applying techniques of LAN Search [18]. The basic idea of LAN (Limited Assignment Number) Search is to restrict the number of attempts to assign a value to the variable by a so called *LAN limit*. This number is maintained separately for each variable, which differentiates LAN Search from other incomplete tree search techniques like Credit Search [3] and Bounded Backtrack Search [9].

The LAN principle is realized by using a counter for each variable. The counter is initialized by the LAN limit. After each assignment of a value to the variable, this counter is decreased. Note that it is possible to assign the same value to the variable in different branches of the search tree and each such attempt is reflected in the counter. When the counter is zero, the variable reached the allowed number of assignments – we say that the *variable expired*.

Let us assume that the time complexity of the search algorithm is defined as a number of attempts to assign a value to a variable. Because different combinations of values should be explored, the worst-case time complexity of the complete search algorithms is exponential expressed by the formula $domain\_size^{number\_of\_variables}$. The LAN limit restricts the number of attempts to assign a value to each variable in the

search tree. Thus, the total number of attempts to assign a value to the variables is *lan_limit*number_of_variables* that is we get a linear worst-case time complexity of the LAN search algorithm. Consequently, the LAN search algorithm does not explore all possible combinations of values and it produces approximate solutions only.

Locked and expired variables do not participate in labeling. Thus, these variables are skipped when selecting a variable to be labeled (`select_variable`), even if they are not assigned yet. Moreover, the variable may expire when already selected for labeling. So, the expiration must be checked during value selection as well (row 7).

Computing Bounds for Optimization

As we mentioned above a minimal perturbation problem is a sort of constraint optimization problems. There are two objective criteria in MPP, namely maximizing the number of assigned variables (called *length*) and minimizing the distance from the initial assignment. Our solver uses the distance function defined as a number of perturbations so the proposed algorithm is looking for the largest consistent assignment and it prefers the assignment with a smaller number of perturbations among the assignments of the same length. Thus the objective function can be described as a lexicographic ordering [maximize length, minimize perturbations].

The core labeling procedure explores locally maximal consistent assignments. When it finds such an assignment, it saves it as a bound (row 4) that is used to prune the rest of the search tree. The pruning is done by checking whether the current partial assignment can be better than the so-far best saved assignment (row 1). To do this comparison, we estimate the maximal length of the assignment and the minimal number of perturbations. In particular, we need an upper estimate of the maximal length and a lower estimate of the number of perturbations. Computing the estimates is included in the function `validate_bound`.

The estimate of the maximal length is simply the number of all variables minus the number of locked variables. Recall, that the algorithm did not succeed to assign a value to the locked variable. Thus, this variable will not participate in a locally maximal consistent assignment. On the other hand, the expired variables may still be part of a locally maximal consistent assignment because the value can be selected for them via constraint propagation.

We estimate the minimal number of perturbations by counting the variables where the initial value is outside their current domain. The initial value for the variable is taken from the initial assignment. If no such value exists – no value is assigned to the variable in the initial assignment – then the variable is always assumed as a perturbation. This is realized by using a dummy initial value for v outside the domain D_v. Note, that there is a constant number of such variables, say K, in the problem so minimizing |W| – the size of the distance set – is equivalent to minimizing |W|+K.

Value and Variable Ordering Heuristics

The proposed algorithm can use existing variable ordering heuristics encoded in the procedure `select_variable`. Only the variables that are neither locked nor expired and that are non-assigned yet can be selected for labeling. We use a standard *first-fail principle* to select among them – the variable with the smallest domain is selected.

The value ordering heuristics (`select_value`) should reflect the goal of search – finding an assignment minimizing the number of perturbations. Therefore, for each value we compute the number of perturbations that will appear in the solution if the value is selected – *added perturbations*. The values with a smaller number of added perturbations are preferred. Computing the number of added perturbations can be done by a problem-specific procedure (like in our tests) or by a problem independent method like singleton consistency (the value is assigned, propagated through the constraints, and the added perturbations are counted).

Experimental results

We have implemented the proposed labeling procedure in `clpfd` library [2] of SICStus Prolog version 3.10.1. For comparison, we have also implemented a local search procedure in Java, JDK 1.3.1. All presented results were accomplished under Linux on one processor of a PC with a Dual Pentium IV Xeon 2.4 GHz processor with 1 GB of memory. Algorithms were tested using a benchmark problem, called a random placement problem, derived from timetabling problems.

Random Placement Problems

The *random placement problem* (*RPP*) (see http://www.fi.muni.cz/~hanka/rpp/) seeks to place a set of randomly generated rectangles – *objects* – of different sizes into a larger rectangle – *placement area* – in such a way that no two objects overlap and all objects' borders are parallel to the border of the placement area. In addition, a set of allowable placements can be randomly generated for each object. The ratio between the size of the placement area and the total area of all objects is denoted as the *filled area ratio*.

RPP allows us to generate various instances of the problem similar to a trivial timetabling problem. The correspondence is as follows: the object corresponds to a course to be timetabled – the x-coordinate to its time, the y-coordinate to its classroom. For example, a course taking three hours corresponds to an object with dimensions 3×1 (the course should be taught in one classroom only). Each course can be placed only in a classroom of sufficient capacity – we can expect that the classrooms are ordered increasingly in their size so each object will have a lower bound on its y-coordinate.

RPP is modeled as a CSP. Each object is represented by a pair of variables: x-coordinate (*VarX*) and y-coordinate (*VarY*) in the placement area. Each variable is given a domain according to the size of the placement area and in case of y-coordinate

also according to its lower bound. The global constraint `disjoint2` [2] over x-coordinates and y-coordinates ensures that the objects will not overlap.

Each object is also represented by a redundant *time-space variable* defined by the constraint *VarXY #= VarX + VarY * SizeX*, where *SizeX* is the size of the placement area in the x-coordinate. Note that there is a one to one mapping between the value of *VarXY* and the pair of values for *VarX* and *VarY* defining it (see Figure 2).

VarY					
	2	**8**	**9**	**10**	**11**
	1	**4**	**5**	**6**	**7**
	0	**0**	**1**	**2**	**3**
		0	**1**	**2**	**3**
			VarX		

Fig. 2. A value of the time-space variable *VarXY* (in the matrix) is defined uniquely by the positions *VarX* and *VarY* via the formula *VarXY = VarX + VarY * SizeX*, where *SizeX*=4.

We can post the global constraint `all_distinct` over the *VarXY* variables which has a similar role like the `disjoint2` constraint. Using both constraints together brings a higher degree of propagation. However, the main reason for introducing the time-space variables is that they will participate in the labeling procedure. It has the advantage of exact placement of the object by assigning a value to its time-space variable.

Let us now describe how we can generate the changed problem and how we evaluate a solution of the MPP. First, we compute the initial solution using the LAN search algorithm [18]. The changed problem differs from the initial problem by *input perturbations*. An input perturbation means that both x-coordinate and y-coordinate of a rectangle must differ from the initial values, i.e. *VarX#\=InitialX*, *VarY#\=InitialY*. For a single initial problem and for a given number of input perturbations, we can randomly generate various changed problems. In particular, for a given number of input perturbations, we randomly select a set of objects which should have input perturbations. The solution to MPP 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.

Experiments

We have completed a number of small experiments on RPPs during implementation of the algorithm. RPPs were very helpful as they are simple enough to help in understanding behavior of the algorithm. On the other hand, one can generate much larger RPPs, which allow performing more complex experiments.

The first set of experiments shows performance of the algorithm for different sizes of the random placement problem. We have generated three data sets with 100, 200, and 300 objects, respectively. The objects were generated randomly in the sizes 2×1 to 6×1. Each object has assigned a random lower bound for the y-coordinate which was in average 35% of the y-size of the placement area. These parameters correspond to a real-life course timetabling problem that we are going to solve in the future. Each data set contained fifty problems and the filled area ratio was 80%. The initial solution was computed by the LAN search algorithm. Then we added changes to the

solution by forbidding the current position of randomly selected objects. Recall that these changes are called input perturbations. The number of input perturbations was chosen in the range from 0 to 25% of the number of objects. For each problem and for each considered number of the input perturbations, five minimal perturbation problems were generated. During the experiments, the algorithm used a LAN limit equal to 5. We also performed experiments with the limit 10 and 100 but no improvement has been achieved.

Figure 3 shows the relative number of additional perturbations, the relative number of assigned variables, and the CPU time as a function of the number of input perturbations. We use relative numbers of perturbations with respect to the number of objects to compare the results for data sets with a different number of objects.

The top left graph of Figure 3 shows an increase of the number of additional perturbations with the increasing number of input perturbations. This behavior is natural – more changes in the problem definition evoke more additional changes in the solution. We can also see that the number of additional perturbations decreases with the size of the problem which may seem surprising at first. However, recall that the size of the objects remains the same while the size of the area is increasing with the larger number of objects (to keep the filled area ratio 80%). Note that it is easier to place 80 objects into an area of size 100 than to place 8 objects into an area of size 10 which explains the above behavior.

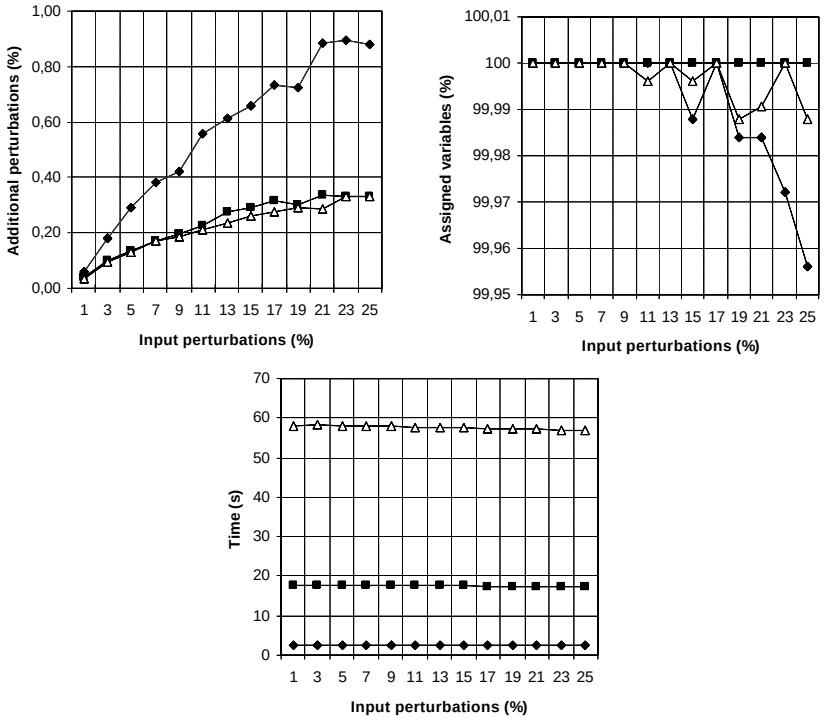

Fig. 3. Performance and quality of the solution for the proposed algorithm on random placement problems with 100 objects (?), 200 objects (¦), and 300 objects (△).

The top right graph of Figure 3 shows a number of assigned variables which is rather large. Actually, all variables were assigned in most of the experiments which is a very promising result. The reason could be that the filled area ratio is 80% which makes the problems under-constrained. Due to time reasons we were not able to perform experiments for over-constrained problems yet.

The bottom graph of Figure 3 shows a non-surprising increase of the running time with larger problems. The increase is non-linear which may seem contradicting with the linear search space explored by the algorithm. However, it is necessary to include the complexity of value ordering heuristics in the total complexity. The complexity of heuristics depends on the number of objects as well as on the number of positions where the objects can be placed.

In the second experiment, we have compared the proposed extension of the branch-and-bound algorithm with a local search algorithm. This local search algorithm is an extended version of the algorithm presented at [13]. It is heuristically oriented to solving RPPs and it was implemented in Java. As before, we used fifty initial problems and five MPPs per an initial problem. We compared the algorithms on the problems consisting of 100 objects and we did the experiments for input perturbations from 0 to 100 with the step 4. Thus 0 to 100% relative input perturbations are covered.

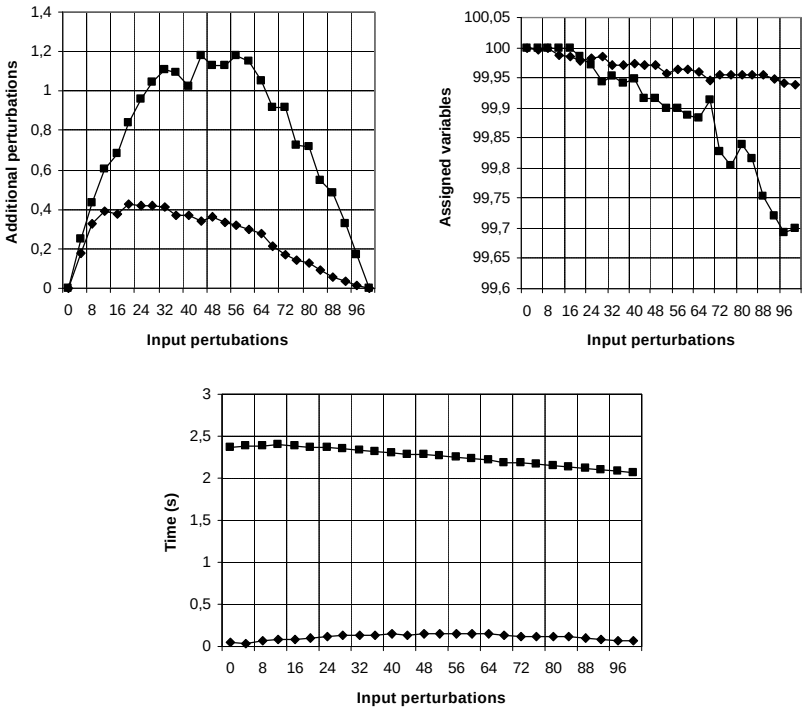

Fig. 4. Comparison of the proposed branch-and-bound algorithm (¦) with a problem specific local-search solver (?).

Figure 4 shows the number of additional perturbations, the number of assigned variables, and the CPU time as a function of the number of input perturbations for both algorithms. We expect the comparison to be the most meaningful for a smaller number of input perturbations (up to about 25%) because this is the area of the highest interest for MPPs. Here the general branch-and-bound algorithm seems to be comparable to the specific problem solver in terms of the solution quality. Still, the dedicated local search algorithm is much faster there.

Conclusions

Including dynamic features to solvers is a recent trend in planning, scheduling, and timetabling. In this paper, we addressed one of these features, namely finding a solution of changed problem that does not differ much from the solution of the original problem – a so called minimal perturbation problem. We proposed a new view of the minimal perturbation problem in the context of constraint satisfaction motivated by a real-life application of university course timetabling. This new framework supports incomplete assignments as a problem solution which is useful to model over-constrained problems as well as hard-to-solve problems. Moreover, this new formulation allows looser changes of the problem formulation like addition and retraction of constraints and variables as well as changes in the variables' domains. We also proposed a new labeling algorithm to solve minimal perturbation problems. This incomplete algorithm has a linear time complexity and it is capable to solve over-constrained and hard problems by looking for locally maximal consistent assignments. The experiments using random placement problems showed that the algorithm is able to find optimal or close to optimal solutions. Next work will concentrate on testing the algorithm in a real-life timetabling application and on improving its real-time performance. We will also explore the possibility to extend the tested local search algorithm for general minimal perturbation problems.

Acknowledgements

This work is supported by the Grant Agency of Czech Republic under the contract No. 201/01/0942 and by Purdue University. Our thanks go to the Supercomputer Center Brno where experiments were conducted. Last but not least we would like to thank to Vlastimil Krápek who implemented an automated tester needed for experimental evaluation.

References

1. Roman Barták, Tomáš Müller, and Hana Rudová. *Minimal Perturbation Problem – A Formal View*. Neural Network World 13(5): 501-511, 2003.
2. Mats Carlsson, Greger Ottosson, and Bjorn Carlson. *An open-ended finite domain constraint solver*. In Programming Languages: Implementations, Logics, and Programming. Springer-Verlag LNCS 1292, 1997.
3. Andrew M. Cheadle, Warwick Harvey, Andrew J. Sadler, Joachim Schimpf, Kish Shen and Mark G. Wallace. *ECLiPSe: An Introduction*. IC-Parc, Imperial College London, Technical Report IC-Parc-03-1, 2003.
4. Rina Dechter and Avi Dechter. *Belief maintenance in dynamic constraint networks.* In Proceedings of AAAI-88, pp. 37-42, 1998.
5. Brian Drabble, Najam-ul Haq. *Dynamic Schedule Management: Lessons from the Air Campaign Planning Domain*. In Pre-proceedings of the Sixth European Conference on Planning (ECP-01), pp. 193-204, 2001.
6. Hani El Sakkout, Thomas Richards, and Mark Wallace. *Minimal Perturbation in Dynamic Scheduling*. In Henry Prade (editor): 13th European Conference on Artificial Intelligence. ECAI-98. John Wiley & Sons, 1998.
7. Hani El Sakkout and Mark Wallace. *Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling*. Constraints 4(5): 359-388. Kluwer Academic Publishers, 2000.
8. Eugene C. Freuder and Richard J. Wallace. *Partial Constraint Satisfaction*. Artificial Intelligence, 58:21-70, 1992.
9. William D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Stanford University, 1995.
10. William D. Harvey and Matthew L. Ginsberg. *Limited discrepancy search*. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 607-615, 1995.
11. Narendra Jussien and Olivier Lhomme. *Local search with constraint propagation and conflict-based heuristics.* Artificial Intelligence, 139(1):21-45, 2002.
12. Waldemar Kocjan. *Dynamic scheduling: State of the art report*, SICS Technical Report T2002:28, ISSN 100-3154, 2002.
13. Tomáš Müller and Roman Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results.* In Edmund Burke and Patrick De Causmaecker (eds.): Proceedings of PATAT2002 , Gent, pp. 58-72, 2002.
14. Yongping Ran, Nico Roos, Jaap van den Herik. *Approaches to find a near-minimal change solution for Dynamic CSPs*. Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, pp 373-387, 2002.
15. Hana Rudová and Keith Murray. *University Course Timetabling with Soft Constraints*. In Edmund Burke and Patrick De Causmaecker (eds.): Practice And Theory of Automated Timetabling IV. Springer-Verlag LNCS 2740, pp. 310-328, 2003.
16. Hana Rudová, *Soft CLP(FD)*. In Susan Haller and Ingrid Russell (eds.): Proceedings of the Sixteenth International Florida Artificial Intelligence Symposium, FLAIRS-03, AAAI Press, pp. 202-206, 2003.
17. Gérard Verfaillie, Narendra Jussien. *Dynamic Constraint Solving*. A tutorial including commented bibliography presented at CP 2003, Kinsale.
18. Kamil Vermirovský and Hana Rudová, *Limited Assignment Number Search Algorithm.* In Maria Bielikova (ed.): SOFSEM 2002 Student Research Forum, pp. 53-58, 2002.
19. Stefan Voß. *Meta-heuristics: State of the art.* In Alexander Nareyek (ed.): Local search for planning and scheduling: revisited papers. Springer-Verlag LNAI 2148, pp. 1-23, 2001.