A New Labeling Algorithm to Solve Minimal Perturbation Problems: A Preliminary Report

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. Solving real-life planning, scheduling, and timetabling problems is usually an iterative process in which, after seeing the generated solution, users may change the problem constraints. This change requires producing a new solution which satisfies these constraints but not being too far from the original solution. This type of problem is called a minimal perturbation problem. The paper formally describes a minimal perturbation problem in the context of constraint satisfaction and it proposes a new depth-first search algorithm for solving a particular instance of the minimal perturbation problem.

Introduction

Solving real-life planning, scheduling, and timetabling problems often requires making changes in the produced solution [5,6,8,14,19]. Then, the automated solver should be capable of integrating these changes into a new solution. There are several reasons for introducing such changes. For example, if the original environment was changed so the computed solution can not be applied then we would like to obtain a new solution based on the changes introduced. A typical example is a gate allocation problem in which the system must be able to react on-line to flight delays and to other unexpected events. It could also be applied where ever there are certain objectives that are impossible to formulate formally in advance. By proposing a change to the solution, the user may describe a preferred solution without specifying the objectives formally. For instance, in making a school timetable, an administrator may be able to specify the individual preferences of professors by re-allocating their lectures [14,17]. Then the timetabling system should re-allocate other lectures in such a way that the problem constraints are met. Last but not least, by proposing changes to a partial solution the user may help the solver to recover from a dead-end. Thus, this user guidance may help to solve problems that would be hard for a fully automated solver.

A typical expectation from such interactive software is that after making a change to the problem formulation, a new solution will be derived from the existing (initial) solution, rather than producing a completely new solution from scratch [4,16,20]. As

a result, the number of differences between the initial solution and the new solution can be kept as small as possible while still respecting all the constraints that the solution must satisfy. Again, there are several reasons to keep the new solution as close as possible to the initial solution. First of all, it is easier for the users to track the solving process if they can observe the additional changes that the system needed to make in order to react to the user changes. Second, if the initial solution has already been published, such as the assignment of gates to flights, then one should avoid too frequent changes, which may just end up confusing passengers. It is also possible that the changes to this published solution would bring about other user changes (because originally satisfied wishes of these users may be violated) and thereby raise the specter of a cumulative "avalanche reaction". Altogether, it is desirable that the problem solving process should continue as smoothly as possible after any proposed change to the initial solution [6,16,20].

Our work has been motivated by a large scale timetabling problem at Purdue University, USA where the primary focus was to provide a support for making changes to a generated timetable. Once timetables are published they require many changes based on additional user input. These changes should be integrated into the problem solution with a minimal impact on any previously generated solution. Since we are already able to generate the solution for the initial problem [17], our current work consists of solving the new problem with some additional requirements. Here again, the basic need is to keep the new solution as close as possible to the published solution of the initial problem – given that the new solution is a solution of the altered problem.

The problem of finding a new solution after making changes to the problem formulation has been addressed by a number of researchers [4-8,13,14,16,19,20]. Probably the closest approach to our problem area is a *minimal perturbation problem* introduced by El Sakkout, Richards, and Wallace [7]. They formally defined the minimal perturbation problem (in the context of constraint satisfaction) as a constraint satisfaction problem (CSP) Θ together with its (initial) solution α and two sets of added C_{add} and removed C_{del} constraints [6]. The task is to find a solution of the new constraint satisfaction problem that arises from the original CSP Θ by adding the constraints from C_{add} and removing the constraints from C_{del} while still remaining as close as possible to the initial solution. The distance between the two solutions (complete assignments of the variables) is measured via a user defined function δ.

In [1], we proposed a new definition of MPP, working with arbitrary changes to the CSP (like adding/removing variables and constraints and changing variables' domains). The main difference from the original formulation is that some variables may be removed while other variables may be added. Moreover, this new definition of MPP allows a partial assignment to be a solution of the problem which supports over-constrained as well as hard-to-solve problems. Last but not least, we have introduced a more specific distance function δ that measures the distance between two (partial) assignments as a number of variables with changed values. In brief, our formulation of MPP requires that only a minimal number of variable assignments is changed after the problem modification. This new formulation of MPP suits better our problem area (university timetabling) because it supports arbitrary changes to the problem (e.g., adding and removing lectures) and it allows partial assignments to be a

solution (e.g., when a complete timetable cannot be found). The changed variables describe naturally the changes in the timetable.

In this paper, we propose a slightly modified formulation of MPP. It preserves the good characteristics of the definition proposed at [1], i.e. arbitrary changes to the problem and handling partial assignments. Also the new definition generalizes a distance function. We also describe a new algorithm to solve a particular instance of MPP. This algorithm is capable of solving over-constrained and hard-to-solve problems while still guaranteeing a limited running time. The algorithm is based on the concept of a limited assignment number (LAN) search algorithm that we had proposed in [18] to find an initial solution of the problem. This algorithm tries to assign a maximal number of variables in such a way that the resulting problem is still consistent. The LAN Search algorithm does not guarantee finding a complete solution but it provides a good partial solution in a limited time.

The paper is organized as follows. We first describe the basic notions of constraint satisfaction problems along with our extension to describe partial solutions of over-constrained problems. Then we formally define a minimal perturbation problem (MPP) and its solution. Finally, we present a new search algorithm for solving a particular instance of MPP. This algorithm combines the principles of LAN Search and Branch-and-Bound algorithms.

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 (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 take at the same time.

A *solution* to the constraint satisfaction problem Θ is a complete assignment of the variables from V that satisfies all of the constraints.

For many problems it is hard or even impossible to find such a solution. For example, in over-constrained problems [9], there does not exist any complete assignment which satisfies all of the constraints. Therefore, other definitions of problem solution, like Partial Constraint Satisfaction [9], were introduced. The basic idea of these approaches is to weaken the problem, e.g., by adding more compatible tuples to the constraints [9]. In [1], we introduced a new approach to solve such problems based on weakening the definition of the solution. The idea is to assign as many variables as possible while still keeping this partial assignment consistent. If the users get such a partial assignment, they may relax some constraints in the problem (typically some of the constraints among the non-assigned variables that cause conflicts) so the system can try to extend the existing assignment to other variables.

Formally, let Θ be a CSP and C be a consistency technique (for example arc consistency). We say that a *constraint satisfaction problem* is *C-consistent* if the consistency technique deduces no conflict (e.g., for arc consistency, the conflict is

indicated by emptying some domain). We denote $C(\Theta)$ 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 a (partial) assignment of variables, then we denote $\Theta\sigma$ application of the assignment σ to the problem Θ, i.e., domains of the variables in σ are reduced to a single value defined by the assignment. Finally, we say that a *partial assignment* σ is *C-consistent* with respect to some consistency technique *C* if, and only if, $C(\Theta\sigma)$ holds. Note that a complete consistent assignment is a solution of the problem. Note also that backtracking-based techniques typically extend a partial consistent assignment towards a complete (consistent) assignment.

Example (partial arc-consistent assignment):

Assume the following CSP:

$\Theta = (\{a,b,c\}, \{D_a=\{1,2\}, D_b=\{1,2\}, D_c=\{1,2,3\}\}, \{a\neq c, a\neq b, b\neq c\})$.

Then $\sigma = \{a/1,b/2\}$ is a partial arc-consistent assignment because

$\Theta\sigma = (\{a,b,c\}, \{D_a=\{1\}, D_b=\{2\}, D_c=\{1,2,3\}\}, \{a\neq c, a\neq b, b\neq c\})$

is (arc) consistent (domains after reduction are $\{D_a=\{1\}, D_b=\{2\}, D_c=\{3\}\}$).

As we already mentioned, for some problems there does not exist any complete consistent assignment. These problems are designated as "over-constrained". In such a case, we propose to look for the maximal *C*-consistent assignment. We say that the *C-consistent assignment* is *maximal* for a given CSP if there is no other *C*-consistent assignment with a larger number of assigned variables. We can also define a weaker version, so called *locally maximal C-consistent assignment*. Locally maximal *C*-consistent assignment is a *C*-consistent assignment that cannot be extended to another variable(s). Notice the difference between the above two notions. The maximal *C*-consistent assignment is defined using the cardinality of the assignment (the number of assigned variables) and it has a global meaning. The locally maximal *C*-consistent assignment is defined using a subset relation, i.e., it is not possible to assign an additional variable without getting inconsistency. The maximal *C*-consistent assignment is the largest (using cardinality) locally maximal *C*-consistent assignment.

Using a particular consistency technique *C* gives users the possibility to define the desired features of the assignment. For example, if the consistency test checks only validity of the constraints between the assigned variables then we get the largest partial assignment and this assignment cannot be extended to another variable without getting a constraint violation. If a stronger consistency technique is used, e.g., arc consistency, we may expect shorter consistent assignments. However, these assignments can be extended to another variable *v* (for AC) without getting a constraint violation (but with getting a failure of the consistency test which indicates that there it is not possible to extend the assignment beyond *v* without getting a constraint violation). In some sense, a solution defined using a stronger consistency level gives the user some advance for future extensions of the assignment. In particular, publishing a shorter solution gives more options for future changes. If these problem changes will not come then the user may easily extend the solution (a bit) without violating the constraints. Still, we want to publish assignments that are as large as possible.

If the constraint satisfaction problem has a solution, then any maximal consistent assignment is the solution. Thus, looking for a maximal consistent assignment is a general way of solving CSP because it covers standard CSP as well as over-

constrained problems. Moreover, it is not necessary to know in advance whether the problem is over-constrained or not. However, it may be even hard to find a maximal consistent assignment for some problems. In such a case, we propose to return the largest locally maximal consistent assignment that can be found using given resources (e.g., time). This approach has a strong real-life applications, for example in scheduling and timetabling [14,17]. It means that the system allocates as many activities as possible in given time and that no more activity can be allocated without getting inconsistency. Typically, the solving algorithms based on the above idea select some sub-space of the solution space. For this sub-space, they find a maximal consistent assignment which is a locally maximal consistent assignment in the original solution space. For example, the LAN Search algorithm [18] restricts the number of assignments tried per variable.

Minimal Perturbation Problem – A Formal View

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

- Θ is a CSP,
- C is a consistency technique defining the solution, i.e. a maximal C-consistent assignment,
- δ is a *distance function* defining a distance between two (partial) assignments,
- α is a (partial) assignment for Θ called an *initial assignment*.

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

Recall that the minimal perturbation problem should formalize handling changes in the problem formulation. One may ask where the original problem Θ_{orig} is in the above definition. We can describe the problem change via a function F that maps the variables in Θ_{orig} to the variables in Θ. For some variables v from Θ_{orig}, the function F might not be defined which means that the variable v has been removed from the problem. However, if the function F is defined (we write $F(v)\downarrow$ if F is defined for v) then it is unique (it is a one-to-one mapping). Formally, $v \neq u \,\&\, F(v)\downarrow \,\&\, F(u)\downarrow \Rightarrow F(v) \neq F(u)$. Also, for some variables v from Θ, the origin might not be defined (i.e., there is no variable x such that $F(x) = v$), which means that the variable v has been added to the problem. Notice also that the constraints and domains can be changed arbitrarily when going from Θ_{orig} to Θ. We do not need to capture such changes using the mapping functions like F because we are concerned primarily with the variable assignments. Now, assume that σ is a solution to Θ_{orig} (in general, σ can be any assignment of variables in Θ_{orig}, but it is more natural for σ to be a solution to Θ_{orig}).

Then the initial assignment α in the definition of MPP is defined in the following way:

$$\alpha = \{v/h \mid \exists x \in \Theta_{orig}\ F(x)=v \ \&\ x/h \in \sigma\}^{1}.$$

However, only changes in the solution are important for us. The current definition already allows arbitrary changes to the problem formulation. In addition, it is enough to know the initial assignment α to solve MPP. As a consequence, we do include neither the original problem Θ_{orig} nor the mapping function F in the definition of MPP.

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 a two (partial) 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, i.e.

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

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 for variables' domains then it is possible to specify other metric distance functions, for example:

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

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 more general but to keep the spirit of our original formulation from [1].

Example:

Let α={b/3} be an initial assignment for a CSP Θ with variables V={b,c,d}, domains D = {D_b={1,3}, D_c={1,2,3}, D_d={2,3}}, and constraints C = {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 $\Pi = (\Theta, AC, |W|, \alpha)$.

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

MPP Solver

This section describes a depth-first search algorithm finding solutions to MPP where arc-consistency is used as the consistency technique. Function W defines the distance between two assignments. In terms of CSP, we describe a labeling procedure that is applied to Θ. This procedure works in steps. In each step, it selects a not-yet assigned variable and tries to find a value for this variable. We use a standard MAC (Maintaining Arc Consistency) approach to keep the constraint network arc consistent during the search.

We formulate the minimal perturbation problem in such a way that an incomplete assignment could be a solution to the problem. It helps us to solve over-constrained as well as hard problems. Traditionally, when all attempts to assign a value to the variable failed, depth-first search algorithms would backtrack to the last assigned variable. Then it would try to find another value for it. To find a locally maximal consistent assignment, we introduce here the concept of *locked variables*. Instead of immediate backtracking, the variable whose assignment failed is locked and the search procedure proceeds to the next, not yet assigned, variable. The locked variables still participate in constraint propagation so the above mechanism extends any partial assignment of variables to a locally maximal arc-consistent assignment. Notice also that the locking mechanism is local so the variable is locked only in a given search sub-tree. Let us now consider what happens when algorithm backtracks above the level where the variable has been locked. At this point, the variable is unlocked and it can then participate in labeling again.

Recall that our original motivation was to support interactive changes in the problem, which expects that the solver will return a solution quickly after any change. Since 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. This number (called *LAN limit)* is maintained for each variable separately. This differentiates the LAN Search from other incomplete tree search techniques like Bounded Backtrack Search [10] or Credit Search [3]. By applying the above restriction, we get a linear search space (*lan_limit * number_of_variables*) instead of exponential ($domain\_size^{number\_of\_variables}$). On the other hand, the algorithm lost completeness so it does not guarantee finding an optimal solution (which is not a problem in real-life applications provided that the algorithm finds "a good solution in a reasonable time").

The LAN principle is implemented using a counter for each variable. The counter is initialized by the LAN limit and after each successful assignment of value to the variable[2], the counter is decreased. When the counter is zero, the variable has reached the maximal allowed limit of assignments – and we say that the *variable expired*. The labeling procedure does not attempt to assign a value to expired variables. However, a value can still be selected for the expired variables via constraint propagation.

[2] Successful assignment of value to the variable means that the value is assigned to the variable, this change is propagated via constraints to other variables and no inconsistency is detected.

A minimal perturbation problem is a sort of constraint optimization problem. There are two objective criteria behind MPP; namely maximizing the size of the assignment and minimizing the number of perturbations. The formal definition of MPP expects a lexicographic ordering [max. size, min. perturbations] so the labeling procedure follows these objectives. The core labeling algorithm, as described above, explores locally maximal arc-consistent assignments. To solve MPP, we need to extend the algorithm to a branch-and-bound like scheme. First, when the algorithm finds a locally maximal arc-consistent assignment, it saves it as a bound and then it continues in exploring the search tree. In each step (an attempt to label some variable), the algorithm checks whether the current partial assignment can be a solution, i.e., whether it can be better than the so-far best saved assignment. If the answer is no then we can backtrack immediately without exploring the sub-tree. To do this comparison, we need to estimate the features of the best locally maximal consistent assignment that can be obtained from the current partial assignment. In particular, we need to estimate the maximal size of such extended assignment and the minimal number of perturbations. To keep the algorithm sound, we need an upper estimate of the size and a lower estimate of the number of perturbations. The upper estimate of the size 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 so this variable will not participate in the locally maximal arc-consistent assignment. On the other hand, the expired variables may still be part of the locally maximal arc-consistent assignment because the value can be selected for them via constraint propagation. We estimate the minimal number of perturbations simply by counting the variables where the initial value is out of its 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 (which is realized by using a dummy initial value for v out the domain D_v). Note, that there is a constant number of such variables, say K, so minimizing |W| is equivalent to minimizing |W|+K.

It is possible to guide the search procedure towards a solution of MPP via standard variable and value selection heuristics. We decided to abstract some of this general information from heuristics directly to the labeling procedure. Then the heuristics can be designed more for a particular problem rather then coping with general principles of MPP. Our motivation is as follows: if the initial value is in the domain of the variable then we may expect that this is a good value for the variable because it was already part of the solution of a similar problem. Thus, we have a verified heuristics telling us what value should be used. This follows the results from [15] where it has been shown theoretically that backtracking-based search guided by information about a complete assignment has a better average running time. For variables without an initial value in their domain, we must rely on general heuristics which have not been verified yet (and thus, it is harder to find a good value for such variables). According to the first-fail principle, it is better to handle variables which do not have an initial value in their domains first, and then to try finding a value that keeps the initial values in the domain of the other variables. In the second stage, the algorithm tries to assign an initial value to remaining variables. If it is not possible to use the initial value for

some variable[3], then the algorithm removes the initial value from domain of this variable and other values will be tried in turn. The built-in variable selection heuristic prefers variables with no initial value in their domain. Figure 1 shows an abstract scheme of the proposed labeling procedure. The procedure `distribute` selects only non-locked, non-expired variables and distributes them into two groups of variables with/without initial value in their current domain.

```
   label(Variables,LockedVariables)
1     if validate_bounds(Variables) then
2        (VarsNoInit,VarsWithInit)<-distribute(Variables,LockedVariables)
3        if empty VarsNoInit then
4           if empty VarsWithInit then
5              save_best_solution(Variables)
6           else
7              V <- select_variable(VarsWithInit)
8              label(Variables,LockedVariables) under V=InitValue(V)
9              label(Variables,LockedVariables) under V\=InitValue(V)
10             label(Variables,[V|LockedVariables])
11          end if
12       else
13          V <- select_variable(VarsNoInit)
14          Value <- nil
15          while Value<-select_next_value(V,Value) & lan_limit_ok(V) do
16             label(Variables,LockedVariables) under V=Value
17          end while
18          label(Variables,[V|LockedVariables])
19    end if
20 end label

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

Fig. 1. Labeling procedure for solving minimal perturbation problems.

We have implemented the proposed labeling procedure in `clpfd` library of SICStus Prolog [2] and we tested it using several benchmark problems derived from timetabling problems. In these tests, the goal is to place a set of rectangles to a restricted area. The position of the rectangles can be further restricted via constraints. We start with some correct placement of rectangles which represents the initial solution. Then, for randomly selected rectangles we changed their position constraints in such a way that the current position is not allowed. The task is to find a new position of rectangles satisfying the position constraints and minimizing the number of differences from the initial solution (Figure 2 shows an example of such problem).

[3] Constraint propagation is local so even if the value is still in the variable domain, it does not imply that the value can be assigned to the variable.

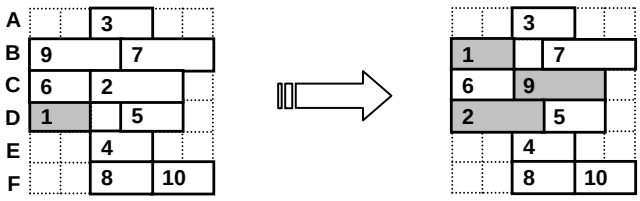

Fig. 2. Solution of a simple minimal perturbation problem: rectangle 1 is required to be placed in row B while rectangles 7 and 9 are required to stay in rows A-C. The solver found an optimal solution with 3 perturbations (right).

Conclusions

Including dynamic features is a recent trend in planning, scheduling, and timetabling. In this paper, we addressed one of these features - namely finding a solution for a changed problem that does not differ much from the solution of the original problem – a so called "minimal perturbation" problem (MPP). We reformulated the definition of MPP from [1] to be more general and we proposed a new labeling algorithm to solve a particular instance of the minimal perturbation problem. The algorithm has a linear time complexity, thanks to a LAN limit [18], and it is capable of solving over-constrained and hard problems by looking for locally maximal arc-consistent assignments. A new technique of variable locking has been introduced there to find locally maximal arc-consistent assignments.

The preliminary tests showed that the proposed algorithm is able to find optimal or close to optimal solutions of MPP. Extensive tests of this model are currently being performed. Our future works will include experiments with different variable ordering heuristics for the labeling procedure as well as comparing different concepts of the solving algorithm, for example algorithms based on local search techniques and their combination with backtracking techniques [21] and with constraint propagation [12,16]. Also the modification of other tree search techniques, such as the Limited Discrepancy Search [11], is assumed.

Acknowledgements

Research is supported by the Czech Science Foundation under the contract no. 201/99/0942 and by Purdue University. We would also like to thank the reviewers for useful comments, in particular for pointing our attention to the formulation of the maximal consistent assignment.

References

1. R. Barták, T. Müller, and H. Rudová. *Minimal Perturbation Problem – A Formal View*. Neural Network World 13(5): 501-511, 2003.
2. M. Carlsson, G. Ottosson, and B. Carlson. *An open-ended finite domain constraint solver*. In Programming Languages: Implementations, Logics, and Programming. Springer-Verlag LNCS 1292, 1997.
3. A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen and M. G. Wallace. *ECLiPSe: An Introduction*. IC-Parc, Imperial College London, Technical Report IC-Parc-03-1, 2003.
4. R. Dechter and A. Dechter. *Belief maintenance in dynamic constraint network*. In AAAI-88, pp. 37-42, AAAI Press, 1988.
5. B. Drabble, N. 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. H. El Sakkout, T. Richards, and M. Wallace. *Minimal Perturbation in Dynamic Scheduling*. In Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98). John Wiley & Sons, 1998.
7. H. El Sakkout and M. Wallace. *Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling*. Constraints 4(5): 359-388. Kluwer Academic Publishers, 2000.
8. M. S. Fox. *ISIS: A Retrospective.* In Intelligent Scheduling. Morgan Kaufmann Publishers, pp. 3-28, 1994.
9. E.C. Freuder and R.J. Wallace. *Partial Constraint Satisfaction.* Artificial Intelligence, 58:21-70, 1992.
10. W. D. Harvey. Nonsystematic backtracking search. PhD thesis, Stanford University, 1995.
11. W. D. Harvey and M. L. Ginsberg. *Limited discrepancy search*. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 607-615, Morgan Kaufmann, 1995.
12. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence, 139(1):21 45, 2002.
13. I. Miguel and Q. Shen. *Hard, Flexible and Dynamic Constraint Satisfaction*. Knowledge Engineering Review, 14(3):199–220, 1999.
14. T. Müller and R. Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results.* In Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT2002), pp. 58-72, 2002.
15. P. W. Purdom, Jr. and G. N. Haven. *Probe order backtracking.* SIAM Journal on Computing, 26(2):456-483, 1997.
16. Y. P. Ran, N. Roos, H. J. Herik. *Approaches to find a near-minimal change solution for Dynamic CSPs*. In Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, pp. 373-387, 2002.
17. H. Rudová and K. Murray. *University Course Timetabling with Soft Constraints*. In Practice And Theory of Automated Timetabling IV. Springer-Verlag LNCS 2740, 2003.
18. H. Rudová and K. Veřmiřovský, *Limited Assignment Number Search Algorithm*. In SOFSEM 2002 Student Research Forum, pp. 53-58, 2002.
19. S. F. Smith. *OPIS: A Methodology and Architecture for Reactive Scheduling.* In Intelligent Scheduling, pp. 29-66, Morgan Kaufmann Publishers, 1994.
20. G. Verfaillie and T. Schiex. *Solution reuse in dynamic constraint satisfaction problems*. In AAAI-94, pp. 307-312, AAAI Press, 1994.
21. J. Zhang and H. Zhang. *Combining local search and backtracking techniques for constraint satisfaction*. In AAAI-96, pp. 369-374, AAAI Press, 1996.