Constraints (2011) 16:228–249
DOI 10.1007/s10601-011-9108-5

Hybrid search for minimal perturbation in Dynamic CSPs

Roie Zivan · Alon Grubshtein · Amnon Meisels

Published online: 15 May 2011

Abstract It is often the case that after a scheduling problem has been solved some small changes occur that make the solution of the original problem not valid. Solving the new problem from scratch can result in a schedule that is very different from the original schedule. In applications such as a university course timetable or flight scheduling, one would be interested in a solution that requires minimal changes for the users. The present paper considers the minimal perturbation problem. It is motivated by scenarios in which a Constraint Satisfaction Problem (CSP) is subject to changes. In particular, the case where some of the constraints are changed after a solution was obtained. The goal is to find a solution to the changed problem that is as similar as possible (e.g. includes minimal perturbations) to the previous solution. Previous studies proposed a formal model for this problem (Barták et al. 2004), a best first search algorithm (Ross et al. 2000), complexity bounds (Hebrard et al. 2005), and branch and bound based algorithms (Barták et al. 2004; Hebrard et al. 2005). The present paper proposes a new approach for solving the minimal perturbation problem. The proposed method interleaves constraint optimization and constraint satisfaction techniques. Our experimental results demonstrate the advantage of the proposed algorithm over former algorithms. Experiments were performed both on random CSPs and on random instances of the Meeting Scheduling Problem.

Keyword Dynamic Constraint Satisfaction Problems (CSPs)

R. Zivan
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel
e-mail: zivanr@bgu.ac.il

A. Grubshtein (✉) · A. Meisels
Department of Computer Science, Ben-Gurion University of the Negev,
Beer-Sheva, 84105, Israel
e-mail: alongrub@cs.bgu.ac.il

A. Meisels
e-mail: am@cs.bgu.ac.il

1 Introduction

Constraint Satisfaction Problems (CSP) form an elegant model for representing many real world combinatorial problems. CSPs are often used for representing and solving scheduling problems [2, 9, 20]. In many scheduling applications, constraints tend to change. In fact, some of the constraints might change after a solution was obtained. Some examples are a problem of assigning nurses to shifts where a nurse might call in sick, and a schedule of tasks in an industrial environment where a machine can break down or a new urgent order might arrive. In many of these scenarios, it is not enough just to find any other solution that satisfies the constraints of the newly derived problem. Solutions, which are very different from the solution of the original problem, are many times less desirable because they require too many adjustments. In the school timetabling example, teachers and students have their schedules and plan their actions according to it. Thus, when some constraints change, it is desired to find a new solution that is as similar as possible to the solution that was valid before the problem changed.

The present paper focuses on scenarios where after a CSP was solved, some of the constraints have changed and the solution to the newly generated CSP is required to include minimal perturbation, i.e., to be most similar to the solution of the former problem. This additional requirement defines a qualitative order on the set of solutions of the new (or changed) problem and the search for the most similar solution becomes an optimization problem.

While industrial scheduling applications often include constraints that allow solving this additional optimality requirement via efficient mathematical tools (e.g., linear programming [15]), discrete scheduling problems like timetabling applications require exhaustive search methods to find the optimal solution with minimal perturbation [3, 6]. This paper demonstrates that by exploiting the problem's properties, it is possible to effectively reduce the additional effort.

In order to find the solution to a newly generated CSP that is the most similar to the previously valid solution, a new search method is proposed that exploits the fact that its outcome must satisfy two requirements. These requirements are *of different levels of complexity*. While the similarity requirement is an optimization requirement, the second requirement (e.g., to solve the modified problem) is a satisfaction requirement. A constraint satisfaction problem is NP-complete. In contrast, finding the most similar solution to a given complete assignment is harder than NP [6]. Former solutions to the above problem were based on algorithms that attempt to fulfill both requirements in a single phase [3, 6, 11]. As a result, the entire search space was scanned in order to find a solution that satisfies both requirements. A different approach can be formed by observing the following two facts. On the one hand, the satisfaction requirement involves the complete search space of the new (changed) CSP. On the other hand, in order to solve the similarity requirement, only the values that were assigned to variables in the solution for the original problem need to be considered (any other value would be considered a perturbation). Thus, when dealing with the optimization problem, which is a harder problem, one can actually consider a much smaller search space.

The present paper solves the problem by iteratively alternating between two phases. In the first phase, a branch and bound ($B\&B$) algorithm is used to find a consistent partial assignment that includes only value assignments of the former solution. In this phase, only constraints between the assignments of the former solution are being checked. An admissible heuristic that counts possible conflicts between these value assignments generates a significant speedup. When a partial assignment that includes only value assignments of the former solution is found by the first phase of the algorithm, a second phase is performed to validate that this partial assignment can be extended to a solution to the new CSP. In this phase, a *maintaining arc consistency* (MAC) algorithm [4] is used and all values of unassigned variables are taken into consideration. The search space to be scanned by the MAC algorithm can become smaller, as the size of the partial solution found by the first phase increases.

When the partial solution found in the first phase can be extended to a solution to the new CSP, the number of perturbations is updated and becomes the new upper bound and the solution is stored as the most similar that was found by the algorithm. Then, the algorithm resumes its search for an assignment with fewer perturbations, performing again the first phase.

Our experimental study shows a clear advantage of the approaches that follow a $B\&B$ scheme over the best first systematic traversal of the search space proposed by former studies [11]. The only exceptions are cases in which there exists a solution within a very small number of perturbations. The proposed hybrid search also outperforms the $B\&B$ algorithm that enforces *generalized arc consistency* GAC [6] by a large factor. The experiments were performed both on randomly generated uniform problems and on instances of the Meeting Scheduling Problem, one of the real world scenarios that triggered this study.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 presents the minimal perturbation problem and Section 4 presents the proposed hybrid algorithm. A correctness proof for the proposed algorithm is presented in Section 5. Our experimental study, which compares the proposed hybrid algorithm with Roos's algorithm [11] and the B&B with GAC algorithm [6] on two families of problems, is presented in Section 6, followed by our conclusions.

2 Related work

Schiex and Verfaillie [16, 19] studied the problem of solving Dynamic CSPs. They define a Dynamic CSP as a series of CSPs that differ one from the other in some of their constraints (added constraints or removed constraints, or both). The main focus of these studies was to find a solution for each CSP in the series as fast as possible. Thus, a number of methods for acquiring a new solution for a changed CSP were proposed. One of the proposed methods reuses inconsistent assignments (*Nogoods*) that were found while solving the original CSP, when searching for a solution to the revised CSP. Another method uses dynamic backtracking and local search techniques starting with the solution of the original problem. However, none of the proposed methods is guaranteed to find the *solution most similar* to the previous solution. Still, the nogood recording (NR) method can be incorporated with any of the methods we present in this paper, regardless of whether they were proposed by us or by others. Our experimental results have shown that the use of short nogoods as proposed in [16] does not improve the run of the algorithms. The reason is that NR is effective for problems in which a CSP search algorithm generates short Nogoods.

These problems include, in general, a very small number of solutions or no solutions at all. On the other hand, the relevant CSPs for the minimal perturbations problem (MPP) are problems with multiple solutions, where it is possible to define an order on the quality of different solutions.

In [11], Roos et al. propose a method that, given a revised CSP and a solution to the original problem, finds a solution to the new CSP that is the most similar to the previous solution. The proposed method traverses the assignments of the CSP according to their distance from the former solution and enforces arc consistency after each change. Roos et al. found that this method is feasible only when the constraints that change are unary constraints, and that it causes a significant slow down in comparison with an algorithm that searches for any solution to the new problem (our experimental study validates the inefficiency of this method). In a later study, the authors offer approximation methods that are apparently feasible but do not guarantee the finding of the most similar solution [12].

Hebrard et al. studied different aspects of finding similar and diverse solutions to specific assignments for a CSP [6]. Their study is motivated by a variety of applications, e.g., preferences elicitation, interactive constraint definitions, and the stability of solutions in a dynamic environment. In [6], the authors prove that finding a solution to a CSP that is as close to (or as distant from) a given set of assignments is *NP-hard* (actually they prove it is $FP^{NP[\log n]} - complete$). They also propose an algorithm for finding the closest solution to a given set of assignments. The proposed algorithm is based on a Branch and Bound scheme and enforces generalized arc consistency (GAC) on a soft global distance constraint in each step of the algorithm. In a different paper, they propose an algebra that enables a combination of similarity constraints to a set of ideal partial assignments, as well as distance constraints from non-ideal partial assignments [7].

The minimal perturbation problem was studied for classic scheduling problems in [14]. The authors proposed an integration of mathematical programming techniques with constraint programming in order to speedup the search on this special optimization problem. More specifically, the problem at hand is analyzed and constraints that enable the use of mathematical programming, e.g., linear constraints, are solved by efficient algorithms. The solutions are used within the constraint program. This approach was extended in [15]. The authors discuss the potential of their approach in detecting sub-problems, which can be solved more efficiently in different CSP applications. Although the present paper is concerned with discrete problems that do not admit of mathematical programming, our work is a response to the proposal for future extensions of the approach presented in [15].

Bartak et al. proposed a formal model of the minimal perturbation problem based on the CSP model and a Branch and Bound algorithm for solving it [3]. The innovation in the proposed algorithm is that it allows incomplete solutions to the problem. The Branch and Bound algorithm maximizes two criteria, the minimal perturbation or maximal similarity as in [6] and the length of the solution found. The present paper adopts the formalism proposed by [3], but seeks to find complete solutions to the problem with minimal perturbation as in [6].

A thorough survey on constraint solving in uncertain and dynamic environments can be found in [18]. The survey considers the wide range of issues and properties of such problems including multiple approaches and frameworks to handle them. These include solution reuse, reuse of reasoning, and the finding of robust (stable) solutions. For the problem of minimal perturbation ("minimal change" in their terminology), they differentiate the CSP version of the problem [12] and the scheduling problem [15]. However, the survey does not describe methods for finding minimal perturbation solutions.

3 Problem definition

The formal description in this section follows the formalism proposed for this problem in [3]. A *Constraint Satisfaction Problem* (*CSP*) is a triple $\Theta = (V, D, C)$, where

- $V = \{v_1, v_2, ..., v_n\}$ is a finite set of variables.
- $D = \{D_1, D_2, ..., D_n\}$ is a set of domains where D_i is a finite discrete set of possible values for the variable v_i.
- $C = \{c_1, c_2, ..., c_m\}$ is a finite set of constraints restricting the values that the variables can simultaneously take.

A *partial assignment* σ is a set of pairs $< var_i, val_{i_j} >, ..., < var_l, val_{l_k} >$, where $var \in V$, $val_i \in D_i$ and a variable is not included in more than one pair. A *complete assignment* is a partial assignment that includes n pairs. Each member of the set of constraints $c_i \in C$ is a partial assignment. A *partial solution* is a consistent partial assignment, i.e., a partial assignment in which no constraints are violated (does not include any members of C). A **solution** to a *CSP* is a partial solution with n value assignments to variables (or a complete assignment with no violated constraints). In other words, $c_i \in C$ is a partial assignment that cannot be a part of a solution. Formally σ is a solution if and only if $\forall \sigma' \subseteq \sigma \Rightarrow \sigma' \notin C$ and $|\sigma| = n$.

Following common practice in constraint programming [10, 17], we only consider CSPs that include only binary constraints for simplicity of presentation. Such a simplification is common in constraint programming since any problem with n-ary constraints can be represented by a problem that includes only binary constraints [1]. However, neither our proposed algorithms nor their implementation are limited to such problems.

A *minimal perturbation problem* (MPP) is a triple $\Pi = (\Theta, \alpha, \delta)$, where:

- Θ is a CSP.
- α is a partial assignment for Θ that is called *initial assignment*.[1]
- δ is a function that defines a distance between any two assignments.

A solution to an MPP problem is a solution to Θ with minimal distance from α according to δ.

For the problems we consider in this paper (and following [3]), δ is defined as follows: Let σ and γ be partial assignments for Θ. We define $W(\sigma, \gamma)$ to be a set of variables such that the value assignment for v in σ is different from the value assignment for v in γ:

$$W(\sigma, \gamma) = \left\{ v \in V \mid \langle v, val \rangle \in \sigma, \langle v, val' \rangle \in \gamma, val \neq val' \right\}$$

[1] Although the formalism does not require the assignment to be complete, it is reasonable from the motivating scenarios that in most problems to be considered, α will be a complete assignment.

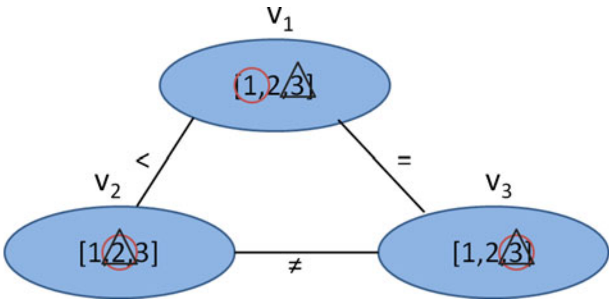


Fig. 1 Example: Problem Θ, assignment α (depicted by *circles*) and the solution with minimal perturbations (depicted by *triangles*)

In [3], $W(\sigma, \gamma)$ is called a *distance set* for σ and γ and the elements of the set are called *perturbations*.

In an MPP, the distance function of some assignment σ from α is defined as the cardinality of the set $W(\sigma, \alpha)$:

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

In other words, δ is the Hamming distance between σ and α.

Notice that the formalization does not include the problem for which α is a solution. That is because the formalism is more general and applies to minimal perturbations from any given assignment α to Θ.

We will refer to the value assignments to the variables within α as *Solution Value Assignments* (*SVA*s). More specifically, if $\langle v_i, val_{i_j} \rangle \in \alpha$ then $v_i.SVA = val_{i_j}$.

An example of an MPP is depicted in Fig. 1. The example includes a CSP Θ with three variables each with three possible values. The constraints between the variables are binary and depicted by logic signs near each arc. The circles represent the assignment α. The triangles represent the solution to Θ that is most similar to α. In Section 4 we use this example MPP to demonstrate the operation of our proposed algorithm.

4 Finding a solution to a minimal perturbation problem

The proposed hybrid search algorithm for the minimal perturbation problem (*HS_MPP*) includes two phases that are interleaved throughout the search. In the first phase the algorithm assigns *SVAs* to variables. This generates a partial solution that includes only *SVAs*, i.e., a partial solution σ to Θ with $\delta = 0$ (zero distance between α and σ). This phase of the algorithm implements a Branch and Bound scheme where the upper bound is the smallest δ among the solutions to Θ that were found so far by the algorithm. If the algorithm detects that the partial solution σ cannot be extended to a solution with a smaller δ than the current upper bound, it backtracks. In order to detect the need to backtrack as early as possible, the algorithm uses an admissible heuristic function described in Section 4.1. If no more *SVAs* can be assigned to unassigned variables (i.e., all SVAs of unassigned variables conflict with assigned SVAs) and the admissible heuristic does not breach the upper bound, the second phase of the algorithm is performed. In the second phase, a Maintaining Arc-Consistency algorithm (MAC-algorithm) is performed in order to validate that the partial solution σ with $\delta = 0$, found in the first phase, can be extended to a complete solution. If the second phase ends successfully and a solution γ is found, it is recorded as the best solution so far and the upper bound is set to $\delta(\gamma, \alpha)$. The other option is that the satisfaction algorithm finds that σ cannot be extended to a complete solution to Θ. In both cases, after this phase is completed, the algorithm resumes the first phase by removing the last *SVA* assignment (i.e., backtracking).

Figures 2 and 3 present the code of the proposed hybrid algorithm for minimal perturbation problems (*HS_MPP*). The main loop of the algorithm calls the first phase in order to find a partial solution with δ equal to zero, which cannot be extended to a larger solution with δ equal to zero and which is larger than n minus the upper bound. When such an assignment is found, the second phase is called in order to validate that the assignment can be extended to a solution to Θ.

Function **phase1** tries to assign an SVA by calling function **assign_SVA**. After each successful assignment of an SVA, the function checks that the current assignment can still lead to a solution with δ smaller than the upper bound (line 9). If not, the algorithm backtracks. When there are no more SVAs to assign, the function **assign_SVA** will return false. Since the upper bound was not breached after the last SVA was assigned, the current partial assignment σ is a partial solution, which, if extended to a solution to Θ, will have a smaller δ than the upper bound. Thus, the function returns true and the second phase is called.

In function **phase2** a standard maintenance of arc consistency (MAC) algorithm is used to check whether the partial solution that was found in **phase1** can be extended

```
HS_MPP:
1.  upper_bound ← n
2.  solution ← null
3.  current_assignment ← $\phi$
4.  unassigned_variables ← $\Theta.V$
5.  while (phase1)
6.     phase2
7.  return solution

phase1:
8.  while(assign_SVA)
9.     while ($n$ − (|current_assignment|+ heuristic ) $\geq$ upper_bound)
10.       if (|current_assignment| > 0)
11.          backtrack
12.       else
13.          return false
14. return true

phase2:
15. current_solution ← MAC(current_assignment)
16. if(current_solution $\neq$ null)
17.    upper_bound ← $\delta$(current_solution, $\alpha$)
18.    solution ← current_solution
19. while ($n$ − (|current_assignment|+ heuristic ) $\geq$ upper_bound and |current_assignment| > 0)
20.    backtrack
```

Fig. 2 Main part of the hybrid search MPP algorithm

```
backtrack:
21. $v \leftarrow current\_assignment.last$
22. remove_SVA ($v$)
23. for each ($v' \in v.removed\_conflicting\_SVAs$)
24.    move_back_to_domain($v'.SVA$)
25.    $v.removed\_conflicting\_SVAs \leftarrow v.removed\_conflicting\_SVAs \setminus v'$
26. unassigned_variables.add(current_assignment.last)
27. current_assignment.remove_last

assign_SVA:
28. $v \leftarrow$ select_SVA_var
29. if ($v \neq null$)
30.    for each v' ∈ unassigned_variables
31.       if (v'.SVA ∈ v'.domain and v'.SVA conflicts with v.SVA)
32.          remove_from_domain(v', SVA)
33.          $v.removed\_conflicting\_SVAs \leftarrow v.removed\_conflicting\_SVAs \cup v'$
34.    current_assignment.add(v)
35.    unassigned_variables.remove(v)
36.    return true
37. else
38.    return false

remove_SVA ($v$):
39. $current\_assignment \leftarrow current\_assignment \setminus v$
40. remove_from_domain(v, SVA)
41. $v' \leftarrow current\_assignment.last$
42. if ($v' \neq null$)
43.    $v'.removed\_conflicting\_SVAs \leftarrow v'.removed\_conflicting\_SVAs \cup v$

select_SVA_var:
44. if ($\exists v_i$ s.t. $v_i \notin current\_assignment$ & $SVA \in D_i$)
45.    return $v$
46. else
47. return null
```

Fig. 3 Functions used by the hybrid search MPP algorithm

to a complete solution of Θ (line 15). If it does, this solution is stored and its δ is saved as the new upper bound (lines 16–18). If not, the function backtracks (lines 19, 20).

Function **assign_SVA** makes an attempt to assign an unassigned variable with its SVA (find an unassigned variable whose SVA is still in its domain). If it succeeds, the function removes all conflicting SVAs from the domains of all unassigned variables (lines 30–32). The function returns false if none of the unassigned variables has an SVA in its domain.

Function **backtrack** removes the SVA assignment that was performed last. All SVAs that were removed from the domains of unassigned variables after the last SVA was assigned are returned back to their domains (lines 23, 24).

Function **remove_SVA** removes the SVA assignment from the current partial assignment and removes the SVA from the variable's domain (lines 39, 40). It also adds the removed assignment to the set of assignments that were removed as a conflict with the SVA that was assigned before it (lines 41–43). This ensures that the

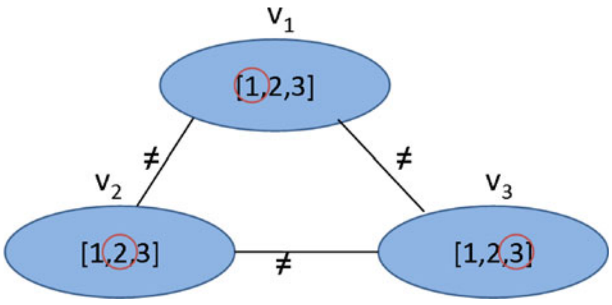


Fig. 4 Example: original problem and solution

SVA will be returned to the variable's domain once the exploration of the current partial assignment will be completed.

Function **select_SVA_var** simply returns a variable whose domain includes an SVA.

Figures 4, 5, and 6 will be used to demonstrate the run of the proposed HS_MPP algorithm. In Fig. 4, the original problem is depicted along with a consistent solution. The problem includes three variables $V = \{v_1, v_2, v_3\}$. All domains include three values $D_i = \{1, 2, 3\}$. There are inequality constraints[2] between v_1 and v_2, v_1 and v_3, and between v_2 and v_3. In Fig. 5 some of the constraints were changed. The constraint between v_1 and v_2 was changed to $v2 < v1$ and the constraint between v_1 and v_3 was changed to equality. After setting the upper bound to 3, the algorithm starts in phase 1 by assigning the SVA of v_1. After the assignment $\langle v_1, 1\rangle$, all of the SVAs that are conflicting with it are removed from the domains of their variables. In the present example, SVAs of both v_2 and v_3 conflict with the SVA of v_1 and are removed. Notice that we do not check whether other values conflict with this assignment. Since we did not describe the admissible heuristic, at this point we will use the number of SVAs in domains of unassigned variables as our example heuristic. Thus, at this point the lower bound is $3 - (1 + 0) = 2$, which is lower than the upper bound. The next attempt to assign an SVA fails so we move to phase 2. Since no value can be assigned to v_2, phase 2 fails to find a solution with a partial solution $\langle v_1, 1\rangle$. After backtracking, the assignment of v_1 is removed and the value 1 (the SVA) is removed from D_1. Then, phase 1 is resumed. The SVA of v_2 is assigned and since the SVA of v_3 is not in conflict with it, we assign it to v_3 and get the partial solution $\langle v_2, 2\rangle$, $\langle v_3, 3\rangle$. Once again, the second phase is called and a solution is found to the problem $\langle v_2, 2\rangle$, $\langle v_3, 3\rangle$, $\langle v_1, 3\rangle$. After storing the solution and updating the upper bound to 1, the algorithm backtracks by removing the assignment of v_3 from the current assignment and removing the value 3 from D_3. Then, since there are no SVAs in the domains of the unassigned variables, the algorithm backtracks again by removing the assignment of v_2, removing 2 from D_2, and restoring 3 to D_3. The next attempt to assign an SVA assigns 3 to v_3. However, since there are no SVAs remaining in D_2 and D_1, the lower bound is $3 - (1 + 0) = 2$, which is larger than the upper bound, 1. Thus, a backtrack is performed and the value 3 is removed from D_3. The next attempt to assign an SVA will fail and the algorithm will terminate. The

[2]For simplicity of the example, we use a logical description of the constraints instead of specifying all excluded pairs of value assignments.

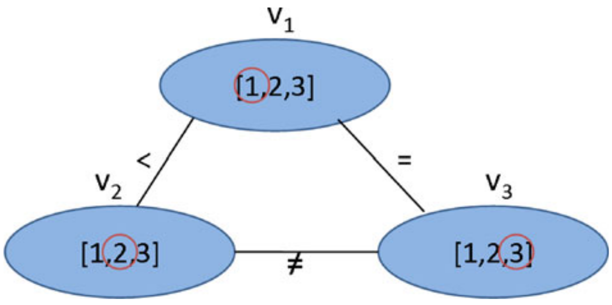


Fig. 5 Example: new problem Θ and assignment α

obtained solution with $\delta = 1$ is depicted in Fig. 6. The triangles represent the solution with minimal perturbations to the changed problem Θ, and the circles represent the original solution α.

4.1 An admissible heuristic for the first phase

The first phase of the HS_MPP algorithm is an optimization algorithm that uses a Branch and Bound scheme. As commonly used in optimization search methods, in order to speed up the algorithm one can use an admissible heuristic that will increase the lower bound [13]. As a result, the algorithm will detect a need to backtrack early.

In the first phase the algorithm is searching for a partial solution with $\delta = 0$, which is larger than the number of SVAs in the best solution found so far (γ). A first step in order to check whether the current assignment can be extended to a partial assignment of SVAs of the required length, would be to check whether there are enough SVAs left in the domains of unassigned variables. More specifically, if we denote by n the cardinality of $\Theta.V$, by k the length of the current partial solution σ, and by s the number of unassigned variables that include an SVA in their domain, then the current assignment can be extended to a solution γ' with $\delta(\gamma', \alpha) < \delta(\gamma, \alpha)$ only if $n - (k + s) < upper\_bound$. A larger lower bound can be achieved if we take into consideration that every pair of unassigned conflicting SVAs cannot be assigned in the same solution. Thus, instead of counting both SVAs in future assignments and incrementing s by two, we increment s by only one for each such conflicting pair. More formally, let $s = s_{\text{pairs}} + s_{\text{singles}}$, where s_{pairs} is the number of the distinct pairs whose SVAs conflict among the s unassigned variables that include SVAs in their domains, and s_{singles} is the number of the rest of the variables with SVAs in their

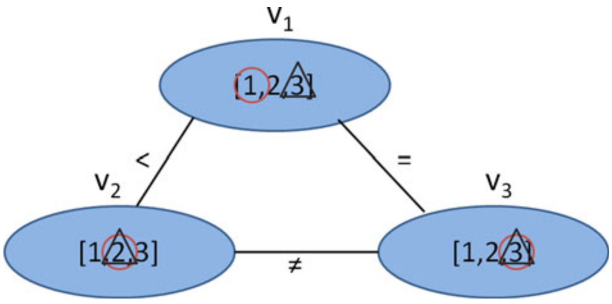


Fig. 6 Example: problem Θ, assignment α (*circles*) and the minimal perturbation solution (*triangles*)

Fig. 7 An admissible heuristic for HS_MPP

```
heuristic:
1.  count ← 0
2.  temp_set ← unassigned_variables
3.  for each ($v_i \in$ temp_set)
4.     if ($v_i.SVA \notin D_i$)
5.        temp_set ← temp_set \ $v_i$
6.     else if ($\exists v_j \in$ temp_set and $v_j.SVA$ conflicts with $v_i.SVA$)
7.        count ← count + 1
8.        temp_set \ {$v_i$, $v_j$}
9.     else
10.       temp_set ← temp_set \ $v_i$
11.       count ← count + 1
12. return count
```

domain. If $s_{\text{pairs}} > 0$, then $n - (k + s) < n - (k + s_{\text{pairs}} + s_{\text{singles}})$. By using the above admissible heuristic, one increases the probability of detecting a need to backtrack. This idea can be extended to cliques of any size (i.e., $s = s_{\text{singles}} + s_{\text{pairs}} + s_{\text{triplets}} \ldots$).

Figure 7 presents the code for the proposed admissible heuristic. First, a counter is initialized to zero (line 1). Next, all unassigned variables are gathered in a temporary set (line 2). The unassigned variables are removed one by one from the set. If a variable does not have the SVA in its domain, the counter is not incremented (lines 4, 5). If it does include an SVA, this SVA is checked for conflict with another SVA in the domain of some variable in the set. If so, both variables are removed from the set and the counter is incremented by one (lines 6–8). If not, only the single variable is removed and the counter is incremented by one (lines 9–11). The counter is returned as the heuristic value (line 12).

As an example of the benefits of the proposed admissible heuristic, consider the problem depicted in Fig. 8. In this example Θ includes 4 variables $V = \{v_1, \ldots, v_4\}$ and the constraints are inequalities between v_1 and v_2, and between v_2 and v_3, equality between v_1 and v_3, and $v_2 < v_4$. α is represented by circles on the SVAs. We examine the state after assigning $\langle v_1, 1\rangle$. Notice that the only SVA that is in conflict is $v_3.SVA$, and therefore it is removed from D_3. Now if we take the lower

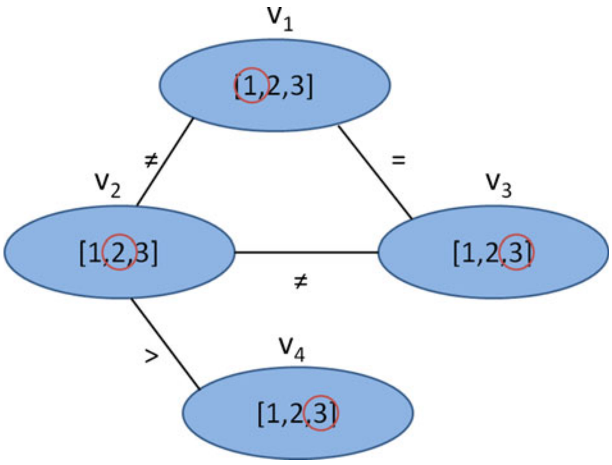


Fig. 8 Admissible heuristic example

bound as the size of the problem minus the size of the current assignment and the unassigned variables with an SVA in their domain, we get $4 - (1 + 2) = 1$. However, if we observe variables v_2 and v_4, we notice that their SVAs cannot be assigned simultaneously. Thus, we count them only once and get $4 - (1 + 1) = 2$ which is a higher lower bound.

4.2 Handling non-binary constraints

Our presentation of the proposed admissible heuristic seems to exploit the properties of problems with binary constraints. Although, as mentioned in Section 3, every CSP with non-binary constraints can be represented by a CSP that includes only binary constraints, it is important to mention that the proposed heuristic is applicable to problems that include non-binary constraints as well.

Note that the phase of removal of all SVAs that conflict with the current assignment, as done in lines 31–33 of the algorithm in Fig. 3 (function **assign_SVA**), is not dependent upon the constraints being binary. For non-binary constraints, after a new SVA assignment is added to the current assignment σ, the SVAs of every unassigned variable v_i should be removed from D_i if $\exists c \in C$ such that $c \subseteq \sigma \cup \langle v_i, v_i.SVA \rangle$.

For our admissible heuristic, a similar observation can be made. We include in s_{pairs} any pair of unassigned variables v_i and v_j for which:

$$\exists c \in C \; s.t. \; c \subseteq \sigma \cup \left\{ \langle v_i, v_i.SVA \rangle, \langle v_j, v_j.SVA \rangle \right\} \ .$$

5 Correctness of HS_MPP

In order to prove that the proposed algorithm is correct, one needs to prove that:

1. A solution reported is a solution to Θ.
2. The solution reported is the solution to Θ with minimal perturbations with respect to α.
3. The algorithm terminates.

The first is immediate. In phase one, after each SVA assignment, all conflicting SVAs are removed from the domains of unassigned variables (lines 21–33 in Fig. 3). Thus, two conflicting SVAs cannot be assigned in this phase. The fact that the extension of the solution in the second phase does not violate constraints is derived from the soundness of the standard MAC algorithm.

In order to prove the second condition for correctness, we need to prove that the solution reported for Θ, γ, is the most similar solution to α. More formally, $\forall \gamma'$, which is a solution to Θ, $\delta(\gamma', \alpha) \geq \delta(\gamma, \alpha)$. To this end we first prove the following lemmas:

Lemma 1 *The algorithm backtracks from a partial assignment generated in phase* 1, *only when the current assignment cannot be extended to a solution with more SVAs than in the best solution found so far.*

Proof There are only two places where the algorithm backtracks from an assignment generated in phase 1. The first is after the check for the upper bound is performed in line 9 of Fig. 2. The difference between the cardinality of the set $\Theta.V$ and the number of SVAs in the current assignment and in the domains of unassigned variables is a clear lower bound for δ. In addition, the heuristic described in Section 4.1 increases this lower bound by considering pairs of conflicting SVAs in the domain of unassigned variables only as a single possible assignment of an SVA. Thus, the heuristic reports the difference between the cardinality of V and possible SVA assignments and therefore it maintains its admissibility.

The second backtrack is performed at the end of phase two, when either a partial solution found in phase one was successfully extended to a solution and its δ was stored as the upper bound, or, the MAC algorithm failed to extend it to a complete solution.

In both cases, the potential of the current assignment to be extended to a solution with the minimal δ was either excluded by phase 1 or explored by phase 2. □

Lemma 2 *In phase* 1*, the algorithm never produces the same partial assignment twice.*

Proof In function **backtrack** the last assigned SVA is removed by calling function **remove_SVA**. When removing the assigned SVA, it is also removed from the domain of its variable. Then, the variable is added to the set of variables whose SVA removal was caused by the last assigned SVA. When the algorithm backtracks from the k'th assigned SVA, this SVA will never be assigned until the algorithm backtracks from the $k - 1$ SVA assignment. Thus, an SVA will not be assigned again until the partial assignment left after its removal is changed. This partial assignment will never be produced again because when the algorithm backtracks from the first assigned SVA, there is no earlier assignment that caused its removal; therefore it will never be assigned again. □

Theorem 1 *HS_MPP reports the solution with the minimal perturbation solution and terminates.*

Proof The first part of the theorem, regarding the minimal perturbation property of the reported solution, is derived from Lemmas 1 and 2, and the exhaustiveness of the **assign_SVA** function. If phase 1 ends successfully, the partial solution generated includes more SVAs than the solution with minimal perturbations found so far. The correctness of the standard *MAC* algorithm ensures that, if this partial assignment can be extended to a solution, it will be found and stored as the new solution with minimal perturbations. From Lemma 1 we know that the algorithm backtracks only if the current assignment cannot be extended to a consistent partial assignment of SVAs that includes more SVAs than in the stored solution. In other words, when no better solution can be found. Lemma 2 ensures that the algorithm would not run into infinite loops. Moreover, function **assign_SVA** attempts to assign any SVA before shifting to the second phase. Therefore, it generates all the combinations of SVAs that were not excluded by a backtrack operation. Thus, all relevant partial assignments are explored.

The termination of the algorithm follows from Lemma 2 and the finite size of the problem. □

6 Experimental evaluation

The performance of the proposed HS_MPP algorithm was empirically evaluated and compared to the algorithms by Roos et al. [11] and by Hebrard et al. [6]. Experiments were conducted on problems from two different domains—randomly generated CSPs and random instances of the Meeting Scheduling Problem (MSP). In both domains HS_MPP considerably outperformed former algorithms.

Our implementation of the algorithms follows the description in [11] and [6]. For the algorithm proposed by Roos we have explored the possible assignments from the minimal number of perturbations, and for every possible assignment we applied a MAC procedure to verify its consistency. The algorithm proposed by [6] was implemented by adding a consistency verification to the standard MAC procedure that all values in the variables' domains can be assigned without breaching the upper bound. More specifically, that by assigning a value to its variable we do not generate a perturbation that will increase the number of perturbations within the current assignment to be equal or larger to the upper bound (the smallest number of perturbations found in a complete solution so far).

6.1 Experiments on random (uniform) CSPs

Randomly generated (uniform) *CSPs* are parameterized by n variables; k values in each domain; a value p_1, which is the probability that two variables are constrained (constraint density); and a tightness value p_2, which is the probability that two values of constrained variables are in conflict. These problems are commonly used in the empirical evaluations of CSP algorithms [17].

The first set of experiments was conducted on random *CSPs*. Several setups were considered: including 20, 30, and 40 variables ($n = 20, 30$, and 40). In all setups, variables' domains included 10 values for each variable ($k = 10$). Several density values were examined, and the tightness value, p_2, was varied between 0.1 and 0.9. For every value of fixed density and tightness (p_1, p_2), 50 different random problems were generated and solved, and the evaluation metric used was the number of constraint checks (CCs). The constraint check counter was increased each time a binary constraint between two value assignments of two distinct variables, was checked. This method is commonly used for evaluating CSP search algorithms, since it is implementation independent [8, 10, 22].

In these experiments, the problems were first solved by a standard CSP algorithm. The solution to the original problem was then saved and used as input to the MPP search algorithms along with a revised version of the problem. The new problems were based on the original ones but had 1% of the constraints changed (removed and added). Next, the Dynamic CSP search algorithms were used to find the solution to the revised problems that is as similar to the input as possible. As the tightness of the problems increases, some of the original problems become unsolvable. Furthermore, the introduction of additional constraints transformed some solvable problems into unsolvable ones. For p_2 values larger than 0.6, none of the problems were solvable.

We began by evaluating Roos et al.'s algorithm (*RB-AC*) and compared its performance to that of HS_MPP. As the authors of *RB-AC* themselves note, the algorithm's performance is expected to exert a large computational effort during the search for the closest solution: "repairing a solution is much harder than creating a

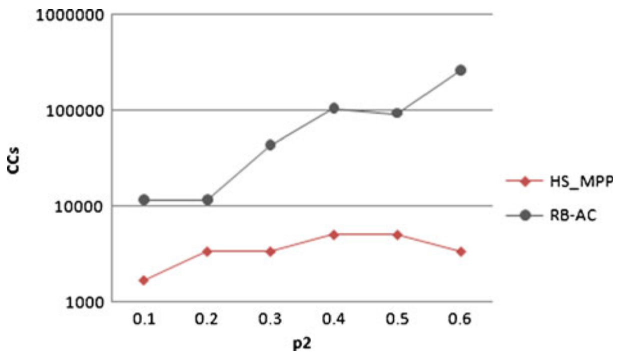


Fig. 9 Performance of hybrid search and RB-AC on random problems ($n = 20$, $p_1 = 0.3$)

new solution from scratch if many non-unary constraints are violated by the solution that needs to be repaired" [11]. Indeed, our results on the smaller problems ($n = 20$), comparing *RB-AC* and our Hybrid search, depicted in Fig. 9, assert this claim and discouraged further use of this algorithm in the following evaluations.

It is clear that the performance of *RB-AC* drastically decreases when the problems become tight, compared to the HS_MPP algorithm. This may be caused by the increased distance of the optimal solution from the initial assignment, α. One way to validate this relation between solutions' distance and the performance of the algorithm is to consider the results of the algorithm on problems sharing the same difference size (same Hamming distance) between the two solutions, as presented in Fig. 10 (using a similar setup). When the smallest distance from α of any solution to Θ is greater than two assignments, *RB-AC* becomes worse by almost an order of magnitude than the proposed hybrid search algorithm. For distance equal or larger than 5, we did not get a result in reasonable time (runs were terminated).

We next proceeded to examine larger problems, comparing the proposed HS_MPP algorithm to two alternatives. A former algorithm, proposed by Hebrard et al. [6], uses global consistency (see Section 2) and will be referred to as MPP_GAC.

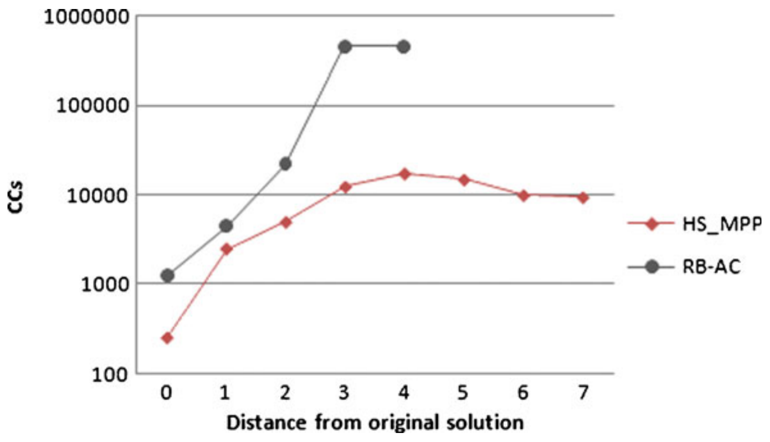


Fig. 10 Performance of hybrid search and RB-AC on random problems ($n = 20$, $p_1 = 0.3$) as a function of the solutions' distance from the initial assignment α

An alternative algorithm of a different nature can be used to establish a scale. Here we use the standard MAC algorithm [4]. It searches for *any solution* for Θ (not necessarily a minimal perturbation solution) ignoring δ and α. This alternative is used only as a scale because it does not provide an optimal solution with a minimal perturbation. The larger problems included 30 variables ($n = 30$) with 10 values ($k = 10$). We ran this setup with $p_1 = 0.3$ and varied the p_2 values over the entire range. Figure 11 depicts the results of this experiment. As the tightness value exceeded $p_2 = 0.4$ the generated problems became unsolvable. It is interesting to note that for $p_2 = 0.4$ the *CCs* count for finding a *non minimal solution* to the problem by the standard MAC algorithm is *longer* on the average than the time to find a minimal perturbation for the proposed algorithm.

Our last experiment (Fig. 12) used even larger problems—40 variables ($n = 40$, $k = 10$, $p1 = 0.1$). In this experiment the initial problems became unsolvable for $p2$ values larger than 0.6. As before, the effort in terms of CCs exerted by MAC exceeded that of our proposed algorithm as $p2$ approached it critical value ("phase transition").

These results clearly demonstrate that the performance of the proposed Hybrid search is far superior to that of RB-AC and MPP_GAC. It is quite encouraging to observe in Figs. 11 and 12 that for large problems the performance of HS_MPP is roughly within an order of magnitude of that of a non-optimal MAC and at times even better. This difference in performance is much smaller than the difference between the MPP_GAC algorithm and HS_MPP (see Fig. 11).

6.2 Experiments on Meeting Scheduling Problems (MSPs)

Following our initial results on random problems, we proceed to examine the impact of dynamic changes on structural CSPs; the specific examples used here are random instances of the Meeting Scheduling Problem (MSP). In an MSP, each variable m_i corresponds to a meeting. The domain of each variable consists of the different time slots available for that meeting. Two meetings m_i and m_j are connected by a constraint if at least one participant is required to attend both meetings. Each constraint

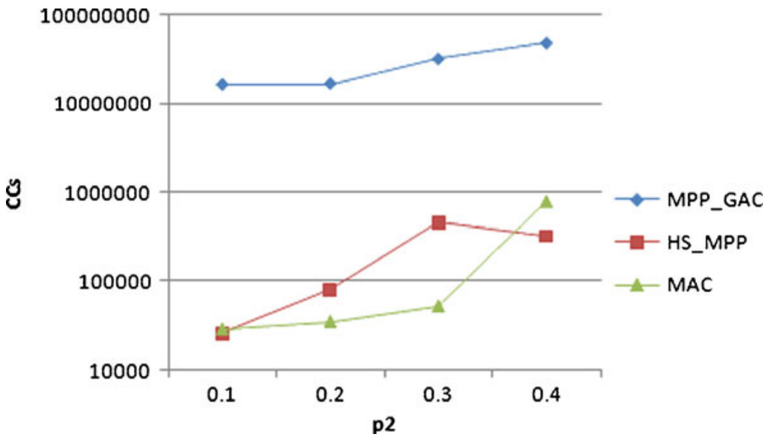


Fig. 11 Performance of algorithms on larger ($n = 30$, $k = 10$) random problems

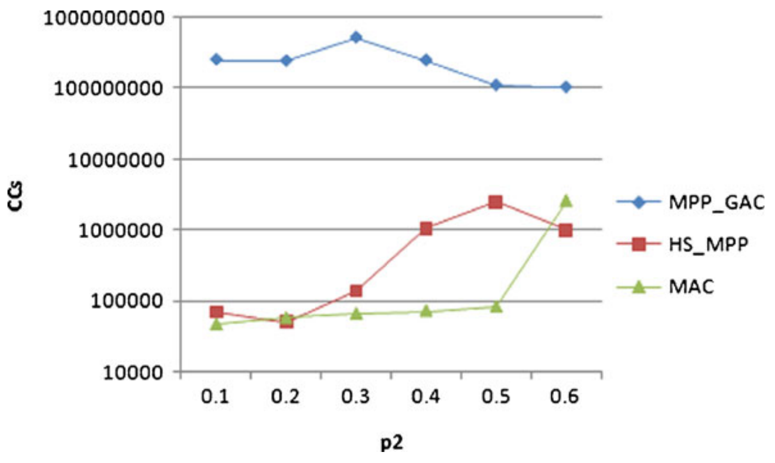


Fig. 12 Performance of algorithms on the largest ($n = 40$, $k = 10$, $p1 = 0.1$) random problems

between two meetings enforces a restriction that is termed *Arrival Constraint* [5, 21] In addition to not being held simultaneously, some meetings need a finite gap of several time slots due to the time it takes to arrive from one meeting to the other. This restriction is not necessarily symmetrical—for example, traveling from A to B by mass transportation does not take the same time as traveling from B to A.

While a change to a random CSP may be easily applied by randomly introducing prohibited pairs, the nature of an MSP problem enforces a specific structure. Our experimental evaluation of dynamic MSPs introduces two types of changes:

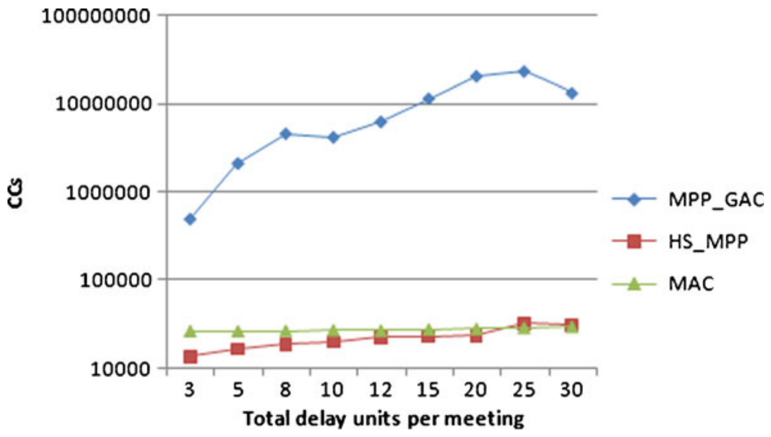
– Additional arrival constraints (due to traffic jams, train re-scheduling, etc).
– New links between meetings previously unconnected (a participant of a meeting m_i is required to attend another meeting m_j as well).

The experimental setup includes 30 meetings with ten different time slots each. The average density value, p_1, is 0.2 (i.e., participants of a given meeting also take part in six other meetings on the average).

The first setup includes additional arrival constraints. That is, following the solution of the original problem, ten meeting pairs were selected at random and a varying number of additional delay units were introduced to the constraint between the two.

Figure 13 (top) depicts the results of comparing the performance of HS_MPP to the global arc consistency algorithm of Hebrard et al. and to MAC [4].[3] The MAC algorithm simply finds a solution to the revised problem (again, with no consideration of the initial assignment α). The results in Fig. 13 show that HS_MPP is robust to additional arrival constraints. In contrast, the global arc consistency algorithm of Hebrard et al. was greatly affected by these additional constraints and required a

[3]The algorithm of Roos et al. was not compared here because it fails to complete its run within a reasonable amount of time in this setting. Our results for random problems and the results presented in [11] indicate why the algorithm proposed by Roos et al. is unsuitable for solving large problems with changed binary constraints as the problems in this set of experiments.



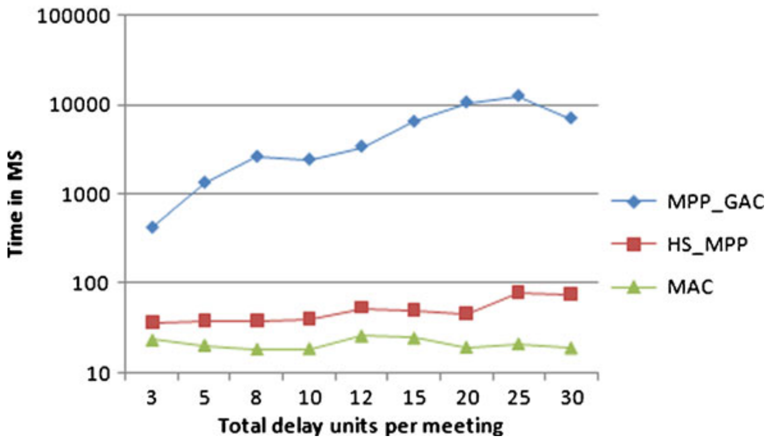
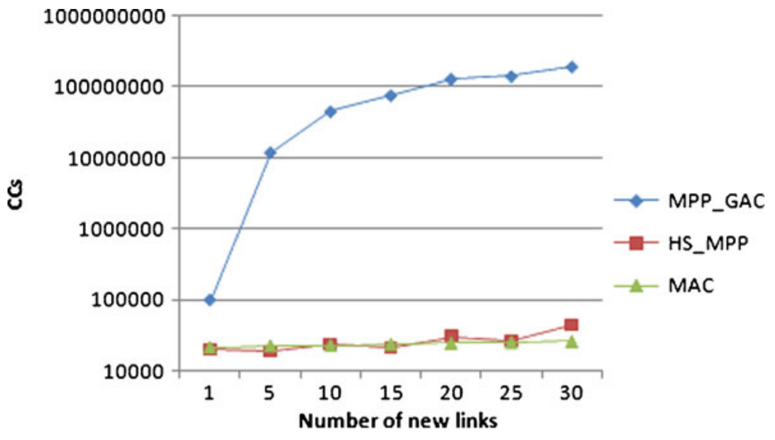


Fig. 13 Impact of adding delays to meeting pairs

computational effort (CC) that was larger by three orders of magnitude. It is also worth noting that when the minimal number of perturbations was small, HS_MPP required slightly less CCs than the MAC algorithm to find a solution.

In real world problems such as the MSP, it is often of interest to examine the actual run time. The bottom part of Fig. 13 depicts the average time to completion in the same experiment. These results verify that the main metric we use in this section (constraint checks) is valid for evaluating run-time. We assume that the small differences are caused by implementation differences. Our results demonstrate again that HS_MPP is able to produce optimal solutions considerably faster than MPP_GAC.

The second setup considered MSPs similar to those used in the first setup. Following the solution to each problem, new links were added between randomly selected meeting pairs. While the additional arrival constraints in the first setup increased the tightness value (p_2), adding new links only increased the problem's density value (p_1). The results of this experiment are depicted in Fig. 14.



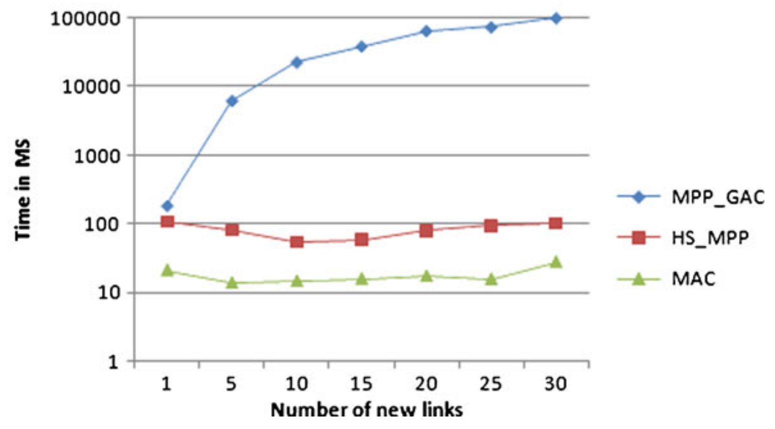


Fig. 14 Impact of adding new links between meetings

Figure 14 (top and bottom) demonstrates the robustness of HS_MPP. While the performance of MPP_GAC decreases dramatically when more links are added, HS_MPP performs the same number of CCs (roughly the same length of time) in order to find the most similar solution. This time is up to four orders of magnitude faster than MPP_GAC. The growth in the number of links between meetings is correlated with the average minimal distance to α (i.e., when a single meeting pair was connected the average distance was 1, when 5 meeting pairs were connected it was 2.12, when ten pairs were connected it was 3.32, followed by 4.2, 4.76, 5.84, and 6.44). While the increased minimal distance severely degraded the performance of MPP_GAC, the proposed HS_MPP maintained a near constant number of CCs throughout this experiment. Once again, both metrics—CCs and run-time—produced similar results.

Figures 15 and 16 present experimental results for much larger MSPs. These MSPs included 60 variables (meetings) with five time-slots each. As before, the trends indicate that HS_MPP is able to produce the optimal result with less computational effort even when the problem instances are significantly larger.

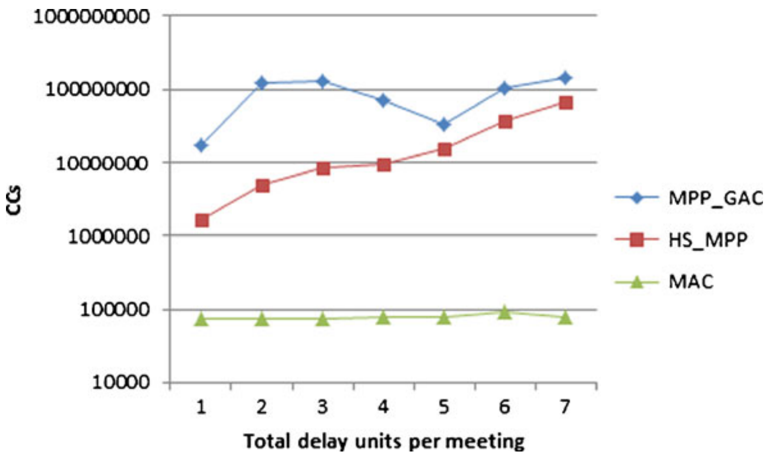


Fig. 15 Adding delays to meeting pairs in large problems

6.3 Discussion

The experimental study of compares three very different algorithms for solving the minimal perturbation problem. The first, *RB-AC*, proposed by Roos et al., searches the assignments according to their distance from the initial assignment α [11]. Thus, the first solution to be found is the optimal solution. This approach was found to be effective only when the tightness of the problem (and the resulting difference between the original and the new solution) is very small. The second algorithm is based on Branch and Bound and enforces global arc consistency on a soft constraint of similarity [6]. The Branch and Bound algorithm is evidently much faster than the best first search of Roos et al. However, forcing global arc consistency can be an overhead in terms of computation mainly because the optimality of a new solution

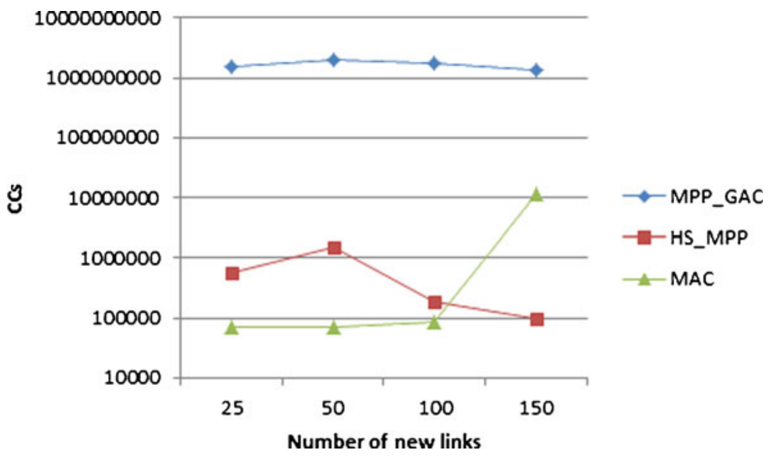


Fig. 16 Adding new links between meetings in large problems

is determined only by the value assignments that were included in α (the SVAs). The Hybrid Search algorithm proposed in the presented study avoids the overhead of enforcing global arc consistency on the entire CSP. It separates the search into an optimization phase and a satisfaction phase. The optimization phase can include only one value for each variable (the value assigned to it in α). The search space of the first phase (the optimization phase) in the proposed algorithm is defined by the number of consistent combinations of SVAs. The second phase, which considers the entire domain of variables, is performed only when a relevant partial solution is found by the first phase. It needs to consider only values in domains of unassigned variables. In addition, it solves a constraint satisfaction problem, which is easier than a constraint optimization problem. This results in a significant improvement in performance. In many cases of the experimental evaluation, the improvement in run-time is by an order of magnitude and more.

7 Conclusion

A hybrid search algorithm for finding a solution to the minimal perturbations problem was presented. The proposed algorithm performs optimization and satisfaction phases alternately. The optimization phase is performed on a much smaller search space than that of the complete problem. The satisfaction phase is used to determine if an assignment with a potential to be optimal can lead to a complete solution. Our experimental study demonstrates the great potential of this approach in comparison to best first search and to the use of global arc consistency over soft constraints.

The proposed algorithm not only outperformed other algorithms that searched for the optimal solution, but in some cases it also outperformed a standard MAC algorithm that searched for any solution to the satisfaction problem. This encourages further investigation of the proposed algorithm in wider scopes of dynamic AI search problems.

References

1. Barták, R. (1998). *Guide to constraint programming*. http://kti.mff.cuni.cz/~bartak/constraints/.
2. Barták, R. (1999). Dynamic constraint models for planning and scheduling problems. In *New trends in constraints* (pp. 237–255).
3. Barták, R., Muller, T., & Rudova, H. (2004). A new approach to modeling and solving minimal perturbation problems. In *Recent advances in constraints* (pp. 233–249). Berlin: Springer.
4. Bessiere, C., & Regin, J. C. (1996). MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. second international conference on principles and practice of constraint programming, CP 96* (pp. 61–75). Cambridge, MA.
5. Gent, I. P., & Walsh, T. (1999). *CSPLib: A benchmark library for constraints*. Technical report, APES-09-1999, 1999. Available from http://csplib.cs.strath.ac.uk/.
6. Hebrard, E., Hnich, B., O'Sullivan, B., & Walsh, T. (2005). Finding diverse and similar solutions in constraint programming. In *The twentieth national conference on artificial intelligence, AAAI-2005*. Pittsburgh, PA, USA.
7. Hebrard, E., O'Sullivan, B., & Walsh, T. (2007). Distance constraints in constraint satisfaction. In *The twentieth international joint conference on artificial intelligence, IJCAI-2007*. Hyderabad, India.
8. Kondrak, G., & van Beek, P. (1997). A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence, 21*, 365–387.

9. Meisels, A., & Kaplanski, E. (2002). Scheduling agents—Distributed employee timetabling. In *Proc. 4th conf. on autom. timetabling, PATAT-2002* (pp. 166–80). Ghent, Belgium.
10. Prosser, P. (1996). An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence, 81*, 81–109.
11. Roos, N., Ran, Y., & van den Herik, H. J. (2000). Combining local search and constraint propagation to find a minimal change solution for a dynamic csp. In *Artificial intelligence: Methodology, systems, applications* (pp. 272–282).
12. Roos, N., Ran, Y., & van den Herik, H. J. (2002). 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).
13. Russell, S., & Norvig, P. (2005). *Artificial intelligence, a modern approach* (2nd ed.). Englewood Cliffs: Prentice-Hall.
14. Sakkout, H. E., Richards, T., & Wallace, M. (1998). Minimal perturbation in dynamic scheduling. In *Proc. 13th European conference on artificial intelligence, ECAI-98* (pp. 504–508). Brighton.
15. Sakkout, H. E., & Wallace, M. (2000). Probe backtrack search for minimalperturbation in dynamic scheduling. *Constraints, 4*(5), 359–388.
16. Schiex, T., & Verfaillie, G. (1994). Nogood recording for static and dynamic constraint satisfaction problem. *International Journal on Artificial Intelligence Tools (IJAIT), 3*(2), 187–207.
17. Smith, B. M. (1996). Locating the phase transition in binary constraint satisfactionproblems. *Artificial Intelligence, 81*, 155–181.
18. Verfaillie, G., & Jussien, N. (2005). Constraint solving in uncertain and dynamic environments—A survey. *Constraints, 10*(3), 253–281.
19. Verfaillie, G., & Schiex, T. (1994). Solution reuse in dynamic constraint satisfaction problems. In *Twelfth national conference on artificial intelligence, AAAI-1994* (pp. 307–312).
20. Wallace, R. J., & Freuder, E. (2002). Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In *Proc. 3rd workshop on distributed constrait reasoning, DCR-02* (pp. 176–182). Bologna.
21. Wallace, R. J., & Freuder, E. (2005). Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence, 161*(1–2), 209–228.
22. Zivan, R., & Meisels, A. (2006). Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI), 46*, 415–439.