An Improved Search Algorithm for Min-Perturbation

Alex Fukunaga

The University of Tokyo

Abstract. In many scheduling and resource assignment problems, it is necessary to find a solution which is as similar as possible to a given, initial assignment. We propose a new algorithm for this minimal perturbation problem which searches a space of variable commitments and uses a lower bound function based on the minimal vertex covering of a constraint violation graph. An empirical evaluation on random CSPs show that our algorithm significantly outperforms previous algorithms, including the recent two-phased, hybrid algorithm proposed by Zivan, Grubshtein, and Meisels.

1 Introduction

In many CP applications it is necessary to find solutions that are as similar as possible to a given, initial assignment of values to variables. For example, in a meeting scheduling problem or resource scheduling problem, constraints can change unexpectedly after a solution has been generated. This is a type of dynamic constraint satisfaction problem. Similarly, there are situations where there is an "ideal" (but possibly infeasible) assignment of values to variables for a CSP, and the goal is to find an assignment which differs as little as possible from the target. Another scenario where a solution similar to a given initial state is desired occurs in staff scheduling. Employees express preferences regarding when they want to work, but their preferences must be balanced against the staffing demands and constraints of the business, requiring a schedule that satisfies staffing requirements while deviating minimally from employee preferences.

This paper considers search algorithms for this class of *minimal perturbation problem* (MPP) for CSPs, where we seek a solution that minimizes the number of variables whose values differ from a target assignment, or equivalently, the minimal number of variable changes that are necessary to a CSP solution when some of the constraints change unexpectedly. In particular, we focus on minimal perturbation for binary CSPs. Previously, Ran et al. proposed an iterated deepening algorithm for the MPP that searches the space of variable assignments that differ from the target/initial assignment by at most d assignments, where d is the iterative deepening bound [9]. More recently, Zivan, Grubshtein, and Meisels proposed a two-phased algorithm that interleaves the problem of bounding the number of necessary perturbations from the initial assignment, and the problem of testing if such an assignment is possible [11].

We propose a new search algorithm for the MPP, where the main features are (1) a search space where nodes represent a set of *committed* variable assignments, (2) a lower bound based on the minimal vertex covering of the current set of violated constraints, which dominates the lower bound by Zivan et al. This generalizes an earlier, domain-specific MPP algorithm proposed in [4].

2 Problem Definition and Preliminaries

The *Minimal Perturbation Problem* (MPP) is defined as follows: Let $C = (V, D, C)$ be a CSP, where $V = v_1, ..., v_n$ is a set of variables, $D = D_1, ..., D_n$ is a set of domains where D_i is a finite discrete set of possible values for variable v_i, and $C = c_1, ..., c_m$ is a set of constraints which restricts the set of values that the variables can be simultaneously assigned.

Let I be a complete assignment for C. The objective of the MPP is to find an assignment A such that all of the constraints are satisfied, and the *number* of variables in A whose value differs from I is minimized. Following [11], the value of variable v in the original assignment is called the *Starting Variable Assignment* of v, or its SVA.

While previous work [9, 11] defined the MPP more generally, i.e., a general distance function, and a partial initial assignment, the lower bound functions used in the previous work assume the definition above, and the actual experimental evaluations of the previous algorithms were performed on binary CSPs based on this definition.

3 Previous Algorithms for the MPP

The first algorithm which specifically addressed the MPP defined in Sec 2 was the *Repair-Based algorithm with Arc-Consistency (RB-AC)*, by Ran et al. [9]. Given an initial variable assignment $I = \{x_1 = v_1, ..., x_n = v_n\}$, let D_i be the set of states which have exactly i variables whose value are different from that of the initial state I. We call the set $D = D_1 \cup ...D_n$ the *difference space*, or *D-space*. The root node of this search space is I. Nodes at depth d of the search tree contain variable assignments which differ by d assignments from I. Each edge in the tree changes the value of one variable which has not yet been changed by any ancestor. RB-AC searches D-space using a depth-first iterative deepening strategy, IDA* [7]. The d-th iteration of IDA* explores the subset of the depth-first branch-and-bound D-space search tree where at each node, the sum $f = g + h \leq d$, where g is the number of differences from the initial state in the current solution, and h is the lower bound on the additional number of differences required to find a conflict-free solution. RB-AC uses a simple lower bound, L_1, which is the number of variables that do not have the SVA in its domain.

Zivan, Grubshtein, and Meisels proposed HS_MPP, a "hybrid" search algorithm for the MPP [11]. Their algorithm consists of two, interleaved phases: The first phase performs branch-and-bound on a binary search tree where each node represents a variable, and the branches correspond to a decision regarding whether to assign the variable to the same value as in the initial assignment. At each node, HS_MPP-Phase1 computes a lower bound on the number of perturbations, and prunes the search if this exceeds or equals the current upper bound. Then, v, variable such that $SVA(v) \in dom(v)$ is selected. If there is no such variable (i.e., all remaining variables must be perturbed), then HS_PP-Phase2, described below, is called to test for feasibility. Otherwise, HS_MPP branches: The left branch assigns v its SVA and recursively searches the remaining variables; the right branch of the binary search tree, HS_MPP eliminates the SVA from the domain of v, and recursively searches the remaining variables.

The HS_MPP algorithm uses a lower bound, which we denote L_Z, to prune the branch-and-bound tree in HS_MPP-Phase1. This bound improves upon L_1 by exploiting the fact that if there is a pair of variables which have the SVA in the domain, but the SVAs conflict with each other, then one of these variables must be assigned a non-SVA, so the bound can be increased relative to L_1 by accounting for such pairs (see [11]).

After each decision in Phase 1, the following, limited filtering function is applied: For each remaining variable v, $SVA(v)$ is removed from $domain(v)$ if $SVA(v)$ is inconsistent with the current assignment of SVAs.

HS_MPP-Phase2 applies a standard MAC (maintaining-arc-consistency) algorithm to the remaining variables (i.e., variables which do not have the SVA in the domain and must be perturbed). If the MAC algorithm finds a satisfying assignment of values to v_r, then this is a solution to the MPP.

Finally, a third previous approach is by Hebrard, O'Sullivan and Walsh, who proposed a GAC for distance constraints [6]. Zivan et al compared HS_MPP to this GAC method and showed that HS_MPP performed significantly better on random binary CSPs (30-40 variables) and meeting rescheduling problems.

Related Work

Other previous work has addressed problems that are related to (but different from) the MPP formulation treated in this paper. A Dynamic CSP is a sequence of constraint satisfaction problems where each instance is derived from the previous instance by modifying some constraints [2]. Verfaillie and Schiex solved Dynamic CSPs by repairing the solution to the previous CSP instance [10]. They proposed a depth-first backtracking algorithm in D-space. Since the goal is to solve the Dynamic CSP instance, there is no mechanism to guarantee minimal perturbation, although they incorporate variable ordering heuristics that tend to bias the search towards a minimal perturbation solution. El Sakkout and Wallace [3] investigated a minimal cost repair problem for scheduling. They consider difference functions that can be expressed linearly (our MPP difference count objective is nonlinear). Their probe backtracking algorithm does not explicitly consider the initial schedule, and reschedules from scratch [3]. Barták et al. investigated overconstrained CSPs for which there is likely to be no feasible solution without violated constraints [1], and studied methods to seek a maximal assignment of consistent variables which also differs minimally from an initial state. They also studied an iterative repair (local search) algorithm biased to seek minimal perturbation solutions for course timetabling [8].

4 A Commitment-Space Search Algorithm for the MPP

We now describe our algorithm for the MPP. Unlike RB-AC, which searches D-space, and HS_MPP, which searches a 2-phase search in the space of variable assignments, our algorithm searches the space of variable commitments.

In a *commitment-based search space* (C-space) for the MPP, each node in the search tree represents a complete assignment of values to variables, where some subset of the variables are committed to their current value. Edges in the search tree represent a decision to commit a variable to some value. For each variable, we represent its current value, as well as whether a commitment has been made to the value. The root node of this search space is the initial assignment I. We say that a variable x is *committed* to value v at node N if x is assigned to v at N and every descendant of N, and *uncommitted* otherwise.

Each node represents the result of committing some variable to a particular value. Thus, this search space has a branching factor of d, the domain size, and a maximum depth of n, the number of variables. We originally proposed C-space for minimal perturbation in [4]. However that previous work focused on a specific type of MPP (bin packing constraint repair e.g., virtual machine reassignment in data centers), and C-space has not been evaluated for standard, domain-independent binary CSPs. C-space has a narrower structure (smaller branching factor) compared to D-space, at the cost of some redundancy. See [4] for an analysis, as well as a figure illustrating example search trees.

We evaluated both a standard depth-first branch-and-bound strategy, as well as an iterative deepening (IDA*) strategy [7] for C-space. Although iterative deepening can repeatedly visit the same state, in cases where the minimal perturbation solution is close to the initial solution, the IDA* search strategy would be expected to be faster than depth-first branch-and-bound.

For both of these strategies, a standard, most-constrained variable ordering is used, and a min-conflicts (with respect to the original values in the initial assignment) value ordering is used. At each node, arc consistency (AC-3) is applied for filtering. The depth-first branch-and-bound version is shown in Algorithm 1.

Lower Bound

The new lower bound function is based on a constrained vertex covering of a constraint violation graph. At every node in the search tree, there is a non-empty set of violated constraints. Given the set of all violated constraints, we construct a *constraint violation graph* G where each variable corresponds to a vertex in G, and there is an edge between vertex v_i and v_j if a constraint between variables x_i and x_j is violated. A vertex cover (VC) of a graph is a subset $vc \subset V$ of the vertices such that for every edge $e = (v_a, v_b)$ in G, either $v_a \in vc$ or $v_b \in vc$. A minimal vertex cover of G is a covering of G which has minimal cardinality.

The minimal VC of a constraint violation graph is clearly a relaxation of the MPP. The minimal VC identifies a subset of variables that could possibly eliminate all constraint violations, without identifying the actual values that must be assigned. The covering has one additional constraint: variables which no longer have the SVA in the domain are forced to be included in the covering. Thus, the cardinality of the (constrained) minimal VC is a lower bound on the number of perturbations required to result in a conflict-free assignment of values to variables. It is easy to see that this bound dominates the L_Z bound [11].

Although computing a minimal vertex cover is NP-complete [5], computing the minimal VC of a graph is much easier than solving the MPP (the search space is a binary tree with depth $= \#vars$, as opposed to a tree with branch factor $|Domain|$ for the MPP C-space search), so the minimal VC can be used as the basis for a lower bound. Our current implementation performs a straightforward branch-and-bound search where each node determines whether a variable is included or excluded from the cover. A simple filtering/pruning rule is used: for every edge (v_a, v_b), if v_a is excluded from the covering, then v_b must be included; conversely, if v_b is excluded, then v_a must be included. No other lower-bounding techniques or optimizations are used in the minimal VC computation, but as shown below, this simple implementation suffices in practice.

Algorithm 1 C-space Search Algorithm

```
mpp_search(uncommittedVars,committedVars,numChanges)
if get_conflicts(uncommittedVars,committedVars)==$\emptyset$ then
  if count_num_perturbations(committedVars) $<$ minimalChanges then
    minimalChanges = count_num_perturbations(commitedVars) {replace best-so-far solution}
  return success
if lowerbound(uncommitedVars,committedVars) $>$ minimalChanges then
  return failure {pruning based on lower bound}
V = select(uncommittedVars)
for all val in Order(domain(V)) do
  commit(V,val) {commitment also applies filtering (arc-consistency)}
  r = mpp_search(uncommittedVars $\setminus$ V, committedVars $\cup$ V)
  if r==success then
    return success
return failure
```

5 Experimental Evaluation

We evaluated the performance of the MPP algorithms using problems derived from standard, randomly generated binary CSPs. The classes of MPPs used in the experiments are defined by 5 parameters (n, k, p_1, p_2, δ). The first 4 parameters are used to first generate a random, uniform binary CSP, C, where n is the number of variables, k is domain size of all variables, p_1 is the constraint density (probability that any 2 variables have a constraint), and p_2 is the tightness (probability that any 2 values in a pair of constraint variables are a nogood). Then, C is solved using a standard CSP solver. If C is unsatisfiable, then it is discarded. If C is satisfiable, then the solution that is found is used as I, the initial assignment for the MPP. Then, C is perturbed by replacing some fraction δ of the constraints, resulting in a perturbed CSP C', and the MPP instance is (C', I). For n=30 and 40 variables, we generated 30 candidate binary CSPs each for all combinations of p_1, p_2, δ, where $p_1 \in 0.3, 0.4, 0.5, 0.6, 0.7$, $p_2 \in 0.3, 0.4, 0.5, 0.6, 0.7$, and $\delta \in 0.05, 0.10, 0.25, 0.50, 0.75, 1.00$. All of these were tested for solvability using a standard CSP solver (i.e., whether there is any satisfying assignment, regardless of distance from the initial configuration I). Of these, 2676 of the 30-variable instances and 1578 of the 40-variable instances were satisfiable MPPs. Similarly, for n=50 variables, we generated 30 candidate MPPs for all combinations of p_1, p_2 taken from $p_1 \in 0.3, 0.4, 0.5$, $p_2 \in 0.3, 0.4, 0.5$, $\delta \in 0.05, 0.10, 0.25, 0.50, 0.75, 1.00$. and 936 were satisfiable.

In the experiments below, we compare the algorithms in such a way that only these solvable instances matter, i.e., comparisons of the time to find solutions, with a time limit of 900 seconds. This is because unsolvable instances can be detected by running a standard CSP solver much more quickly than any of the MPP algorithms (clearly, checking satisfiability is a simpler problem than seeking a minimal perturbation). Our new algorithm is at least as fast as the previous algorithms in detecting unsatisfiability. In practice, the best strategy would be to first run a standard CSP to check for satisfiability, then run a dedicated MPP solver to minimize the perturbations.

We compared the following algorithms:

- C-space/L_{vc} - our C-space search algorithm using the new L_{vc} lower bound and depth-first branch-and-bound.
- C-space/L_{vc}/ID - Iterative Deepening C-space search algorithm using the L_{vc} lower bound.
- HS_MPP - The hybrid algorithm by Zivan et al [11].
- RB-AC/L_{vc} - A modified version of RB-AC algorithm by Ran et al [9], which uses use our L_{VC} lower bound instead of the L_1 bound [9] and searches D-space using iterative-deepening.
- C-space/L_Z - C-space search algorithm using the L_Z lower bound [11]. This comparison isolates the effect of the lower bound function L_{vc} compared to L_z.

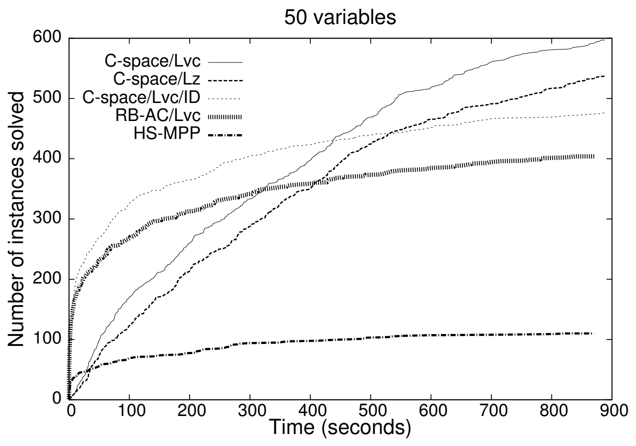


Fig. 1: n=50, cumulative number of problems that can be solved after a given time

Each algorithm was executed on each of the 30, 40, and 50-variable random binary MPP instances, with a 900 second time limit per run. Note that although we focus on runtime due to space restrictions, comparisons of the number of backtracks and constraint checks are qualitatively similar to the runtime results.

Figure 1 shows an overall comparison of the MPP algorithms, and plots the cumulative number of problems solved (y-axis) as the amount of time increases (x-axis) by each algorithm for the 50-variable problems. For example, the C-space/L_{vc} algorithm solved around 500 instances within 500 seconds. Overall, C-space/L_{vc} performed best on the hardest instances (which require > 400 seconds), while C-space/L_{vc}/ID performed best on problems requiring less than 400 seconds. Another interesting result is that RB-AC with the L_{vc} bound performs significantly better than HS_MPP, suggesting that the success of HS_MPP compared to the original RB-AC algorithm was due much more to the lower bound than to the hybrid search strategy.

While results for 30 and 40 variable problems are not shown due to space, they look similar, except that C-space/L_{vc}/ID performs relatively better with fewer variables.

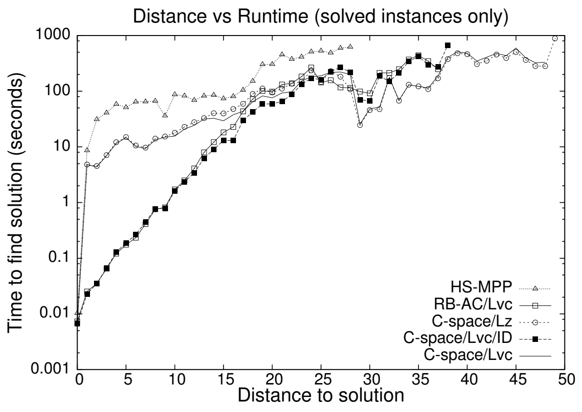


Fig. 2: Effect of Distance from Initial/Target State

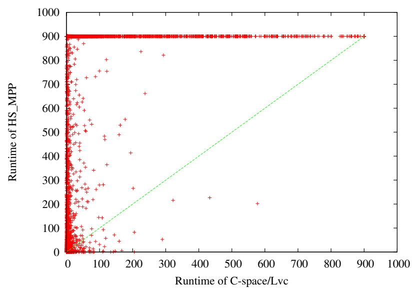
Figure 2 plots average runtime required to solve instances as a function of the distance of the solution found to the initial assignment. This only includes successful runs and excludes runs that timed out, so some of the lines appear truncated (otherwise, for the less successful algorithms, it is difficult to see the impact of distance because there were so many failed runs). Overall, if the distance to a solution is within 10-15 variable assignment changes (i.e., the amount of repair required is small), the faster algorithms such as C-space/L_{vc} can solve the problems within 10 seconds (if at all).

Figure 3 compares key pairs of MPP algorithms on all of the 30, 40, and 50 variable problems. Each figure plots the runtimes for all instances on a pair of algorithms A_1, A_2, where the x-coordinate is the runtime of A_1 on the instance, and the y-coordinate is the runtime of A_2. An x or y value of 900 indicates failure to solve the instance. The straight diagonal line is (x=y), i.e., points above the line indicate that C-space/L_{vc} performed better, while points below the line indicate that the other algorithm performed better.

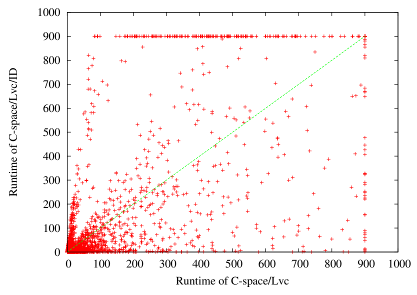
Figure 3a shows that C-space/L_{vc} clearly outperforms HS_MPP, the previous state-of-the art algorithm. The average ratio of runtimes for HS_MPP vs C-space/L_{vc} is 86.48 for all problems that were solved by at least 1 of these solvers, and 160.35 for problems that took more than 60 seconds for the faster solver on each instance.

Figure 3d compares C-space/L_{vc}/ID and RB-AC/L_{vc}. These two algorithms, which both use iterative deepening search and the same lower bound (L_{vc}) differ mainly in the choice of search space (C-space and D-space, respectively). Figure 3d shows that C-space/L_{vc}/ID clearly outperforms RB-AC/L_{vc} on almost every problem instance, suggesting that C-space is better structured for search than D-space. However, the advantage of C-space over D-space seems to be less pronounced for this class of benchmarks compared to the virtual machine reassignment problem in [4].

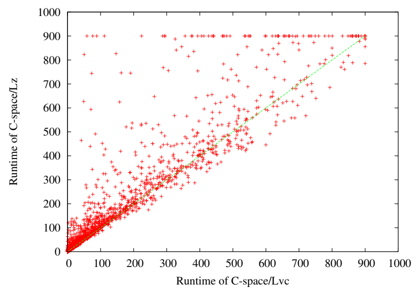
Figure 3b compares C-space/L_{vc} and C-space/L_Z. Combined with Figures. 1, and 2 the results show that iterative deepening is a good strategy for quickly solving relatively



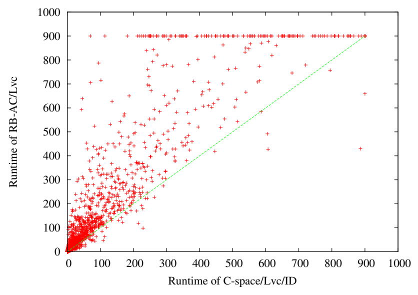
(a) Comparison with previous state-of-the-art (C-space/L_{vc} vs. HS_MPP)



(b) Effect of iterative deepening (C-space/L_{vc} vs. C-space/L_{vc}/ID)



(c) Impact of new lower bound L_{vc} (C-space/L_{vc} vs. C-space/L_Z)



(d) Impact of search space (C-space/L_{vc}/ID vs. RB-AC/L_{vc})

Fig. 3: Pairwise comparison of MPP algorithms (includes all 30,40, and 50 variable problems)

easy problems (problems where the distance from I to a solution is small); however, for harder problems (where the distance from I to a solution is large), straightforward depth-first branch-and-bound seems to be a more robust choice.

Figure 3c compares C-space/L_{vc} and C-space/L_Z. The results show that the new vertex-cover based lower bound L_{vc} clearly outperforms the previous lower bound L_Z by Zivan et al [11]. The average ratio of runtimes using lower bound L_Z vs L_{vc} is 1.73 for all instances solved by at least one solver, and 2.10 for instances that required 60 seconds or more for the faster solver.

6 Discussion and Conclusions

We proposed a search algorithm for optimal solutions to the min-perturbation problem. Our main contributions are: (1) We showed that our new CSpace/L_{vc} algorithm significantly improves upon the previous state of the art (HS_MPP) for random binary CSPs generated with a wide range of parameters. (2) We showed that both L_{vc}, the new lower bound for the MPP based on vertex covering of the constraint graph, as well as the C-space search space contribute significantly to the performance of the new algorithm (Fig. 3). Future work includes evaluation on applications such as employee shift rescheduling and meeting rescheduling.

References

1. Barták, R., Müller, T., Rudová, H.: A new approach to modeling and solving minimal perturbation problems. In: Recent Advances in Constraints. LNAI, vol. 3010, pp. 233–249. Springer-Verlag (2004)
2. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: Proc. AAAI. pp. 37–42 (1988)
3. El-Sakkout, H., Wallace, M.: Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5, 359–388 (2000)
4. Fukunaga, A.: Search spaces for minimal perturbation repair. In: Proceedings of CP. pp. 383–390 (2009)
5. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
6. Hebrard, E., O'Sullivan, B., Walsh, T.: Distance constraints in constraint satisfaction. In: Proc. IJCAI. pp. 106–111 (2007)
7. Korf, R.: Depth-first iterative-deepening: an optimal admissible tree search. Artificial Intelligence 27(1), 97–109 (1985)
8. Müller, T., Rudová, H., Barták, R.: Minimal perturbation in course timetabling. In: PATAT Selected papers, LNCS vol.3616. pp. 126–146. Springer-Verlag (2005)
9. Ran, Y., Roos, N., van den Herik, H.: Approaches to find a near-minimal change solution for dynamic CSPs. In: Proc. CP-AI-OR. pp. 378–387 (2002)
10. Verfaillie, G., Schiex, T.: Solution reuse in dynamic constraint satisfaction problems. In: Proc. AAAI. pp. 307–312. Seattle, Washington (1994)
11. Zivan, R., Grubshtein, A., Meisels, A.: Hybrid search for minimal perturbation in dynamic CSPs. Constraints 16, 228–249 (2011)