Search Spaces for Min-Perturbation Repair

Alex S. Fukunaga**

Global Edge Institute, Tokyo Institute of Technology, Meguro, Tokyo, Japan
fukunaga@is.titech.ac.jp

Abstract. Many problems require minimally perturbing an initial state in order to repair some violated constraints. We consider two search spaces for exactly solving this minimal perturbation repair problem: a standard, difference-based search space, and a new, commitment-based search space. Empirical results with exact search algorithms for a min-cost virtual machine reassignment problem, a minimal perturbation repair problem related to server consolidation in data centers, show that the commitment-based search space can be significantly more efficient.

1 Introduction

Most research on constraint satisfaction and optimization focus on generating solutions from scratch – given a set of variables and constraints, generate an (optimal) assignment that satisfies the constraints. In practice, there are many situations where it is necessary to find solutions to constraint satisfaction problems that are as close as possible to a given, initial state. One example is related to *server consolidation*, the use of virtual machines to consolidate multiple servers onto fewer servers [15]. There is currently great interest in server consolidation due to opportunities for improved energy efficiency and cost reduction. Server consolidation can be modeled as a bin packing problem [8]. Consider a set of n virtual machines (VMs), where each VM has a *weight* (representing resource demand). Given m physical servers, each with a some *capacity* (representing aggregate CPU/RAM resources) the server consolidation problem is the problem of assigning the n VMs to the m physical servers, such that each VM is assigned to exactly one physical server, and for every physical server, the sum of the assigned VM demands is within the capacity of the physical server.

Suppose that after the initial server assignments are made, the resource requirements of the VMs deviate from the original forecasts, resulting in some servers being overloaded. VMs can be reassigned among the servers in order to rebalance the loads. However, migration of a VM between servers incurs costs (e.g., system administration costs, possible downtime for a service). The *Min-Cost Virtual Machine Reassignment Problem* (VMRP) seeks a new assignment of VMs to servers such that no server is overloaded, and the number of jobs that are moved from their initial assignment is minimized. Heuristics for this problem were investigated in [2]. An approximation for a similar problem was considered in [1]. Similar problems arise for process migration in distributed systems.

** This research supported by JSPS, JST, MEXT, and the Okawa Foundation.

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 general class of *Min-Perturbation Repair Problems* (MPRP) seeks to *repair* an initial assignment of values to variables (i.e., find a conflict-free solution), with minimal *cost*, where cost is the number of differences between a candidate solution and the initial assignment. While a standard CSP seeks a solution which does not violate any constraints, the MPRP imposes the additional goal of minimizing the distance from an initial assignment of values to variables.

This paper considers search algorithms for the MPRP, focusing on alternative search spaces. In Section 2, we consider a standard, difference based search space which has been used in previous work on the MPRP, as well as a new, commitment-based search space. Using the VMRP as a case study, we experimentally evaluate these search spaces (Section 3). We discuss related work in Section 4, and conclude in Section 5

2 Search Spaces for Minimal Perturbation Repair

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. A standard depth-first branch-and-bound (DFBNB) can be applied to explore this search space. Problem-specific pruning techniques, such as those described in Sec 3, are applied at each node.

Instead of a depth-first search strategy, we can also use a best-first search strategy, such as Iterative-Deepening A* (IDA*) [11], which expands nodes in a best-first order using linear space (at the cost of reopening some nodes). The admissible heuristic function, h, used by IDA* is the same as the lower bounding function used for DFBNB, and the d-th iteration of IDA* explores the subset of the DFBNB 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. Ran et al [13] applied iterative-deepening in D-space to solve a minimal perturbation problem for binary CSPs.

We now introduce *commitment*, a useful concept for MPRP search algorithms. 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. For variables $x_1, ..., x_n$, we denote a search state as $S = \{x_1 = v_1, ..., x_n = v_n\}$, or more concisely, $\{v_1, ..., v_n\}$. Furthermore, the values are underlined if the variable is committed to that value. In a 2-variable MPRP where the current assignments

are $x_1 = 1, x_2 = 2$, and we have committed $x_1 = 1$, we can denote this state as $\{\underline{x_1 = 1}, x_2 = 2\}$, or more concisely, $\{\underline{1}, 2\}$. At the root node of this search space, the variables are assigned the values of the initial assignment I, and all variables are uncommitted. Furthermore, we annotate a value to be different from the initial state I with an asterisk (*). Thus, $\{\underline{1}, \underline{3}^*, 2\}$ denotes a state where x_1 is committed to value 1, x_2 to 3, and x_3 is uncommitted (but assigned to 2), and where the value of x_2 differs from the value of x_2 in the initial state. Figure 1 shows a search tree for the VMRP. The sibling nodes are ordered according to a variable ordering implemented in the search algorithm (lex order in Fig. 1).

One issue with the D-space search tree is symmetry, e.g., given two variables x_1 and x_2, assigning $x_1 = 2$ first, followed by $x_2 = 1$, is symmetric to assigning $x_2 = 1$, then $x_1 = 2$. One approach to eliminating such symmetries is a standard nogood-based approach [13]. We use a different approach, which eliminates symmetries by asserting commitments on the siblings of a node. For example, in Figure 1, node B is the result of starting with the initial state A and moving the VM with weight 20 from $S_{c=10}$, the server with capacity 10, to $S_{c=25}$, the server with capacity 25. For all of the siblings of node B, we commit this VM to $S_{c=10}$ (i.e., forbid moving it to $S_{c=25}$). This is equivalent to saying that assigning the VM with weight 20 to $S_{c=10}$ is nogood for all siblings and descendants of B. Thus, a separate representation for nogoods is unnecessary. This method generalizes straightforwardly when there are more than 2 possible values. [1]

An explicit notion of commitment (as opposed to just assignment) in a MPRP is very useful for purposes other than symmetry elimination. For example, in a VMRP, if we have committed a VM with weight 8 to a server S with capacity 10, then all nodes which assign more than 2 additional units of demand to S can be pruned. Note that merely *assigning* a VM with weight 8 (for example, in the initial assignment I) does not allow the same pruning, because it is possible to move that VM out of S, allowing another VM with demand greater than 2 to be assigned to S. It is the *commitment* which allows us to prune. Similarly, note that this type of pruning is not captured by the nogoods used in [13]. Commitments also constrain the feasible domains for a variable, which has a significant impact on the effectiveness of variable ordering (we use most-constrained ordering).

A new, alternative search space for searching the space of commitments, rather than differences, is a *commitment-based search space* (C-space), where each node in the search tree represents a partially committed assignment of variables to values, and edges represent a commitment of 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 I. Nodes at depth d of the search tree contain variable assignments with d commitments.[2] A sample C-space search tree is shown in Figure 1. Each edge represents a single commitment. While all edges in D-space have cost 1, some

[1] Another class of symmetries, not handled by nogoods or commitments, arises with low-precision instances, e.g., multiple VMs with the same weight; this is future work.

[2] In D-space, nodes at depth d can contain more than d commitments because of the commitments asserted for symmetry elimination.

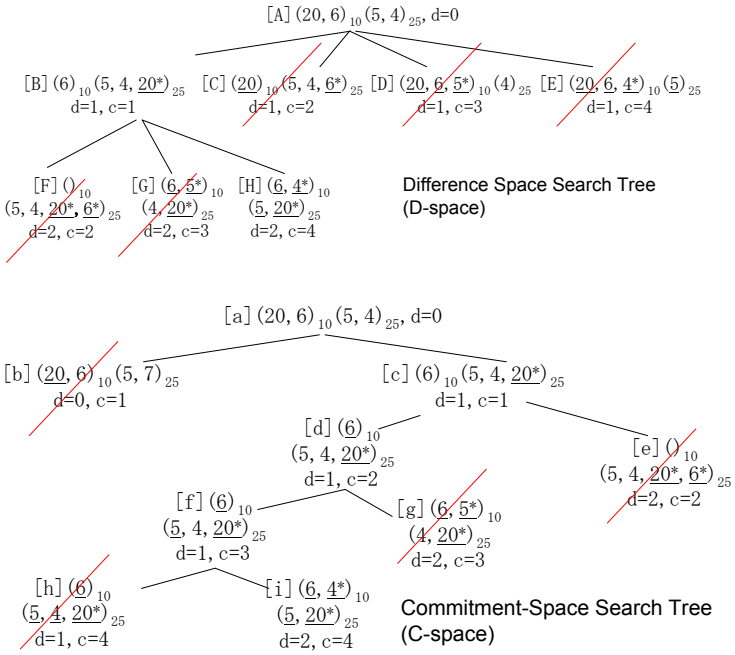

Fig. 1. Difference-Space and Commitment-Space search trees for VMRP with 2 servers ($c_1 = 10, c_2 = 25$), and 4 VMs ($w_1 = 20, w_2 = 6, w_3 = 5, w_r = 4$). The initial assignment is $I = \{1, 1, 2, 2\}$. For each node, $d = \#$ of perturbations from the initial assignment I, and $c = \#$ of commitments. Underlined values are committed, and an asterisk (*) indicates that the committed value is different from I. For example, node H in D-space has 4 committed variables, where 2 have values different from the initial assignment. Infeasible nodes are pruned (slash through node).

edges in C-space have cost 0 (e.g,. the edges $a \rightarrow b, c \rightarrow d, d \rightarrow f, f \rightarrow h$ in Figure 1). As with D-space, C-space can be searched using either the DFBNB or IDA*.

D-space can be seen as the subset of C-space where all committed decision variables are assigned a value different from the initial assignment. For example, given variables x_1, x_2 and initial assignment $I = \{1, 2\}$, the assignment $\{\underline{3}, 2\}$ is in D-space because v_1 is committed to a value that is different than in I, but $\{\underline{1}, 2\}$ is not in D-space because x_1 is committed to the same value as in I. Each node in D-space corresponds to a unique assignment of variables to values. In contrast, C-space has multiple nodes representing the same assignment of values to variables, except that the nodes have different commitments. For example, the search state $\{1, 2\}$ and $\{\underline{1}, 2\}$ represent the same variable assignments, but in the latter, a commitment has been made to assign x_1 to 1 for all of its descendants, whereas the former has not made any commitments.

In a problem with n variables with a domain of m possible values, the search tree for D-space contains $T_D = m^n$ nodes, because there is a 1-to-1 correspondence between the unique assignments of variables to values and the nodes in

the D-space search tree (after elimination of symmetric nodes), and there are m^n unique assignments. This tree does not have a regular branching factor; the first level below the root node consists of the $n(m-1)$ assignments which differ from the initial assignment by exactly 1.

The root node of the C-space search tree is the initial assignment, and at each tree depth, we commit a variable to one of m values (one of the m values is the initial value in I). Thus, the C-space search tree has size $T_C = 1+m+m^2+...m^n$, and by straightforward manipulations, $T_C = (m^{n+1}-1)/(m-1)$. Comparing T_D and T_C, T_C/T_D approaches 2 in the worst case (when $m = 2$).

Despite the redundancy in C-space compared to D-space, C-space has a lower branching factor (m) compared to D-space (depends on node; $n(m-1)$ at first level of the search tree). Given a comparable number of nodes, and the same set of pruning techniques, a narrower, deeper tree is easier to search than a wider tree, because a successful pruning in the narrower tree tends to prune more nodes than a successful pruning at the same depth in the wider tree. Thus, the main, potential advantage of C-space is in reorganizing the search tree structure from the relatively "top-heavy" D-space tree to a relatively narrow tree with branching factor m, at the cost of some redundancy.

3 Experimental Comparison of MPRP Algorithms for the VMRP

We evaluated search algorithms for the VMRP based on the search strategies described above, enhanced with the following VMRP-specific bounds.

A server is *oversubscribed* if the sum of the VM weights assigned to it exceeds its capacity. A lower bound LB_O (for pruning in DFBNB and for the admissible heuristic h in IDA*) is computed as follows: For each oversubscribed server S, sort the uncommitted VMs assigned to S in non-decreasing order, and count the number of VMs that must be removed from S in this order such that usage no longer exceeds capacity. For example, for the VMRP state $\{(5,6)(4,3)(\underline{10},1,2)\}$ where server capacity is 10, $LB_O = 3$. This is because either the VM with weight 5 or 6 must move from the first server, and the 1 and 2 must move from the third server (the VM with weight 10 is excluded from consideration because it is uncommitted). In addition, there are some bounds based on the wasted space in the servers, which we detailed in a workshop paper [7]. However, since those other VMRP-specific techniques affect all search strategies equally, and have less than a factor of 2 impact on runtime, we omit the details here due to space.

A common scenario for server consolidation in practice involves consolidating tens of services into fewer servers, where the target ratio of VMs to physical servers is commonly around 3-5 [3]. We generated solvable, random benchmarks based on this scenario as follows. For each server s_j, $1 \leq j \leq m$ (all servers with a capacity of 1000), VMs were uniformly generated in the range $[200, 400]$ and assigned to s_j until the remaining capacity was under 100. At that point, one 'filler' VM was generated, whose weight was constrained such that the slack (remaining capacity) in s_j was between 0 and 20. Minimizing the slack in this way increases

	Commitment Space (C-space)									D-space					
	DFBNB			IDA*			IDA*/B			IDA*			IDA*/NG		
m	**fail**	time	nodes	**fail**	time	nodes	**fail**	time	nodes	**fail**	time	nodes	**fail**	time	nodes
5	0	0.1	34481	0	0.05	10379	0	0.23	81431	0	0.17	42783	0	0.18	25770
6	0	1.9	5.0e5	0	0.4	7.2e4	0	3.3	1.1e6	0	3.0	7.6e5	0	4.54	5.6e5
8	4	119.9	2.0e7	0	32.4	5.3e6	**8**	157.0	5.4e7	**8**	104.7	2.9e7	**6**	52.6	6.6e6
10	**28**	112.5	1.5e7	**14**	150.6	2.1e7	**29**	251.3	1.1e8	**29**	156.3	5.2e7	**23**	145.8	13.5e7
12	**30**	-	-	**26**	267	2.8e7	**30**	-	-	**30**	-	-	**30**	-	-

Table 1. Virtual Machine Reassignment Problem results with varying # of servers (m) The *fail* column indicates # of instances (out of 30) not solved within the time limit (600 seconds/instance). The *time* and *nodes* columns show average runtime and nodes generated on the successful runs, excluding failed runs.

the instance difficulty. Then all the VMs were removed from the servers, shuffled and reassigned to the servers in a round-robin manner, resulting in a solvable VMRP instance where each server has a balanced number of VMs, but some servers are overloaded. We tested each of the search algorithm configurations on 30 instances with m varying from 5 to 12, with a time limit of 600 seconds per instance on a 3.0GHz Intel Core2 processor. We used a most-constrained variable ordering and a lexicographic value ordering for all algorithms (results using different variable and value ordering strategies were similar). In addition to combinations of C-space and D-space with DFBNB and IDA*, we also ran two additional configurations: (1) The C-space+IDA*/B configuration is similar to C-space+IDA*, except that all pruning is disabled for nodes which have the same variable *assignment* as their parents (i.e., the "extra" nodes resulting from cost 0 edges in C-space which are not present in D-space, such as nodes b, d, f, h in Figure 1). At all other nodes, pruning is enabled as in C-space+IDA*. This tests how pruning at these "extra" nodes impacts C-Space+IDA*. (2) The D-space+IDA*/NG configuration is similar to D-space+IDA*, except that this algorithm is based on the iterative deepening algorithm in [13], which, on the d-th iteration, enumerates variable assignments with at most d differences compared to the initial assignment. However, it does not assert commitments, and nogoods are used for symmetry pruning. While this searches the same space of variable assignments as D-space, the lack of commitments means that variable ordering and bound-based pruning techniques are less effective.

Table 1 shows the results. The *fail* column indicates the number of instances (out of 30) that were not solved within the time limit. The *time* and *nodes* columns show average time spent and nodes generated on the successful runs, excluding the failed runs.

As shown in Table 1, C-space+IDA* significantly outperformed the other algorithms. D-space+DFBNB is not shown due to space, but performed significantly worse than D-space+IDA* and C-space+DFBNB. C-space+IDA*/B performs significantly worse than C-space+IDA*, and is comparable with D-space+IDA*, indicating that in fact, pruning at the "extra", inner nodes in C-space plays a significant role in enhancing the performance relative to D-space.

The poor performance of D-space+IDA*/NG, which does not assert any commitments, shows the importance of exploiting commitments for pruning and variable ordering. Similar results were obtained with VM sizes in the range $[100, 300]$

4 Related Work

Several techniques for solving CSPs and combinatorial optimization problems which apply some backtracking strategy to a nonempty variable assignment (similar to our algorithms) have been previously proposed, including [5, 14]. Although these techniques use a partial or complete variable assignment to guide the search for a solution, they do not have an explicit goal or mechanism for ensuring a *minimal* count of perturbations from a particular initial assignment.

Previous work has explicitly addressed minimal perturbation. Ran, et al. proposed iterative-deepening in D-space for binary CSPs [13], with symmetry pruning based on nogoods. El Sakkout and Wallace [6] considered a minimal cost repair problem for scheduling. They consider difference functions that can be expressed linearly - the MPRP objective of minimizing difference count is excluded ([6],p.368). Their probe backtracking algorithm does not explicitly consider the initial schedule, and reschedules from scratch. Barták et al. investigated overconstrained CSPs for which there is likely to be no feasible solution without violated constraints [4], 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 [12].

IDA* in C-space is related to Limited Discrepancy search (LDS) [9] and its variants, as both algorithms search a space which is characterized by some notion of "discrepancy". LDS can be viewed as a best-first search, where the cost of a node is the number of discrepancies (from the first value returned by a value ordering heuristic) [10]. Let δ be the class of all value ordering heuristics where the first value returned by the ordering is the value in the initial state. Using some value ordering from δ, we can implement LDS in C-space which is similar to IDA* in C-space. The differences are: (1) On the d-th iteration, LDS explores nodes with up to d discrepancies, while IDA* searches nodes with cost estimate (Section 2) $f \leq d$. The reason for this difference is that the goal of LDS to find a solution for a standard CSP – it is not explicitly trying to find a min-perturbation solution. The lack of a nontrivial bound/heuristic means that LDS performs even less pruning than C-space+IDA*/B (which at least prunes at all non-redundant nodes) (2) LDS specifies a specific *value ordering strategy*, i.e., a policy from the class δ, whereas IDA* (as well as DFBNB) does not specify a particular value ordering among sibling nodes (our VMRP experiments used lexical ordering). Thus, LDS, when applied directly to the MPRP, is a special case of C-space IDA* with a trivial lower bound ($h = 0$) and a value ordering heuristic from δ. On the VMRP, we found that all the C-space algorithms in Table 1 significantly outperforms LDS (more than an order of magnitude).

5 Conclusions

We investigated exact search algorithms for repairing constraint violations with a minimal number of perturbations from an initial state. Starting with a straightforward difference-based search space which enumerates differences from the initial assignment, we showed that introducing an explicit notion of commitments to values allows elimination of symmetries (subsuming nogoods), as well as domain-specific pruning. We then propose a commitment space, which reorganizes the search tree into a regular, narrow structure at the cost of some redundancy. Experimental results for the Virtual Machine Reassignment Problem, a min-perturbation variant of bin packing, show that search in C-space is significantly more efficient than search in D-space, and that the combination of C-space with IDA* result in the best performance for the VMRP. Although we used the VMRP as an example, C-space and D-space are general notions for MPRP problems, and future work will investigate additional MPRP domains. Some preliminary results for an antenna-scheduling related domain are in [7].

References

1. G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. In *Proc. 15th ACM Symp. on parallel algorithms and architectures*, pages 258–265, 2003.
2. Y. Ajiro. Recombining virtual machines to autonomically adapt to load changes. In *Proc. 22nd Annual Conf. of the Japanese Society for Artificial Intelligence*, 2008.
3. Y. Ajiro. NEC System Platform Research Labs. Personal Communication, 2009.
4. R. Barták, T. Müller, and H. Rudová. A new approach to modeling and solving minimal perturbation problems. In *Recent Advanced in Constraints*, volume 3010 of *LNAI*, pages 233–249. Springer-Verlag, 2004.
5. C. Beck. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, 2007.
6. H. El-Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5:359–388, 2000.
7. A. Fukunaga. Search algorithms for minimal cost repair problems. In *Proc. CP/ICAPS 2008 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 2008.
8. R. Gupta, S.K. Bose, et al. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. In *IEEE Int. Conf. on Services Computing*, 2008.
9. W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proc. IJCAI*, pages 607–615, 1995.
10. R. Korf. Improved limited discrepancy search. In *Proc. AAAI*, pp. 286–291, 1996.
11. R.E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
12. T. Müller, H. Rudová, and R. Barták. Minimal perturbation in course timetabling. In *PATAT Selected papers, LNCS vol.3616*, pages 126–146. Springer-Verlag, 2005.
13. Y.P. Ran, N. Roos, and H.J. van den Herik. Approaches to find a near-minimal change solution for dynamic CSPs. In *Proc. CP-AI-OR*, pages 378–387, 2002.
14. G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. AAAI*, pages 307–312, Seattle, Washington, 1994.
15. W. Vogels. Beyond server consolidation. *ACM Queue*, 6(1), 2008.