

# Search Algorithms for Minimal Cost Repair Problems

Alex S. Fukunaga

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

## Abstract

Many scheduling scenarios involve altering or repairing an initial, candidate schedule, sometimes as a result of unexpected changes to the problem. Given a constraint satisfaction problem and an initial assignment of values to variables that violates some constraints, we consider the problem of finding a solution which differs minimally from the initial state. We consider two search spaces for optimally solving this problem, a commitment-based search space and a difference-based search space. Two search strategies are evaluated – a depth-first branch-and-bound strategy as well as an iterative-deepening strategy. We present preliminary empirical investigations of these search spaces and strategies are on a simple resource-scheduling repair problem, as well as a resource allocation repair problem related to server consolidation. An iterative-deepening strategy applied to a commitment-based search space is shown to have the best performance on these two problems.

## Introduction

Most research on scheduling and resource allocation problems consider the problem of generating solutions from scratch – given a set of objects to schedule or assign to onto resources and a set of constraints on the assignment (e.g., capacity and/or temporal constraints), the problem is to generate an assignment that satisfies the constraints. It is usually assumed that there is no initial assignment or schedule. However, in practice, there are situations where it is necessary to find solutions to scheduling and resource allocation problems that are as close as possible to a given, initial state.

One example involves the rebalancing of loads among servers. *Server consolidation* is the process of using virtual machine technology to consolidate multiple servers onto a set of fewer servers. (Vogels 2008). There is currently great demand in the IT industry for server consolidation products and solutions due to the significant improvement in energy efficiency and reduction in “server sprawl” (reduced hardware and datacenter space requirements) which can be achieved by server consolidation. Consider a set of  $m$  servers, each with capacity  $c_1, \dots, c_m$ , and a set of  $n$  jobs, where each job has a weight  $w_1, \dots, w_n$  representing the amount of workload that the job demands of a server. Each “job” represents

a virtual machine, and each “server” represents physical server machine. The server consolidation problem, which is to determine whether all jobs can be assigned to exactly one server such that the sum of all jobs assigned to each bin  $j$  is less than or equal to  $c_j$  is an instance of the decision version of the classical NP-complete *bin packing problem*. (Garey & Johnson 1979). Exceeding the capacity would cause the overloaded physical server to fail or have unacceptable performance.

Now, suppose that due to changes in demand, the current job weights differ from loads that were projected when the initial server assignments were made, resulting in some servers being overloaded. It is possible to reassign jobs (virtual machines) among servers in order to rebalance the loads. However, migrating a virtual machine between servers incurs costs (direct costs include system administration costs, intangible costs are the costs of possible downtime for a service). The *Min-Cost Load Rebalancing Problem* (LRP) is the problem of finding a new assignment of jobs to servers such that (1) no server is overloaded, and (2) the number of jobs that are moved between servers is minimized.

The need to find a new solution which minimally perturbs a previous solution frequently arises in scheduling problems. One possible scenario is similar to the load rebalancing scenario above, in that requirements/constraints suddenly change, invalidating the current solution. For example, consider airport gate scheduling. Flight delays can result in the current gate-flight assignment becoming invalid, requiring a new gate schedule. However, reassigning gates incurs significant costs such as moving baggage handling crews and gate attendants, notifying passengers, etc.

Another scenario where solutions similar to a given initial state is desired is when where the initial solution is (at least partly) the result of human decision-making and negotiation. For example, in staff scheduling problems employees express preferences regarding when they want to work, but their preferences must be balanced against the staffing demands of the business. This requires generating a schedule that satisfies staffing requirements while deviating minimally from employee preferences.

We investigate algorithms for finding optimal solutions to this *minimal-cost repair problem*. We define

two simple model problems for minimal cost repair problems. We investigate two candidate search spaces, commitment-space and difference-space and two search strategies, depth-first-branch-and-bound and IDA\* for exploring each search space, significant as well as domain-specific pruning mechanisms. We show that a new combination of commitment-space and IDA\* results in the best performance in our two domains.

## The Minimal Cost Repair Problem

The class of problems we considering is the *Min-Cost Repair Problem* (MCRP), a type of constraint satisfaction problem where the goal is to *repair* an initial assignment of values to variables (i.e., find a conflict-free solution), with minimal *cost*, where we define cost simply as the number of differences between a candidate solution and the initial assignment.

While a standard CSP requires that we find a solution which does not violate any constraints, the MCRP imposes the additional goal of minimizing the distance from an initial assignment of values to variables. The MCRP is a special case of constraint optimization, where the objective function seeks to minimize the number of differences relative to an initial assignment.

The term “minimal perturbation” has been used to denote several related but different, problems in the literature (see Related Work section). In this paper, we focus on the particular version of the minimal perturbation problem posed in (Ran, Roos, & van den Herik 2002). We introduce the term “MCRP” to avoid confusion with the previous, different problems studied in the literature. MCRP problems are also similar to optimal AI planning/search domains, in that we search for the shortest transformation from an initial state to a final state which satisfies some criterion. However, our MCRP domains are more limited in that there are no preconditions for state changes, i.e., any value can be assigned to any variable at any point in the search, except that final goal state needs to be feasible with respect to problem-specific constraints.

## The Min-Cost Load Rebalancing Problem (LRP)

We have already informally described the Min-Cost Load Rebalancing Problem. More formally, the Min-Cost Load Rebalancing Problem (LRP) is defined as follows: Given  $m$  servers with capacities  $c_1, \dots, c_m$ , a set of  $n$  jobs with weights  $w_1, \dots, w_n$ , and an initial assignment  $I_1, \dots, I_n$  of jobs to servers, find a new assignment of jobs to servers  $x_1, \dots, x_n$ ,  $1 \leq x_i \leq m$ , such that each job is assigned to exactly one server, and for every server  $j$ , the sum of the job weights assigned to it is less than or equal to its capacity  $c_j$ . The objective is to minimize  $\sum_{i=1}^n (x_i \neq I_i)$ , the number of differences between the initial and final assignments.

The LRP is interesting because bin packing constraints are at the core of many resource allocation and scheduling problems, so the problem of repairing bin packing

constraints serves as a useful benchmark for our domain-independent repair strategies. We have already described an application in virtual machine migration. A closely related problem was considered in (Aggarwal, Motwani, & Zhu 2003), which presented approximation algorithm for minimizing the maximum load on any server while moving less than  $k$  jobs. Similar applications also exist in the problem of process migration in distributed systems.

## The Min-Cost Schedule Repair Problem (SRP)

Consider a resource with limited capacity  $R$  (or equivalently,  $R$  identical resources with unit capacity),  $T$  discrete contiguous timer intervals, and a set of jobs to be executed on the resource. Each job requires 1 unit of resource capacity to execute, has a duration  $d(j)$ , and a window of time during which must be executed, defined by the earliest start time  $e(j)$ , and the latest start time  $l(j)$ . The Resource Scheduling Problem with Time Windows (RSP-TW) is to assign a start time for each job such that no time interval overlaps more than  $R$  jobs. The RSP-TW models problems of allocating a set of identical resources (or a single resource with limited, discrete, capacity) among jobs/requests that have a temporal window during which they must be executed or served. The RSP-TW is NP-complete, since a special case where all tasks have the same deadline  $l(e) = T$  and same earliest start times ( $s(t) = 0$  for all  $j$ ) is the NP-complete *multiprocessor scheduling problem* (Garey & Johnson 1979).

The RSP-TW is a highly simplified model that captures some of the essential resource and temporal constraints present in some real-world scheduling problems such as the NASA Deep Space Network antenna array scheduling problem (Clement & Johnston 2005), and the Air Force Satellite Communication Network scheduling problem (Barbulescu, Whitley, & Howe 2004).

We define the *Min-Cost Schedule Repair Problem* (SRP) as an extension of the RSP-TW, such that given an initial assignment of jobs to start times, and the objective is to find a conflict-free schedule where the number of jobs that have different start times than the initial schedule is minimized.

## MCRP Search Spaces and Strategies

In general, a Min-Cost Repair Problem that is based on an NP-complete problem is itself NP-complete. For example, the LRP is NP-complete by a straightforward reduction from the decision version of bin packing, and the SRP is NP-complete by reduction from the multiprocessor scheduling problem.

One possible approach to solving a MCRP is to modify an existing algorithm for the underlying CSP or scheduling problem, such that instead of terminating after finding a feasible solution, the solver searches for the solution with minimal distance from the given initial state. For example, for the LRP, we might try to take an existing

bin packing algorithm and modify it so that instead of terminating after finding a feasible bin assignment of  $k$  bins, it searches for a  $k$ -bin solution with minimal distance from the initial assignment. Unfortunately, this is difficult because the optimal solution to the MCRP can be pruned by the pruning techniques used by solvers for the underlying CSP. For example, all of the currently available bin packing algorithms, including column generation-based integer linear programming solvers as well as search-based branch-and-bound algorithms, all derive their effectiveness from mechanisms such as column generation, lower bounds, and dominance detection, which, if applied straightforwardly, can prune the optimal MCRP solution. In other words, it is nontrivial to start with an existing solver and derive an algorithm for finding optimal solutions the MCRP without disabling or substantially altering the pruning mechanisms that are responsible for making the original solver effective in the first place. Thus, we investigate exact (complete) search algorithms specifically designed to solve the MCRP.

Given an initial state  $I = x_{1,0}, \dots, x_{n,0}$ , let  $D_i$  be the set of states which differ from  $I$  by exactly  $i$  variable assignment i.e.,  $i$  variables in  $X \in D_i$  have a value which differs from that of the initial state  $I$ . We call the set  $D = D_1 \cup D_2 \cup \dots \cup D_n$  the *difference space*, or *D-space*. The root node of this search space is the initial state  $I$ .

We can perform a standard depth-first branch-and-bound (DFBNN) search in D-space, where at each node, we select a variable  $x$  and assign it some value which differs from the value in the initial state  $I$ . The lower bounds and infeasibility checks described below are applied at each node. Figure 1 shows the key features of depth-first backtracking in D-space. The *symmetrySet*, which is initialized to the empty set at the top level, ensures that symmetric paths are not considered. For example, consider a problem with two variables  $v_1$  and  $v_2$ . Without the symmetry check mechanism implemented in lines 1, 2, and 4 in Figure 1, the algorithm would enumerate both the node ( $v_1 = 2, v_2 = 2$ ) as well as the node ( $v_2 = 2, v_1 = 1$ ), which correspond to the same assignment of values to variables.

Another way to view the MCRP is as a path-finding problem, where the start state is the initial assignment, and the objective is to find a goal state with minimal distance from the start state. A natural candidate is a best-first strategy such as A\*, but because A\* requires exponential memory, we instead consider IDA\* (Korf 1985), which expands nodes in a best-first order using linear space (at the cost of reopening some nodes). The admissible heuristic function used by IDA\* is the same as the lower bounding function used for DFBNN (see below), and the  $d$ -th iteration of IDA\* explores the subset of the DFBNN 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.

Search in D-space has been the basis of previous work on finding minimal perturbation solutions for CSPs. Ran

et al (2002) applied IDA\* in D-space to solve a minimal perturbation problem for binary CSPs.

An alternative search space is a *commitment-based search space* (C-space), 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. 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  $x_1, \dots, x_n$ , we denote a search state as the list  $S = \{x_1 = val_1, \dots, x_n = val_n\}$ , or more concisely,  $\{val_1, \dots, val_n\}$ . Furthermore, the values are annotated with an underline “  ” if the variable is committed to that value. For example, in a 2-variable MCRP where the current assignments are  $v_1 = 1, v_2 = 2$ , and we have committed  $v_1 = 1$ , we can denote this state as  $\{v_1 = \underline{1}, v_2 = 2\}$ , 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. As with D-space, it is possible to apply either DFBNN or IDA\* search strategy in C-space. DFBNN in C-space was previously proposed by Minton et al (1992).

IDA\* in C-space is related to Limited Discrepancy search (LDS) (Harvey & Ginsberg 1995), as both algorithms search a space which is limited by some notion of “discrepancy”. In fact, it has been noted that LDS can be viewed as a best-first search, where the cost of a node is the number of discrepancies in its path from the root (Korf 1996). In LDS, a “discrepancy” refers to a decision which deviates from the first value suggested by a value-ordering heuristic. Let  $\delta$  be the class of all value ordering heuristics where the first value suggested for variable  $x_i$  by the value ordering is the value of  $x_i$  in the initial state. Using some value ordering from the class  $\delta$ , 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. On the other hand, on the  $d$ -th iteration, IDA\* explores all nodes where at each node, the sum  $f = g + h \leq d$ , where  $g$  is the number of discrepancies so far, and  $h$  is a lower bound on the additional number of discrepancies required to reach a conflict-free state. (2) IDA\*, like DFBNN, is a strategy which specifies the overall backtracking strategy. This is orthogonal to the selection of a *value ordering strategy*, which specifies the order in which children of a node are sorted. LDS and its variants prescribe both a backtracking strategy as well as a particular value ordering strategy. Thus, IDA\* in C-space is a generalization of LDS for the LRP, and LDS for the LRP is a special case of the C-space IDA\* with a trivial lower bound ( $h = 0$ ) and some value ordering heuristic from class  $\delta$ . We found that using the lower bounds on discrepancies described below significantly improves performance compared to an LDS without a discrepancy lower bound, so our experimental results only consider IDA\* in C-space with a nontrivial lower bound, and we do not further consider LDS.

We are currently evaluating various variable ordering and value ordering heuristics in both C-space and D-space. In our experiments, we used a standard most-constrained variable ordering and a simple, random value ordering for all combinations of C-space, D-space, DFBNB, and IDA\*. Results using different variable and value ordering strategies are qualitatively similar.

Note that D-space can be seen as the subset of C-space where all committed variables are assigned a value different from the initial assignment. For example, given variables  $v_1, v_2$  and initial assignment  $v_1 = 1, v_2 = 2$ , the assignment  $\underline{v_1 = 3}, v_2 = 2$  is in D-space because  $v_1$  is committed to a value that is different value than in the initial assignment and  $v_2$  is uncommitted, but  $\underline{v_1 = 1}, v_2 = 2$  is not in D-space because  $v_1$  is committed to the same value as in the initial assignment. Each node in D-space corresponds to a unique assignment of variables to values.

Despite the redundancy in C-space compared to D-space, C-space has a much lower branching factor compared to D-space, which can make pruning techniques more effective and result in better performance.

### Algorithm for the Min-Cost Load Rebalancing Problem (LRP)

Our search algorithm for the LRP is the basic search algorithms described above, enhanced with the following new, domain-specific pruning techniques. A bin is *oversubscribed* if the sum of the weights of the items assigned to the bin exceeds its capacity.

A search node  $N$  is *infeasible* if there exists no descendant of  $N$  that is a conflict-free state. Infeasible nodes can be pruned. The *wasted space* of a bin  $B$  is amount of space in the bin that can not be occupied by any uncommitted item currently not assigned to  $B$  without making  $B$  oversubscribed. For example, suppose we have a bin  $B = (\underline{7})$  with capacity 10, and three remaining uncommitted items 7, 4, and 2 (which are currently in other bins). The wasted space of  $B$  is 1, because the minimal amount of unused space that can be in  $B$  is 1, after moving and committing the 2 to  $B$ . The wasted space of a bin assignment is the sum of the wasted space of each of the individual bins.

Let  $W_{UB} = \sum_{j=1}^m c_j - \sum_{i=1}^n w_i$ , the difference between total bin capacity and total item weights, be an upper bound on the total amount of wasted space allowed. A node in the search tree is infeasible if a lower bound on the wasted space exceeds  $W_{UB}$ . Such a lower bound is obtained by summing the lower bound on the wasted space of each single bin,  $\sum_{j=1}^m W_{LB}(j)$ . For each bin  $j$ ,  $W_{LB}(j)$  is computed by find the packing of bin  $j$  with minimal wasted space, using all uncommitted items (this is a relaxation because we allow uncommitted items to be used by more than one bin). This is a subset sum problem, which our current implementation solve using a straightforward, branch-and-bound algorithm. The remaining space in the bin after packing the optimal subset-sum packing is a lower bound on the actual wasted space of the bin.

We use the following lower bound (admissible heuristic) for DFBNB and IDA\*. A bin is *oversubscribed* if the sum of the weights of the items assigned to the bin exceeds its capacity. An oversubscription-based lower bound  $LB_O$  is computed as follows: For each oversubscribed bin  $B$ , sort the uncommitted items assigned to the  $B$  in non-decreasing order of weight, and count the number of items that must be removed from  $B$  in this order until the bin occupancy no longer exceeds capacity. For example, given the bin assignment  $\{(5, 6)(4, 3)(\underline{10}, 1, 2)\}$  where bin capacity is 10,  $LB_O = 3$ . This is because either the 5 or 6 must move from the first bin, and the 1 and 2 must move from the third bin (although the 10 is the largest number in the third bin, it is committed so it is not considered for movement by the  $LB_O$  computation). Another lower bound,  $LB_U$ , is based on the bins that are undersubscribed. If any bin has more free space than  $W_{UB}$ , then some of the remaining uncommitted items must move into the bin to reduce the wasted space. A valid lower bound is computed by adding (and counting) the largest remaining uncommitted items until the free space no longer exceeds  $W_{UB}$ . Since both  $LB_O$  and  $LB_U$  are inexpensive to compute, we use a combined lower bound,  $LB_{OU} = \max(LB_O, LB_U)$ . We can not add  $LB_O$  and  $LB_U$  because that may result in the double-counting of the potential required moves.

### LRP Experimental Results

We generated a set of solvable benchmarks as follows. For each bin  $b_j$ ,  $1 \leq j \leq m$  (all bins with a capacity of 100), items were randomly generated in the range [10,30] and assigned to  $b_j$  until the remaining space was under 10. At that point, one 'filler' item was generated and added to the bin such that the remaining space in  $b_j$  was between 0 and 2. Then all the items were removed from the bins, shuffled and reassigned to the bins in a round-robin manner. This results in an instance with some overloaded bins, but it is guaranteed that the instance has a feasible solution. The number of items in each instance varies, but is approximately  $5m$ . We tested each of the four search algorithm configurations on 20 random instances with  $m$  varying from 4 to 15, with a time limit of 300 seconds per instance on a 2.4GHz Intel Core2 processor. Results are shown in Table 1. The *fail* column indicates the number of instances (out of 20) 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, IDA\* in C-space significantly outperformed the other three algorithms. Both of the C-space algorithms significantly outperformed the D-space search algorithms, and in both search spaces, IDA\* outperformed DFBNB. Future work will present more detailed results, including the effect of various variable and value orderings, as well as additional bounds.

```

Dspace_search(committed, uncommitted, depth, changes, symmetrySet)
... (termination checks, domain-specific infeasibility and lower-bound pruning applied)
1 newSymmetrySet = copy(symmetrySet) //initialize symmetry set for next level in tree
2 foreach v ∈ order_vars(uncommitted\symmetrySet)
3   oldValue = value(v)
4   newSymmetrySet = newSymmetrySet∪v //added v to symmetrySet for descendants and siblings
5   foreach val ∈ sort_values(get_feasible_values(v)\oldVal)
6     SetValue(v,val)
7   result = Dspace_search(committed∪v, uncommitted\v, depth+1, changes+1, newSymmetrySet)

Cspace_search(committed, uncommitted, depth, changes)
... (termination checks, domain-specific infeasibility and lower-bound pruning applied)
1 v = choose_variable(uncommitted)
2 oldValue = value(v)
3 foreach val ∈ sort_values(get_feasible_values(v))
4   set_value(v,val)
5   if val = oldVal then newChanges = changes else newChanges = changes+1
6   result = Cspace_search(committed∪v, uncommitted\v, depth+1, newChanges)

```

Figure 1: Outline of Commitment-space (C-space) search and Difference-space (D-space) search for min-perturbation repair. Most-constrained variable ordering and min-conflict value ordering are used. The sets of uncommitted and committed variables are represented by the variables `uncommitted` and `committed`. Both C-space and D-space searches can use either a depth-first branch-and-bound or iterative deepening A\* strategy.

m (# bins)	Commitment Space (C-space)						Difference Space (D-space)					
	DFBNB			IDA*			DFBNB			IDA*		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
4	0	0.007	1230	0	0.002	149	0	28.516	3455195	0	0.009	839
5	0	0.083	14365	0	0.007	1082	0	32.832	3608220	0	0.112	9103.3
6	0	1.091	188503	0	0.044	5072	<b>7</b>	29.868	37833329	0	0.897	63796
8	0	51.011	8580060	0	1.132	110905	<b>16</b>	141.160	8569661	<b>2</b>	23.196	1326308
10	<b>20</b>	n/a	n/a	0	8.010	690901	<b>20</b>	n/a	n/a	<b>11</b>	51.997	2448719
12	<b>20</b>	n/a	n/a	0	58.731	4911799	<b>20</b>	n/a	n/a	<b>14</b>	95.74	4446458
15	<b>20</b>	n/a	n/a	<b>7</b>	140.660	10848252	<b>20</b>	n/a	n/a	<b>20</b>	n/a	n/a

Table 1: Min-Cost Load Rebalancing Problem: DFBNB and IDA\* in C-space and D-space. The *fail* column indicates # of instances (out of 20) that were not solved within the time limit (300 seconds/instance). The *time* and *nodes* columns show average time spent (seconds on 2.4GHz Intel Core2) and nodes generated on the successful runs, excluding the failed runs.

## Algorithm for the Min-Cost Schedule Repair Problem (SRP)

Our search algorithm for the SRP is based on the basic search algorithms in C-space and D-space described above, where the following new, domain-specific lower bound is applied. At each node, we compute a lower bound for the SRP using the following relaxation: Consider the problem of determining the minimal set of requests to *remove* from the schedule such that removing these requests results in a conflict-free schedule (i.e., a schedule where no time interval is oversubscribed). The size of this minimal set is clearly a lower bound on the SRP. This relaxation is solved using a straightforward branch-and-bound algorithm.

During DFBNB and IDA\* search, each time a variable is assigned a value, forward checking is applied, and the algorithm backtracks if an uncommitted variable ends up with an empty domain as a result of this variable assignment. As with the LRP, we have experimented with a number of other enhancements to detect infeasibility,

dominance, and symmetry for this problem, but these were not used in the experiments nor described because the cumulative improvement has not been significant so far. For both the LRP and SRP, the pruning techniques described were verified to be significantly better than more obvious alternatives

## SRP Experimental Results

To generate difficult but solvable instances, we generate an initial schedule that looks like a packed rectangle, and then randomly perturb it. This way, the instance has little slack, is guaranteed to be solvable, and is likely to be difficult. Given a resource capacity  $R$ , a number of time intervals  $T$ , minimum request duration  $d_{min}$ , and maximum request duration  $d_{max}$ , number of start times  $s$ , and number of perturbations  $p$ , we generated an instance as follows: (1) Initialize an integer  $i = 0$ . (2) Generate a request with duration  $d$  randomly chosen from  $[d_{min}, d_{max}]$ , where the earliest start time is set to  $i$ , and the latest start time is  $i + s$ . (3) increment  $i = i + d$ . (4) If  $i < T$ , go to (1), else quit. After repeating (1)-(4)  $R$

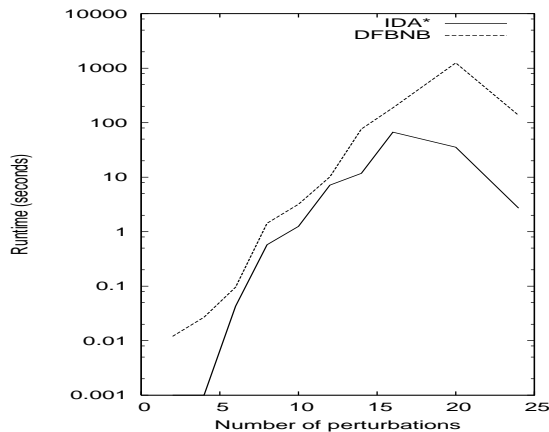


Figure 2: Min-Cost Schedule Repair Problem: Effect of varying number of perturbations in initial assignment

times, if we place all the requests at their earliest possible start time, we will have a conflict-free schedule that looks mostly like a fully packed rectangle of height  $R$  except at the right edge, which may be jagged. If necessary,  $T$  is increased to accommodate this packed schedule. This initial schedule is perturbed by changing the start time of  $p$  randomly selected requests to a random start time from its domain. This results in a broken schedule with oversubscribed intervals.

We compared C-space and D-space search using DFBNB and IDA\*, using the problem generator described above. First, we measured performance as the problem size was scaled, holding other parameters constant. The resource capacity  $R$  was varied between 2 and 8, and we set  $I=24$ ,  $d_{min} = 4$ ,  $d_{max} = 6$ ,  $s=4$ , and  $p=15$ . This generates instances with between 10 (when  $R = 2$ ) to 40 (when  $R = 8$ ), requests each with 4 possible start times. The results (20 instances per configuration, 600 second time limit per instance) are shown in Table 2.

It is possible that the relative performance depends on the degree of perturbations applied to the initial perfect packing by the problem generator (fewer perturbations mean that solutions tend to be shallower on the search tree, giving IDA\* an advantage). To observe the effect of  $p$  on the relative performance of DFBNB and IDA\* search strategies in C-space, we generated instances with  $R = 5$ ,  $I = 24$ ,  $d_{min} = 4$ ,  $d_{max} = 6$ ,  $s=4$ , and varied  $p$  between 2 and 24 (20 instances per parameter set). Note that the number of requests generated given these parameters is approximately 20-25, so  $p = 24$  results in all requests being assigned a random start time from their domains. The results in Figure 2 show that IDA\* outperformed DFBNB for all values of  $p$ , but aside from the trivial problems ( $p = 2$ ), there is no clear relationship between  $p$  and IDA\* performance relative to DFBNB.

## Related Work

The C-space and D-space search spaces described in this paper were first considered by researchers in contexts

that did not require minimal perturbation. Minton et al. (1992) describe a backtracking repair algorithm for constraint satisfaction problems, which they call “informed-backtracking”. This is depth-first backtracking in C-space, applied to a heuristically generated initial assignment of values to variables. Their goal is to solve a new constraint satisfaction problem from scratch, so there is no mechanism for finding a *minimal* set of perturbations from the initial assignment. Backtracking in D-space was previously proposed for dynamic constraint satisfaction by Verfaillie and Schiex (1994), and Ran, et al. (2002) proposed IDA\* in D-space for binary CSPs.

El Sakkout and Wallace (2000) considered a minimal cost repair problem for an abstract scheduling problem. A key difference from the MCRP is that they consider difference functions that can be expressed linearly (the MCRP difference count is not expressible in their model). Also, their probe backtracking algorithm does not explicitly consider the initial schedule, and reschedules from scratch. Barták, Müller, and Rudová have investigated a Minimal Perturbation Problem (MPP) for overconstrained constraint satisfaction problems for which there is likely to be no feasible solution without violated constraints (Barták, Müller, & Rudová 2004). The objective is to find a maximal assignment of consistent variables which also differs minimally from an initial state. This differs from the MCRP, which seeks a complete, conflict-free variable assignment with minimal distance from an initial assignment (in other words, their MPP is fundamentally a constraint optimization problem, while the MCRP is fundamentally a constraint satisfaction problem). Müller, Rudová, and Barták also investigated an iterative repair algorithm which is biased to find minimal perturbation solutions for course timetabling (Müller, Rudová, & Barták 2005). Because this is a local search algorithm, the minimality of the perturbation from the initial schedule is not guaranteed.

Thus, while previous work considered both C-space and D-space searches, as well as iterative-deepening and standard depth-first branch-and-bound search strategies, the combination of iterative deepening and C-space has not been previously considered in the literature, as far as we are aware. See the section on MCRP search algorithms for a discussion on the relationship between limited discrepancy search (Harvey & Ginsberg 1995) and C-space IDA\* search.

## Conclusions

We investigated optimal search algorithms for the min-cost repair problem, where the goal is to find solutions to constraint satisfaction problems that differ as little as possible from a given initial state. Our contributions are as follows: We proposed two basic problems as models for the min-cost repair problem: the min-cost load rebalancing problem, a model of resource capacity allocation repair, and min-cost schedule repair problem, a basic model of resource allocation repair with time windows. We investigated two search spaces for the MCRP, commitment space and difference space, and applied depth-

R (capacity)	Commitment Space (C-space)						Difference Space (D-space)					
	DFBNB			IDA*			DFBNB			IDA*		
	fail	time	nodes	fail	time	nodes	fail	time	nodes	fail	time	nodes
2	0	0.002	90	0	0.003	84	0	0.650	18451	0	0.576	16355
3	0	0.018	370	0	0.008	166	4	49.339	803478	0	5.677	124090
4	0	0.528	9676	0	0.404	7677	8	26.116	580920	1	85.889	1331199
6	2	23.524	101057	0	14.706	46969	20	n/a	n/a	9	104.788	1010714
8	5	39.880	102149	0	75.586	314058	20	n/a	n/a	14	266.545	2800607

Table 2: Min-Cost Schedule Repair Problem: The *fail* column indicates the number of instances (out of 20) that were not solved within the time limit (600 seconds/instance). The *time* and *nodes* columns show average time spent (seconds on 2.4GHz Intel Core2) and nodes generated on the successful runs, excluding the failed runs.

first branch-and-bound and IDA\* search in both search spaces. In addition, we developed domain-specific pruning techniques for the LRP and SRP. Our preliminary experimental results show that C-space search is significantly more efficient than D-space search, and that the new combination of C-space with IDA\* result in the best performance on the LRP and SRP. While we focused on algorithms for finding optimal solutions, non-optimal algorithms (e.g., weighted IDA\*, local search) are an area for future work.

While most scheduling research focuses on generating solutions from scratch, we believe that the MCRP is an important model in many practical scenarios, and there is much work to be done in this area. Future work includes extending this work to more complex problems (e.g., min-cost repair in scheduling problems with ordering constraints), as well as investigating additional metrics for differences relative to the initial assignment, for example, adding weights to requests to model the fact some parts of the schedule are more “expensive” to modify than others.

## References

- Aggarwal, G.; Motwani, R.; and Zhu, A. 2003. The load rebalancing problem. In *Proc. 15th ACM Symposium on parallel algorithms and architectures*, 258–265.
- Barbulescu, L.; Whitley, D.; and Howe, A. 2004. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *Proceedings of AAAI*.
- Barták, R.; Müller, T.; and Rudová, H. 2004. A new approach to modeling and solving minimal perturbation problems. In *Recent Advances in Constraints*, volume 3010 of *LNAI*, 233–249. Springer-Verlag.
- Clement, B., and Johnston, M. 2005. The Deep Space Network scheduling problem. In *Proc. IAAI*.
- El-Sakkout, H., and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* 5:359–388.
- Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.
- Harvey, W., and Ginsberg, M. 1995. Limited discrepancy search. In *Proc. IJCAI*, 607–615.
- Korf, R. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Korf, R. 1996. Improved limited discrepancy search. In *Proc. AAAI*, 286–291.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58(1-3):161–205.

Müller, T.; Rudová, H.; and Barták, R. 2005. Minimal perturbation in course timetabling. In *PATAT Selected papers, LNCS vol.3616*, 126–146. Springer-Verlag.

Ran, Y.; Roos, N.; and van den Herik, H. 2002. Approaches to find a near-minimal change solution for dynamic CSPs. In *Proc. CP-AI-OR*, 378–387.

Verfaillie, G., and Schiex, T. 1994. Solution reuse in dynamic constraint satisfaction problems. In *Proc. AAAI*, 307–312.

Vogels, W. 2008. Beyond server consolidation. *ACM Queue* 6(1).