Cost-based Large Neighborhood Search*

Tom Carchrae[1] and J. Christopher Beck[2]

1 - Cork Constraint Computation Centre, University College Cork - IRELAND
carchrae@4c.ucc.ie
2 - Department of Mechanical and Industrial Engineering, University of Toronto - CANADA
jcb@mie.utoronto.ca

Abstract: Large Neighborhood Search (LNS) is a method for combining Constraint Programming (CP) and local search. We present a new neighborhood which focus search effort on the part of the problem which has the greatest effect on the objective. We explore this idea through minimizing makespan in the job-shop scheduling problem. In this domain, we determine the impact of each activity on the objective through slack. Slack is a measure of how critical each activity is in the best known solution. By choosing to spend our search effort on the most critical variables we reduce the typical size of the neighborhood space which results in improved performance. We test our approach on three sizes of job-shop scheduling problems, the largest being 80 jobs on 80 machines, or 6400 activities. While there is not much of an improvement on small problems, we see the importance of a good neighborhood as the problem size increases.

Keywords: Metaheuristics - Constraint Programming - EU/ME - CP/OR

1 Introduction

One of the central drawbacks of constructive search methods like constraint programming (CP) is the limitation of the size of the problem that can be solved in a reasonable time. While constructive methods give very strong performance on small problem instances, this performance does not tend to scale well on large problems, especially in the case of optimization problems. Local search techniques such as tabu search tend to perform much better on larger problems.

Large Neighborhood Search (LNS) is a hybrid method which combines the power of constructive search with the scaling performance of local search. As in local search, we modify an existing solution to the problem. However, instead of making small changes to a solution (as is typical with local search move operators) we select a subset of the problem to search for improved solutions. For example, a typical local search move in scheduling would be to swap the ordering of two activities, eg $A_i < A_j \rightarrow A_j < A_i$. Instead, in LNS we specify to search for improved solutions by choosing to reassign some activities, eg search $\{A_i, A_j\}$. However since CP-search can efficiently deal with more than two variables many variables are typically selected for the improvement search. As in local search, a neighborhood is defined by ranking all possible moves, in the case of LNS each move is a set of variables to search.

However, the choice of neighborhood is crucial to the performance of LNS. We wish to select variables which are likely to give an improvement to the cost of the solution. We are also concerned with the number of variables in each search, fewer variables give a quicker search, but we must also choose related variables so that an improvement can be found. Promising neighborhoods have been identified for vehicle routing, network design and car sequencing[4].

In this paper we present a new neighborhood based on the cost impact of variables involved in the objective function. We explore this idea using job-shop scheduling with the objective of minimizing makespan. In this case, the impact on cost of an activity can be determined by how critical the start time variable is. We measure criticality using slack time, the difference between the earliest start time and the latest start time of an activity. Our neighborhood consists of

*This material is based upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075, the Embark Initiative of the Irish Research Council of Science Engineering and Technology under Grant PD2002/21, and ILOG, S.A.

Algorithm 1: Large Neighborhood Search

```
1 Create initial solution
2 while Not Optimal and Time Left do
3 |   Choose part of problem
4 |   Freeze the rest of the variables
5 |   if Search finds improvement then
6 |   |   Update solution
7 |   end
8 end
```

increasing supersets of variables, where $CBN(S_i)$ is the set of activities which have a slack value less than or equal to S_i, where S is an ascending list of unique slack values which appear in the current solution.

2 Large Neighborhood Search

The central idea of Large Neighborhood Search (LNS) is very simple and presented in Algorithm 1. Starting with a solution to the entire problem, we select a part of the problem and re-optimize that part. By focusing efforts on a part of the problem we can intensify search to find better solutions but this is balanced by choosing different parts of the problem to diversify search to avoid getting stuck in a local optima. The key benefit of LNS is that we can use constraint programming in the intensification step giving more search power than simple move operators.

We note there are several parameters to choose when implementing LNS. We must decide on a method to freeze the variables in step 4. In our case, we use a scheduling precedence graph where the variables represent the next activity and freezing is achieved by assigning the next activity for all consecutive pairs of frozen activities. Next, some implementations of LNS use a time or failure limit to limit search in step 5 however we chose not to apply any limit. Finally, we choose to completely search for all improvements in step 5 rather than stopping after we find the first improvement.

3 Cost-based Neighborhood

We now introduce the idea of a cost-based neighborhood. For the task of minimizing makespan in the job-shop scheduling problem it is well known that to improve a solution it is necessary to reorder some activities along a critical path. A critical path is defined as a sequence of critical activities (which have a slack value=0) that are connected by precedence constraints, either induced by an ordering on a resource or by the problem definition.

$$CBN(S_i) = \{activity_j | j \in Activities, slack_j < S_i\} \tag{1}$$

We choose our neighborhood to exploit the fact that re-optimizing critical activities is likely to improve solution quality. First, we create an ascending list S of each unique slack value found in the current solution. Then, we progressively search subproblems where we re-optimize all activities whose slack is less than or equal to S_i. In the first subproblem, we attempt to exhaustively search all permutations of critical activities. We progress to larger subproblems including the less critical activities if no improvement is found. We restart the procedure when an improved solution is found. Because we are choosing subproblems based on the slack value we hope that we are choosing parts of the problem in increasing order of importance to the objective function.

Problem	Pure	Random	CBN	60/40 Mix
20x20	0.02463	**0.00482**	0.01612	0.00610
40x40	0.06699	0.02084	0.01710	**0.00139**
80x80	0.01757	0.01592	**0.00061**	0.00665

Table 1: Mean MRE on each problem set, running time of 600 seconds.

We note this approach of re-optimizing the critical path is very similar to the method iFlat presented in [1]. Their approach differs in that they always replace the current solution after a move, even if no improvement has been found. They also use a heuristic to search rather than a complete search as is implemented in this work.

There are some interesting aspects of this approach which can yield further exploitation. When we fail to find a solution we incrementally add more activities to the next subproblem, but we have already performed a search on the previous subproblem. Let F_i be the set of activities that were fixed in the subproblem produced by slack S_i, that is all activities which are not in $CBN(S_i)$. If we have completely searched the subproblem S_i, then we know that no improved solution exists to the larger subproblem S_j where $i < j$ if the current assignment of the activities in F_i stays the same. We can then place a no-good on F_i and avoid search. Such an approach is used in [3] with their systematic local search. Indeed, taking an approach where all subproblems are completely solved yields a complete method.

However, the harvesting of no-good information needs to be traded off against computational effort. As the subproblems grow, it may be prohibitively expensive to perform a complete search on the subproblem. In this case, a tradeoff needs to be considered to increase performance. We have not yet implemented a no-good learning scheme and so far have only investigated the tightening of the upper bound on makespan.

Another interesting development is the idea of Propagation Guided LNS presented in [4]. Propagation Guided LNS is used to create neighborhoods of related variables. It would be interesting to this combined with cost-based selection.

4 Experiments

We explored the effects of using cost-based neighborhoods on three sizes of job-shop scheduling problems. The small set consists of 20 jobs and 20 machines, a medium set has 40 jobs and 40 machines, and a large set has 80 jobs on 80 machines. There are 10 instances in each set and the total number of activities per instance is 400, 1600 and 6400 respectively. We are interested to see how the techniques scale as the problems grow in size.

We compare the performance of cost-based methods against a random approach. We use the same number of activities that would have been chosen by $CBN(S_i)$ however we select the activities randomly. This allows us to fix the size of the neighborhood and compare purely based on the activities selected. We also show the results of combining random selection with the cost-based approach, with 60% cost-based choices and 40% random choices. During the search for improved solutions, we use a constructive search technique that uses texture-based heuristics [2] and strong constraint propagation [6, 5]. *Pure* represents the result of solving the problem using constructive search without LNS.

We ran each configuration for 10 minutes on a Pentium 4, 1.8Ghz computer running Fedora Core 2 with the 2.6 linux kernel.

5 Results

Table 1 displays mean relative error (MRE), a measure of the mean extent to which an algorithm finds solutions worse than the best known solutions. MRE is defined as follows:

$$MRE(a, K) = \frac{1}{|K|} \sum_{k \in K} \frac{c(a, k) - c^*(k)}{c^*(k)} \tag{2}$$

where K is a set of problem instances, $c(a, k, r)$ is the lowest cost solution found by algorithm a on problem instance k during run r, and $c^*(k)$ is the lowest cost solution found in these experiments for problem instance k.

As we can see from the results, the difference in performance varies greatly on each problem set. Perhaps the most striking result is how randomization wins on small problems, but as the problems get larger the careful selection of activities becomes more important. Although it is not shown here, the medium sized problems show an interesting crossover, before 200 seconds the cost-based method performs best but the performance plateaus and then the combination of the cost-based method with randomization performs better.

We also performed an analysis of the size of neighborhoods which lead to improved solutions. This is not shown here, however the cost-based method and the combination method were able to find improved solutions on much smaller sized subproblems compared to random selection.

6 Conclusions

We have presented a new neighborhood which is based on variables which contribute to the cost of the solution. The effectiveness of the approach was investigated on three sizes of problems. The results show that as problem size increases, the importance of choosing variables to improve is more important.

[1] Cesta A., Oddi A., and Smith S.F. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 742–747. AAAI Press / The MIT Press, 2000.

[2] J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

[3] B. Dilkina, Duan L., and Havens W.S. Extending systematic local search for job shop scheduling problems. In *Proceedings of Eleventh International Conference on Principles and Practice of Constraint Programming*, 2005.

[4] Perron L., Shaw P., and Furnon V. Propagation guided large neighborhood search. In *Proceedings of Tenth International Conference on Principles and Practice of Constraint Programming*, pages 468–481, 2004.

[5] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143:151–188, January 2003.

[6] W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.