

Greedy Job Scheduling

Forward and Backward State Searches and Heuristics

Michelle E. Ruse

Iowa State University
Department of Computer Science
Ames, Iowa 50011
mruse@iastate.edu

Abstract

The world is full of problems, some seem easy for the human mind to grasp and solve, but when asking a machine to solve the problem, many of the intricacies become apparent. Such is the case with Job Scheduling, a Constraint Satisfaction Problem where the goal is to get all the necessary jobs completed most efficiently, meaning the shortest amount of time. This paper explores forward and backward state searches, using a breadth-first algorithm to create the plans, stopping when a schedule is first found, thus the shortest possible schedule. The chosen constraints are very restricting and greedy; no machines can sit idle if a job was on the queue waiting to be done, once started a sub-job must be completed and sub-jobs of a given jobs have to be completed in order. Several experiments were run, varying the number of jobs, the number of sub-jobs of jobs, the number of machines and randomizing the number of sub-jobs.

Problem Specification and Code

The problem is a Constraint Satisfaction Problem where jobs consisting of sub-jobs are assigned to specified machines to be completed. The initial state is that no jobs have been assigned to machines, nor completed and all have been requested, waiting to be assigned to machines. The goal state is all jobs are completed. The constraints are greedy and restrictive. No machines can sit idle if a job was on the queue waiting to be done, once started a sub-job must be completed, each sub-job has a specified machine to which it must be assigned and sub-jobs of a given jobs have to be completed in order. The functions allowing the assignment of the next possible states are the Forward and Backward-State Search Algorithm, both used in an exhaustive breadth first search and using a heuristic based on the shortest next possible sub-job.

The problem involves creating a specified number of jobs, each containing a specified or random number of sub-jobs, depending on the experiment. Each sub-job has a required amount of time it takes to be completed, between 10 and 100 minutes. A breath first creation of the schedules for each minute was created. Each plan has a

plan node for each possible minute, thus a plan node tree is created in a breadth-first fashion.

A java program was written and modified for the several experiments. The classes included the scheduler, planNode, subjob, randomjob, and list. The scheduler class creates several random jobs, each of which creates a subjob. The scheduler stores the subjobs and their states in the planNode, creating a breath-first tree. At each level, all new possible planNodes are created based on each parent node plan. Thus, every possible path of the search tree is a schedule for the jobs.

The selection for the possibility of next possible planNodes is decided upon by the states of the sub-jobs at that time interval. Sub-jobs are either waiting on the machine queue, they are on the machine or they are done, representing requested, begun and done states, respectively, which are common industry assignments. There is no need to track a job being in the done state, since the completion of the final sub-job is synonymous with completion of the entire job.

Since the algorithm is exhaustive up until a shortest time solution is found, a large constraint is the memory of the machine on which the experiments are performed. This limited the experiments to number of jobs, sub-jobs and machines.

Forward-State Search

Greedy Forward-State Search

Forward-State Search starts with the first tasks that need to be completed. In this problem solving algorithm, the first sub-jobs of all jobs are considered as the first level of schedules at time one. The proceeding sub-jobs can be assigned to the machines and completed only when their preceding sub-jobs have been completed (sub-job n cannot be completed unless sub-job n-1 has been completed). Also, sub-jobs are never removed from a machine, once they are assigned to the machine until they have been completed. So at each time step new planNodes are created with all possible states.

Greedy Forward-State Search with minimum time Heuristic

longest schedules for the lower number of sub-jobs and GBH shows a few longer schedule with more sub-jobs, but not significantly longer than the GB and GF schedules, as GFH had. Although GF and GB find the shortest schedules, there is a much larger cost. The schedule times are in the hundreds while the costs for GF and GB reach the tens of thousands in the higher number of sub-jobs. The cost does not appear to be worth the minimal decrease in schedule time over the GBH created schedules.

Four Jobs. With four jobs, only two and three sub-jobs experiments could be performed due to the memory limitation and the exhaustive search algorithms. Before running into large memory errors, both the GF and GB algorithms consistently give the same schedule times, with BGH giving some higher times, followed by GFH. The cost for the consistently smaller times range from the thousands into the hundred thousands. One example in the tens of thousands cost range:

	Forward	w/Heuristic	Backward	w/Heuristic
Cost:	11181	336	25540	336
TIME:	336	336	336	336

This is example text. It is 10 point Times Roman. This is example text. It is 10 point Times Roman. This is example text. It is 10 point Times Roman. This is example text. It is 10 point Times Roman. This is example text. It is 10 point Times Roman. This is example text. It is 10 point Times Roman.

Five Jobs. With five jobs, only two sub-jobs were tested. The schedule times were consistent with what has been seen in the previous job experiments, more closer to those of the four job experiment, with schedule times generally equal. The costs here start in the tens of thousands and some memory errors are encountered even with only two sub-jobs.

Three Machines Experiments

In the three machine experiments, twenty runs of each experiment were gathered. An experiment set a number of jobs, between two and five, to create and a number of sub-jobs, between two and six, to create. Also, for each job a random number of sub-jobs was tested. The random jobs reflected similar results to the fixed number of sub-job tests.

Two Jobs. The schedule times were all very similar for the algorithms except the GBH with longer times. There was only one instance in the twenty runs where GFH had a longer time. Surprisingly, the cost was very close or equal to the path size for the GF and GB, with the GB algorithm faring worse than the GF. With an increase in schedule

time, there was a significantly large increase in cost for the GF and GB algorithms.

Three Jobs. Schedule times for GFH and GBH were longer than those for GF and GB algorithms, with GFH having longer times with lower number of sub-jobs and GBH having longer times with more sub-jobs. The schedule times run in the hundreds, around three- to five-hundred. These are equivalent to the cost of the GFH and GBH algorithms. The costs are generally significantly larger for GF and GB, starting in the thousands and a few instances of jobs in the tens of thousands. If there was a large difference in the cost of GF and GB, GB was generally had the highest cost.

Four Jobs. The schedule times for GFH and GBH were longer than those for the GF and GB. There was not a clear winner for longest schedule times, as either would have a longer time, depending on the sub-jobs. The cost for GF and GB were rarely in the hundreds, mostly in the thousands and tens of thousands. The costs varied significantly, but neither algorithm between GF and GB was consistently the higher costing algorithm.

Five Jobs. Only the possibility of two sub-jobs was explored. Often the schedule times were equal. Some longer schedule times were seen with GFH and GBH. The costs for GF and GB were exorbitant, in the thousands, tens of thousand and even in the hundred thousands. As will all previous experiments, the even distribution of sub-jobs across jobs and machines leads to shorter schedule times and lower costs.

Summary and Conclusions

Costs. The higher costing jobs in all experiments for GF and GB algorithms had uneven machine usage distribution of the sub-jobs. As seen in the previous example, one job requires usage of machine for all its sub-jobs and another job has all but one sub-job on the same machine. This leads to higher costs for GF and GB. This situation also appears to be the culprit for poor schedule times for GFH and GBH.

Efficiency. Each time step creates a new node, for each possible state. Sometimes this is just a copied node. This could be handled more efficiently.

Future Directions.

Efficiency. Each time step creates a new node, for each possible state. Sometimes this is just a copied node. This could be handled more efficiently by not creating a new node if it just cloned for the next time step. A more complicated node and handling of these would be necessary.

Greedy Backward-State Search with Maximum Time Heuristic. Also, for this Greedy Backward-State Search, a

maximum time requirement heuristic was created as an alternative to the minimum time Heuristic. The thought was that then it would match the plan of the Greedy Forward-State Search with minimum time heuristic in that all minimum time times will be done first, since the maximum time is selected for the sub-jobs from the last to the first. This is a future test and comparisons.

Total Schedule. It would be easy to also look at the number of schedules begin considered at the time the algorithms stop and return a schedule. A more efficient program could fully perform the exhaustive searches and give the total number of possible schedules.

Constraints. A less constraining approach where jobs could be assigned to any machine might yield useful results. Also, allowing machines to be idle at any time step would make the search more useful and could find possible shorter schedules. This would require a more efficient implementation, as this was tried for this instance, but memory error did allow any useful results to be produced.

References

Ghallab, M. (2002) Plan-Space, *Planning-Graph & CSP-based Techniques for Planning*. ESSLII 2002.

Ghallab, M. (2002) *Handling Time for Planning*. ESSLII 2002.

LaValle, S.(2003) *Planning Algorithms*. University of Illinois.

Lopez, A. & Bacchus, F.(2003) *Generalizing GraphPlan by Formulating Planning as a CSP*. University of Toronto: Canada.

Russell, S. & Norvig, P. (2003)*Artificial Intelligence: A Modern Approach*.Englewood Cliffs, NJ: Prentice-Hall.

Smith, B.(1995) *A Tutorial on Constraint Programming*. School of Computer Studies Research Report Series 95,14: University of Leeds.