Discovering Heuristic Strategy for Solving Scheduling Problems

Rasaiah Loganantharaj
Center for Advanced Computer Studies
USL, Lafayette, LA 70504
rasaiah@cacs.usl.edu

Abstract

Optimally solving a scheduling problem is computationally intractable, hence varieties of techniques and methods have been explored to obtain sub-optimal solutions to such problems. In this paper we introduce a technique that effectively combines appropriate heuristics so as to find a quality solution when used with a hill climbing algorithm. We will demonstrate the proposed approach by applying it to solve some single machine scheduling problems. We compare the quality of the solutions with the ones obtained by using randomized techniques.

1. Introduction

Scheduling of tasks involves allocating time slot to each task while satisfying their resource requirements and temporal constraints. such as, predecessor successor relationships. Such assignments lead to feasible schedules. An optimal schedule is a feasible schedule that satisfies the given objective function. Finding an optimal solution to many scheduling problems is computationally intractable, hence sub optimal solutions are being sought. There are two major approaches to finding sub-optimal solutions to scheduling problem: applying heuristics, or applying randomized techniques, which include simulated annealing and genetic algorithm. In this paper we focus on discovering *heuristic strategy*, which is a weighted average of some heuristic parameters, to solve certain class of scheduling problem.

A solution to a scheduling problem can be modeled as a heuristically guided search of a path from the initial node to a goal node. The path from the initial to the goal node represents a feasible solution. From each node of the search space, there can be multiple branches in the exploration and the heuristic values help to select the most promising branch that leads to the next node. There are two search methods: local and global. While a global search method, such as A* is guaranteed to obtain the optimal solution when it is combined with permissible heuristics, the search space for such method grows exponentially. On the other hand, local search methods, such as hill climbing takes linear space, but do not guarantee the optimal solution. The success of using hill climbing method depends of the quality of heuristics applied to guide the search. In this paper we learn such useful heuristics and compare the results with the one obtained by applying randomized technique especially genetic algorithm.

This paper is organized as following. We describe the problem in section 2, and it is followed by an introduction of genetic algorithm. In section 4 we describe our approach to learning heuristic strategy. In section 5, we apply the learning method to obtain sun-optimal solution to some NP-hard single machine scheduling problem. The paper is concluded with a summary and a discussion.

2. Description of problem

To study our technique, we consider conceptually simple problem of scheduling a single machine. In a typical single machine scheduling problem, jobs will be arriving at different time, each job will be having different duration, and there will be a non zero preparation time to make the appropriate changes in the machine configuration to make it ready for the incoming job. Let us consider a simplified version of the problem: All the jobs arrive at the same time, and there is no preparation time.

We will use the following notation for the rest of this paper.

Description	Notation
duration of job k	Dur(k)
completion time of job k	Comp(k)
scheduled time /start time of job k	Sch(k)
due date of job k	Due(k)

Suppose there are N jobs and the objective function is to minimize the average completion time.

$$\text{average completion time} = \Sigma_j (\text{Sch}(j) + \text{Dur}(j)) / N$$

Optimally achieving this objective function is polynomial; when jobs are scheduled in the ascending order of their duration. We will show that our method will discover the strategy of favoring the shortest job first.

Not all the single machine scheduling problems can be solved polynomially. Consider an objective function of minimizing the total tardiness. A job is said to be *tardy* if it is completed after its due date. The total tardiness is given by Σ_j max(0, (Sch (j) + Dur(j) - Due(j))) where max(a,b) returns the maximum of *a* and *b*. If a job completes before the due date we assume that there is no reward and the penalty is zero. It has been shown [Du90] that even this simplified scheduling problem is NP-hard.

We will be using genetic algorithm to learn the heuristic strategy as well as to solve the same scheduling problem for comparison of the results. We, therefore, introduce genetic algorithm in the next section.

3. Introduction to Genetic Algorithm

A genetic algorithm [Davi87, Mich96] starts with an initial population consisting of a set of chromosomes. Each chromosome in the population corresponds to a feasible solution of a problem that we are trying to solve. Each chromosome consists of a sequence of genes. Typically a binary string represents a gene, though other representations are also possible. New solutions are created from the population by selecting a pair of chromosome and mating them or mutating a chromosome.

As the population grows with new offspring, the stronger ones are kept while removing the weaker chromosomes. A single or a multiple crossover operation performs a mating of a pair of chromosomes. Linear interpolation between a pair of chromosome is also used for producing offspring. The crossover points are randomly selected. Let us illustrate it with an example. Consider the pair of chromosomes G_{11}, G_{12}, G_{13}, G_{14}, G_{15}, G_{16} and G_{21}, G_{22}, G_{23}, G_{24}, G_{25}, G_{26}. Suppose a single crossover takes place after the second gene. The offspring are G_{11}, G_{12}, G_{23}, G_{24}, G_{25}, G_{26} and G_{21}, G_{22}, G_{13}, G_{14}, G_{15}, G_{16}. This is illustrated in Figure1.

A double crossover operation after genes 2 and 4 on these two chromosomes is illustrated in Figure 2.

A chromosome of *k* gene is considered to be a point in a *k*-dimensional space. Mating of a pair of chromosome can be considered as linear interpolation of these two points corresponding to these chromosomes. The gene *j* of the offspring generated by linear interpolation of chromosome 1 and 2 (of the above example) with distance *d* from the first chromosome is given by $G_j = G_{1j} + d*(G_{2j} - G_{1j})$.

4. Our approach to discovering heuristic strategy

Let us illustrate our approach with an example. Consider a single machine scheduling problem with the objective of minimizing the total completion time. That is, to minimize

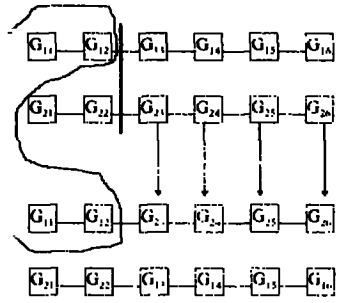

Figure 1

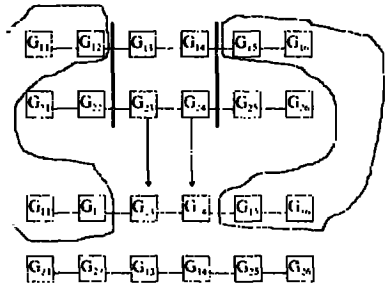

Figure 2

Σ_j (Sch (j) + Dur(j)) / N. The objective function is dependent on the duration and the scheduled time, which in turn dependent on the duration of the previously scheduled jobs. Therefore, duration is the only independent variable. The heuristic can be (1) favoring the shortest job, (2) favoring the longest job, or (3) favoring some combination of both. Suppose the heuristic parameter h_1 and h_2 respectively refers to the shortest first and the longest first. The weighted summation of the heuristics is $w_1*h_1 + w_2*h_2$. We apply hill-climbing method with the weighted summation of the heuristic. At each step of the search, job with the highest combined heuristic value is scheduled and the process continues until the entire jobs are scheduled.

How do we discover the weights of the relevant heuristic that find the best solution to a problem? To discover the appropriate weights we apply genetic algorithm.

4.1 Chromosome

If we have k heuristics, we will have to discover k weights (w_m for heuristic h_m). We generate chromosome of length k of which each gene corresponds to each weight. Each gene's value varies from 0 through 1. We randomly generate a pool of chromosomes each of length k.

4.2 Fitness value

In our model, each gene of a chromosome corresponds to the weight of the corresponding heuristic parameter. For each chromosome, we apply the weights and obtain the corresponding schedule. The objective function is evaluated for each schedule and it becomes the fitness value of the chromosome. The best chromosome is the one that associates with the lowest fitness value.

4.3 Operations

Single crossover operation or linear interpolation (we provide equal chances) creates the offspring. Further, mutating randomly selected gene modifies a chromosome. The percentage of mutation is controlled by a parameter (in our experiment we use 10%).

4.4 Fine Tuning

To allow the possibility of one heuristic vale to dominate the others, we generate the first k chromosome each has zero weights except for one position, say m, that correspond to heuristic m. For example, the chromosome 000100 corresponds to the one with m=4 and k=6.

4.5 Discovering Strategy

After generating the initial chromosome and evaluating their fitness value, the convergence phase is started. After we generated offspring of four times the mating pool size, the mating pool is updated with the best of the total population. The generation is continued until there is no improvement in the best chromosome. When the algorithm converged, the best chromosome has the best weights to combine the heuristics. This is how a heuristic strategy is discovered.

5. Applications

Let us apply this technique to discover the strategy for single machine scheduling problem with the objective function of minimizing the average completion time. As we have discussed before, there are two possible heuristic parameters: shortest job and longest job. The normalized values of the shortest and the longest jobs are obtained as in table 1, where MaxD and MinD respectively reefers to maximum and the minimum duration of the jobs of the training set. We randomly generated a training set of arbitrary duration. A portion of the training set is given in the following table.

Jobs	Duration	Shortest normalized	Longest Normalized
1	20	0.5	0.5
2	15	0.75	0.25
3	25	0.25	0.75

In the complete table, the maximum and the minimum duration of the jobs are 30 and 10 units respectively. We have applied the techniques to learn the strategy to solve the problem. The algorithm converged with the best chromosome of 1 0 indicating that favoring the shortest job is the best strategy in minimizing the total summation of the completion time. This experiment is repeated for jobs ranging from 10 to 100. We consistently obtained the same result confirming that the shortest job first achieves the minimal average waiting time as we have expected.

Let us apply the same technique to learn the strategy for minimizing the tardiness. Since there is no polynomial solution to solve this problem optimally, we have to compare the results with randomized technique to quantify the quality of the result. We have generated a set of jobs and their duration and due dates randomly. The objective function is to minimize the total tardiness, which is given by Σ_j max(0, (Sch (j) + Dur(j) - Due(j)). A careful examination revels that the objective function is dependent on the duration of the jobs and their due dates. The heuristics are, therefore, longest and the shortest jobs and the due dates. The data we used for training is given in Table 1. Shortest and the longest jobs are normalized as before. Similarly, the shortest and the longest due dates are defined as

shortest(k) = 1 - (Dur(k)-MinD)/(MaxD -MinD), and

longest(k) = 1 - (MaxD-Dur(k))/(MaxD -MinD).

Where, MinDu and MaxDu are the minimum and the maximum due dates.

Let h_1, h_2, h_3 and h_4 respectively represent the normalized attributes shorter duration, longer duration, shorter due dates and longer due dates.

After convergence, we got the weight vector for the training set as [0.93, 0.13, 0.93, 0.08]. We ran the learning algorithm for randomly generated data set for jobs ranging from 10 through 90. The weight vector changes with the data set, but it lies closer to the vector [1, 0,1,0].

Jobs	Duration	Due date
1	10	33
2	12	41
3	13	59
4	15	103
5	11	95
6	12	32
7	12	15
8	11	19
9	12	33
10	18	40

Table 1

We use the weight vector [1,0.1,0] to schedule jobs to minimize the total tardiness. That is, the priority for each job is computed as the summation of the following parameters: normalized shorter job and shorter duration. The schedule is obtained by arranging the jobs in the descending order of their priority. The total tardiness of each schedule is computed. The qualities of the schedules (total tardiness) are compared with the ones obtained by applying randomized technique. To get the best result from the randomized scheduling algorithm, we applied random key encoding [Norm94] and feature-based encoding [Leon95]. We considered the following features of the job: duration of a job, and the interval corresponding to the difference between the due date and the duration. The results are shown in Table 2.

Jobs	heuristic	Randomized technique		
		random	duration	due
10	200	205	206	222
10	116	96	99	127
10	172	95	93	141
50	14,038	14,201	13,271	16,188
50	12,306	11,992	12,098	15,635
50	15,521	15,100	14,715	17,822
90	35,702	36,818	36,111	49,322
90	26,700	30,688	31,178	42,148
90	20,564	30,564	30,610	44,097

Table 2

Each problem is generated randomly using the parameters minimum and maximum duration of jobs, and the maximum due dates. Feasible due dates are those that are grater than their the duration of jobs. The cost in the table 2 represents the total tardiness for each run using the given encoding technique. There is no relation between different rows and the result must be compared along each row.

6. Summary and Discussion

In this paper we have proposed a technique to discover heuristic strategy that effectively solve problem. We have demonstrated our strategy to a certain class of scheduling problems. We applied to a single machine scheduling problem with two different objective functions. In the first instance, the objective function is to minimize the average completion time of jobs. This problem has a polynomial solution; scheduling the task in the ascending order of the duration will achieve the optimal solution. Our training program consistently generated the weight vector [1,0,0,0] for all the training data sets. The weight vector indicates the highest priority to the shortest job, which is exactly the same result as the one obtained by solving the problem optimally.

When the objective function is to minimize the total tardiness, optimally solving the problem is no longer computationally tractable. We have applied our technique to train and discover a strategy to solve the problem sub-optimally. We found that the weight vector varies with training date set, which did not surprise us since the optimal solution of the problem is computationally intractable. Even though the weight vector is dependent on the training set, it varies around the vector [1,0,1,0]. We use this vector to schedule tasks heuristically. Using the specification of a problem with the job duration and due dates, a schedule is created by arranging the jobs in the descending order of their priorities. The priority of each job is the summation of the normalized shorter duration and the shorter due date. Sorting the jobs according to their priority computationally dominates other aspects of normalizing the duration and the due dates, therefore, the computational time complexity of scheduling jobs is same as that of a quick sort, which is O(N log(N)). Where N is the number of jobs. From the results of Table 2 it is quite clear that the quality of the schedule using heuristic strategy is better or as good as the one obtained from using randomized technique. The heuristic strategy performs better in larger problem than in smaller problem.

The technique we described in this paper is quite general and it can be applicable to any problem that can be modeled as a search problem. As we have shown, for a

computationally intractable problem, the trained weight vector varies with training instances. It is reasonable to get a median value of the weight vector and apply it to solve other problems. We have applied similar technique to improve the performance of a constraint-based scheduler [Loga97]. This method of learning and applying it to find a solution can be combined with any time good algorithm [Dean88, Gras96]. The initial solution can be the one using the heuristic strategy and the subsequent solutions can be obtained by refining the initial solution using the methods and techniques amenable to any time good algorithm.

Acknowledgment: The author would like to thank Thomas Bushrod of USL and Kelvin Manning of NASA, KSC for helpful hints and suggestions during the early phase of this research.

References

[Davi87] L. Davis, *Genetic Algorithms and Simulated Annealing, Research Notes in Artificial Intelligence*, Morgan Kaufmann, 1987.

[Dean88] T. Dean and M. Boddy, An Analysis of Time Dependent Planning, in *Proceedings of AAAI-88*, 1998.

[Du90] J. Du and J. Y. Leung, Minimizing Total Tardiness on One Machine is NP-Hard, in *Mathematics of Operations Research*, Vol. 15, pages 483-495.

[Gras96] J. Grass and S. Zilberstein, Anytime Algorithm Development Tool, in *Sigart Bulletin*, Vol. 7, No. 2, 1996.

[Leon95] V. J. Leon and R. Balakrishnan, Strength and Adaptability of Problem-Space based Neighborhoods fore Resource-constrained Scheduling, in *OR Spektrum 17*, 1995, Springer Verlag, pages 173-182

[Loga97] R. Loganantharaj and T. Bushrod, Improving the Efficiency of the Ground Processing Scheduling System, in *NASA/ASEE Summer faculty Research Report*, 1997.

[Norm94] B. A. Norman and J. C. Bean, Random Keys Genetic Algorithm for Job Shop Scheduling, in *Technical Report 94-5*, Dept. of Industrial and Operations Engineering, The University of Michigan, 1994

[Mich96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, Third, Revised and Extended Edition, Springer 1996.