Genetic Algorithms for Open Shop Scheduling and Re-Scheduling

Sushil J. Louis **Zhijie Xu**

Department of Computer Science
University of Nevada
Reno - 89557
email: sushil@cs.unr.edu

Abstract

We combine genetic algorithms and case-based reasoning principles to find optimally directed solutions to open shop scheduling and open shop re-scheduling problems. Appropriate solutions to open shop scheduling problems are injected into the genetic algorithm's population to speed up and augment genetic search on a related open shop re-scheduling problem. Preliminary results indicate that the combined genetic algorithm – case-based reasoning system quickly finds better solutions than the genetic algorithm alone.
Keywords: Genetic Algorithms, Case-Based Reasoning, Open Shop Scheduling.

1 Introduction

Genetic algorithms (GAs) are stochastic, parallel search algorithms based on the mechanics of natural selection, the process of evolution [3, 2]. GAs were designed to efficiently search large, non-linear search spaces where expert knowledge is lacking or difficult to encode and where traditional optimization techniques fail. They are flexible and robust, exhibiting the adaptiveness and graceful degradation of biological systems and we use them as the search method of choice for this research.

However, problems of interest usually have extremely large search spaces leading to slow progress during search. Furthermore, every time a genetic algorithm is given a new problem, it does not have access to, and cannot use, any knowledge gleaned from previous problem solving attempts to help speed up the current search.

We would like to extract and use knowledge gained during a problem solving attempt to help solve related problems. Case-based reasoning (CBR) systems use past experience, stored in memory as cases, to help solve a current problem[6]. This paper indicates the feasibility of combining genetic algorithm and case-based reasoning principles to attack open-shop scheduling and re-scheduling problems.

We use open shop scheduling (OSSP) and re-scheduling problems as a test-bed for our system. A genetic algorithm finds and saves solutions to an open shop scheduling problem P_{old}, the problem is changed slightly (to P_{new}), and the old solutions are injected into the initial population of the genetic algorithm that is trying to solve the new problem. Preliminary results indicate that compared to a genetic algorithm that starts from scratch, the genetic algorithm with injected solutions very quickly finds good solutions to P_{new} and that the quality of solutions after convergence is usually better.

Genetic algorithms have been used to solve scheduling problems in [1, 4, 7]. Ramsey uses case-based initialization to improve genetic algorithm performance in dynamic environments [5]. We do not know of any research that combines GAs and CBR for solving open shop scheduling.

The next section introduces the open shop scheduling problem. A short section on case-based reasoning is included for completeness and can be skipped by readers already familiar with the material. Section 4 introduces genetic algorithms and describes our encoding for open shop scheduling problems. Section 5 provides preliminary results and outlines issues in combining genetic algorithms with case-based reasoning systems. The last section furnishes conclusions and indicates future directions.

2 Open Shop Scheduling Problems

The open-shop scheduling problem is an important practical scheduling problem, but is known to be very hard to solve in a reasonable amount of time. Re-scheduling is also an important aspect of the problem in the real world. It involves modifying a schedule in the process of execution in order to take account of a changing environment. In the general $j \times m$ open-shop scheduling problem, there are j jobs and m machines. Each job contains a set of tasks which must be done on a different machine for a different amount of time, with no a-priori ordering on the tasks within a job. The problem is to minimize the makespan, the total time between the beginning of the first task till the end of the last task. A valid schedule is a schedule of job sequences on each machine such that a machine is not processing two different tasks at the same time, and different tasks of the same job are not simultaneously being processed on different machines. We must take into account both resources and time constraints with the aim of finding a valid schedule with a minimum makespan. The OSSP differs from the job-shop scheduling problem in that there is no *a-priori* ordering on the tasks within a job. This leads to an extremely large search space. For example on a 5×5 OSSP, there are 25! different task orderings which is approximately equal to 1.5×10^{25} different valid schedules.

In open-shop re-scheduling (OSRP), we need to be able to handle changing conditions on the shop floor. For example, jobs may be canceled, machines may fail, or we may run low on inventory. If the work has not yet begun on the current schedule, then a simple way of re-scheduling is to rerun the GA for this scheduling problem in the changed environment. We need quick and efficient methods of re-scheduling to tackle the problems of frequently changing environments. In this paper we handle the case of a machine breaking down and being replaced with a new, faster machine. P_{old} is the problem with the old machine while P_{new} is the problem with the new machine, where the processing times of all tasks on the new machine have been decreased by 10 time units; note that we have made sure that P_{new} is similar to P_{old}. We use individuals from a GA's run on P_{old} as cases for the re-scheduling problem, P_{new}.

3 Case-Based Reasoning

Case-based reasoning (CBR) is based on the idea that reasoning and explanation can best be done with reference to prior experience. This experience is stored in memory as *cases* [6]. When confronted with a problem to solve, a case-based reasoner extracts the most *similar* case in memory and uses information from the retrieved case and any available domain knowledge to tackle the current problem. The idea is first to collect and index a large collection of examples; then, when presented with a new situation, to fetch and *adapt* the stored example that most closely resembles the new situation and this solve the current problem. This paper uses the basic tenet of CBR – the idea of organizing information based on "similarity" – to help solve open-shop re-scheduling problems.

4 Genetic Algorithms

Conceptually, GAs use the mechanisms of natural selection in evolving individuals that, over time, adapt to an environment. In our case, the individuals were candidate schedules and the genetic algorithm minimized the makespan.

The genetic algorithm works with a population of schedules encoded as strings of integers (genotypes), evaluating and modifying them by probabilistically applying the genetic operators of selection, crossover, and mutation from one generation to the next. The initial population is usually generated randomly; the GA starts from scratch. Selection chooses relatively fitter individuals for crossover and mutation, concentrating search in the area specified by fitter individuals. Mutation insures against the permanent loss of genetic material and with low probability flips a bit in the genotype. Crossover is a structured yet stochastic operator that allows information exchange between candidate solutions. One point crossover is implemented by choosing a random point in the selected pair of strings and exchanging complementary substrings defined by the chosen point. Figure 1 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents.

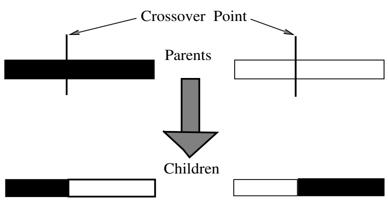

Figure 1: One point crossover.

We represent a schedule as a string of integer pairs. The string (chromosome) contains the information required to construct a candidate solution. In the OSSP, a chromosome contains a series of tasks which represents a schedule. With j jobs, and m machines, there are $j \times m$ tasks. One task is represented by two digits. The second digit represents a job and the first digit represents a task within that job. The total number of digits for a 5×5 problem is $2 \times (5 \times 5) = 50$. For example, consider the chromosome:

55 43 12 32 44 11 15 24 ...

The chromosome contains 25 different tasks with 5 tasks per job and can be decoded from left to right as: 5th task of job 5 will be done first, 4th task of job 3 will be done first, 1st task of job 2 will be done first, 3rd task of job 2 will be done second because task 32 comes after task 12 in the chromosome, and so on.

One-point crossover creates problems when used with our representation because crossover of two parents can produce offspring that encode illegal schedules[2]. To overcome this problem we modify crossover so that although it still mixes information from both parents, it only produces legal offspring.

In PMX [2] a crossing site is randomly selected and the string is divided into two parts. We swap the digits at corresponding positions on the left part of the crossover point under the condition that the same values can be found on the right part of the crossover point.

$$A = 23159|4867$$
$$B = 38247|1956$$

We exchange 5 and 4, and 9 and 7. We can not exchange 2 and 3, 3 and 8, and 1 and 2 because the same values cannot be found on the right part of the crossover point. After crossover, we get the two offspring A' and B':

$$A' = 23147|5869$$
$$B' = 38259|1746$$

Strings A' and B' contain ordering information partially determined by their parents without any repeating digits. We essentially use the same algorithm but swap integer pairs (our representation uses an integer pair to represent a task)

5 Results

After using a variety of criteria to pick individuals for cases and to choose how many individuals to inject into the initial population of size 200 run for 300 generations, we obtained good performance under the following conditions.

1. We inject only 8 cases into the initial population. Injecting between 5% and 10% of the population usually produced good results across the set of population sizes that we tried. Higher injection percentages did not increase performance and often caused the search to converge too quickly.

2. The individuals chosen as cases were:

   - The best individuals in the first and the last generations (Since we used a kind of elitist selection, the best individual in the last generation was the best ever).
   - Six good individuals in the generations between the first and last. For example, if we ran for 300 generations, then we pick the best individuals in generations which are a multiple of $300/6 = 50$.

We mutate these six individuals picked from intermediate generations and inject all eight individuals into the initial population of the GA for the OSRP. Figure 2 compares the average performance of a genetic algorithm initialized with these cases against a randomly initialized GA on one 5×5 OSRP problems. We call the the GA that uses cases, a **CA**se **G**enetic **A**lgorithm or CAGA, and the **R**andomly **I**nitialized **GA**, RIGA, in the rest of this paper. The graphs in this section display the best makespan averaged over 10 runs with different random seeds.

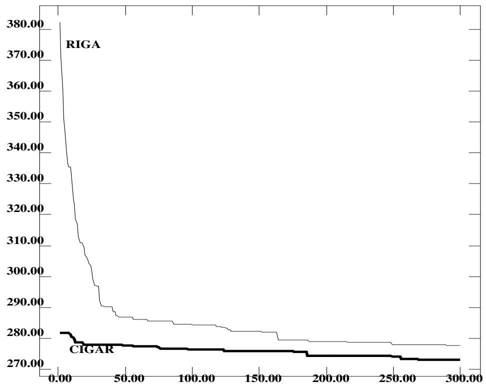

Figure 2: Comparing performance of a RIGA with a CAGA for 5x5 OSRP problem 1

We can see that the CAGA starts with a much lower (better) initial best makespan and that even after 300 generations CAGA solutions are better than solutions from a randomly initialized GA. In our experiments we note that CAGA does comparatively well in early generations and provides good schedules more quickly

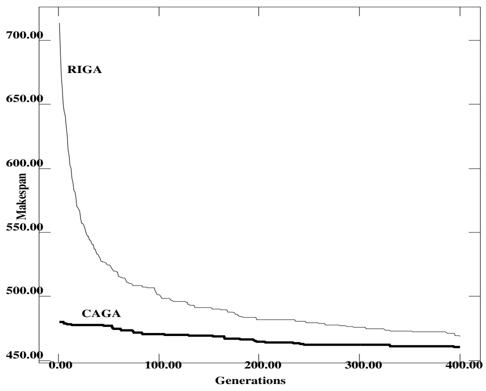

Figure 3: Comparing performance of a RIGA with a CAGA for 7x7 OSRP problem 2.

than the randomly initialized GA. Out of the set of eight benchmark problems, the Case-GA usually produces schedules that are better, except on two problems, where the makespans at generation 300 differ by three time units.

We get similar results on a set of five 7×7 OSRP benchmark problems. Figure 3 displays average best makespans over ten runs with different random seeds on one of the problems. Once again the CAGA does better, especially in early generations.

In early experiments, we were not very successful in combining GAs with CBR principles. First, when the environment was changed by deleting a whole job, the results were not good. We found that it was hard for the GA with CBR to tackle problems where the environment changes significantly. CBR is based on the idea that stored cases resemble a new situation. Since a job contains a series of different tasks, after deleting a whole job, good schedules for the current problem may be very different from the previous one, for our encoding.

We also failed to get good results when we tried to save all, half, or a quarter of the best individuals in the previous run (P_{old}) and insert them into the initial population for P_{new}. Saving and re-using too many good individuals makes the GA quickly converge to local optima. We obtain enough exploration when we seed a small percentage of the population.

6 Conclusions and Future Work

When combining genetic algorithms and case-based reasoning, the results broadly show that there is a tradeoff between performance, both in terms of speed and quality of solutions, and problem similarity. When more interested in quality, we should inject a small number of cases of varying quality into the initial population leading to a better balance between exploration and exploitation of the search space by the genetic algorithm.

Currently, we are researching the quality and quantity of cases to inject. We believe this combination of paradigms has promise and will be applicable in a wide variety of domains.

References

[1] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A promising genetic algorithm approach to job-shop scheduling, re-scheduling, and open-shop scheduling problems. In Stephanie Forrest, editor, *Proceedings of Fifth International Conference on Genetic Algorithms*, pages 375–382. Morgan Kauffman, San Mateo, 1993.

[2] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[3] John Holland. *Adaptation In Natural and Artificial Systems.* The University of Michigan Press, Ann Arbour, 1975.

[4] R. Nakano and T. Yamada. Conventional genetic algorithms for job-shop problems. In R. Belew and L. Booker, editors, *Proceedings of Fourth International Conference on Genetic Algorithms*, pages 474–479. Morgan Kauffman, San Mateo, 1991.

[5] C. L. Ramsey and J. J. Grefenstette. Case-based initialization of genetic algorithms. In Stephanie Forrest, editor, *Proceedings of Fifth International Conference on Genetic Algorithms*, pages 84–91. Morgan Kauffman, San Mateo, 1993.

[6] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Cambridge, MA, 1989.

[7] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In D. Schaffer, editor, *Proceedings of Third International Conference on Genetic Algorithms*, pages 133–140. Morgan Kauffman, San Mateo, 1989.