Proceedings of ACDM 2000
PEDC, Unviersity of Plymouth, UK

Evolving genetic algorithm for Job Shop Scheduling problems

James C. Werner
Mehmet E. Aydin
Terence C. Fogarty

School of Computing, Information Systems and Mathematics
South Bank University
103 Borough Road
London SE1 0AA, UK
{wernerjc,aydinme, fogarttc}@sbu.ac.uk

ABSTRACT

This paper addresses an attempt to evolve genetic algorithms by a particular genetic programming method to make it able to solve the classical Job Shop Scheduling problem (JSSP), which is a type of very well known hard combinatorial optimisation problems. The aim is to look for a better GA such that solves JSSP with preferable scores. This looking up procedure is done by evolving GA with GP. First we solve a set of job shop scheduling benchmarks by using a conventional GA and then an association of GP to evolve a GA. The instance of JSSP tackled are available in OR literature.

INTRODUCTION

Scheduling, especially job shop scheduling, has been studying for a long time. Because of its NP-Hard nature, there has not been found a global problem solver for this kind of problems. Recently, some meta-heuristics like Simulated Annealing (SA), Taboo Search (TS), and Genetic Algorithms (GA) have been implemented as pure methods and hybrid of different method, where the hybrid methods are superior over pure ones. The main problem is how to cope with local minima within a reasonable time. Among them, GA has been studied and implemented to like the other problems with moderate score of success. The problem with GA is how to find the most effective types and order of operators to evolve the solutions space, where one operator may be successful for one type of problem with a certain order and may not be so preferable for some other problems. Our aim is to make clear that if it is possible to evolve a GA by Genetic Programming (GP) methods in such a way that GP tries different crossover, mutation and selection operators in various orders.

In the following three sections we give brief descriptions of JSSP, GA, and how to apply GA to JSSP. After that, the genetic control method, genetic programming to evolve GA, and its application to JSSP is presented. Then a comparison with a particular SA method is explained and paper concluded.

JOB-SHOP SCHEDULING PROBLEMS (JSSP)

The JSSP consists of a number of machines, M, and a number of jobs, J. Each job consists of M tasks, each of fixed duration. Each task must be processed on a single specified machine, and each job visits each machine exactly once. There is a predefined ordering of the tasks within a job. A machine can process only one task at a time. There are no set-up times, no release dates and no due dates. The makespan is the time from the beginning of the first task to start to the end of the last task to finish. The aim is to find start times for each task such that the makespan is minimised. As a constraint problem, there are M*J variables, each taking positive integer values. The start time of t^{th} task of the j^{th} job will be denoted by x_{jt}, and the duration of that task by d_{jt}. Each job introduces a set of *precedence* constraints on the tasks within that job: $x_{jt} + d_{jt} \quad x_{j(t+1)}$ for $t = 1$ to $M-1$. Each machine imposes a set of *resource* constraints on the tasks processed by that machine: $x_{jt} + d_{jt} \quad x_{pq}$ *or* $x_{pq} + d_{pq} \quad x_{jt}$. The aim is to find values for the variables such that no constraint is violated. By defining an objective function on assignments (which simply takes the maximum of $x_{jt} + d_{jt}$), and attempting to minimise the objective, we get a constraint optimisation problem. (See for more information [1] and [2]).

THE GENETIC ALGORITHM APPROACH.

The genetic algorithms (GA) mimic the evolution and improvement of life through reproduction, where each individual contributes with its own genetic information to build up new ones adapted to the environment with higher chances of survival. This is one of the main ideas behind genetic algorithms and genetic programming (Holland [5], Goldberg [6], and Koza [7]). Specialized Markov Chains underline the theoretical basis of GA in terms of change of states and search procedures [8]. Each 'individual' of a generation represents a feasible solution as coded in a chromosome with distinct algorithms /parameters to be evaluated by a fitness function. GA operators are mutation (the change of a randomly chosen bit of the

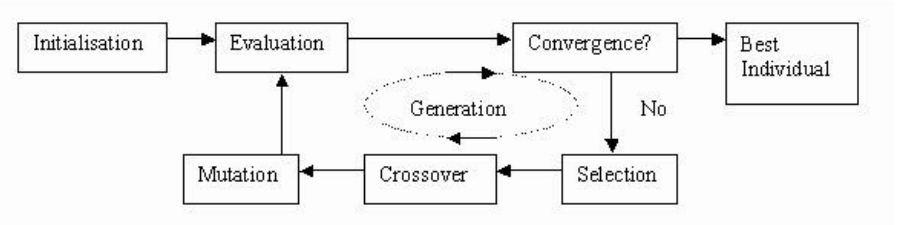

Fig. 1 Genetic algorithm: the sequence of operators and evaluation of each individual.

chromosome) and crossover (the exchange of randomly chosen slices of chromosome).

Fig.1 shows a generic cycle of GA where the best individuals are continuously being selected and operated by crossover and mutation. Following a number of generations, the population converges to the solution that performs better.

GA applications for JSSP have special chromosome representation as well as genetic operators to be applied to feasible schedules [4]. In our case, the chromosomes are coded as a list of sets of numerical values for each particular schedule. A generalisation of the GA is the Genetic Programming (GP) algorithm where each 'individual' in a generation represents, with its chromosome, a feasible model solution, in our case, a sequence of genetic operators that will define one genetic algorithm. There are two kinds of information defined for the GP algorithm: terminals (variable values and random numbers) and functions (mathematical functions used in the generated model).

GENETIC ALGORITHMS IN JSSP

We have generated schedules in a particular way in which the chromosome will be feasible after performing genetic operators. The decision management in JSSP distributes the jobs for each machine, selecting sometimes one task among the other alternatives so as to have a better fitness. We code a chromosome with M*J values between 0 and 1, one for each decision, which points for the job in the requestin g jobs list that win the right to use the machine. Fig. 2 shows an example of chromosome coding based in decision process.

This approach allows using the same traditional GA operators to solve the problem because the chromosome contains a sequence of numb ers, all representing feasible schedules. Two fitness functions can be applied to evaluate a solution: the makespan and the total idle time. The main problems with this approach are as follows:-

- the disruptive effect of crossover operator,
- the precocious convergence to some local minima that blinds the system to find the global one,
- eventually raising the inability of genetic operators to permute the solutions in a reasonable time,
- the lack of hill climbing in GA.

The total size of the population is 100 individuals and the number of generations is number machines*number job *10, the crossover probability is 80% and mutation probability is 25%, and each decision is coded as 8 bits. The increase of population size does not change the results too much, but increase the processing time. For example, in LA20 problem using makespan fitness, the final result was 959 time units after 35 generations of 900 individuals (no better solution was founded until 149,500 generations) and 979 time units with 100 individuals. Table 1 shows the results for both fitness functions at columns "GA makespan" and "GA Idle time".

GENETIC PROGRAMMING GENERATING GENETIC ALGORITHMS – THE GENETIC CONTROL APPROACH

We used GP to obtain what genetic operators and in which order would be applied to solve the problem. Fig 3 shows the algorithm and the relation between GP and GA. The possible operators are: usual crossover, usual mutation, extend the limits for some decisions by 10% in schedule generation, narrow the limits for some decisions by 10% in schedule generation, exchange of some region of one parent chromosome, exchange of the remainder chromosome of one parent chromosome. In this case, we are evolving the genetic algorithm with genetic programming. Table 1 presents the

Job	Tasks		
0	1	0	2
1	**0**	1	2
2	2	1	0
3	1	2	0

Decision 1: machine 0 receives job 1, any value of the chromosome (only one candidate).

Job	Tasks		
0	**1**	0	2
1	**1**	2	
2	2	1	0
3	**1**	2	0

Decision 2: machine 1 could receive jobs 0, 1 or 3. If chromosome value is 0.7, job 3 begins

Job	Tasks		
0	1	0	2
1	1	2	
2	**2**	1	0
3	**2**	0	

Decision 3: Machine 2 is required by jobs 2 and 3. If chromosome >0.5 then job 3 executes, otherwise job 2.

Fig. 2 Decisions sequence in JSSP.

results for both fitness functions at columns "GC makespan" and "GC idle time", where GP parameters are 50 individuals and 50 generations and GA parameters are 900 individuals, 100 generations, crossover and mutation probability 50%, and each decision is coded in 4 bits.

RESULTS AND ANALYSIS

We have obtained many results for a set of benchmark problems for both conventional and evolved GA applications. In order to measure our efficiency we compare our results with an instance of SA used for the same benchmarks. This SA approach is called as modular simulated annealing (MSA) algorithm, which is a multi-start SA algorithm employed with a population rather then an individual. The details of this method is given in [10] and [3]. The idea of that work was to evolve a population of solutions by applying a modular SA constantly to selected solutions. Table 1 shows the results of MSA in column "MSA".

Table 1 shows the results of several problems of OR literature available in internet [9]. The columns "Time" contains the makespan and "%" contains the percentage of the optimal value (column "Opt.").

Fig. 4 shows the percentage of error for GA, GC and MSA against the number of decisions, sorted by the %error. The number of decisions defines the solution space size and is not related with the complexity problem. A little problem can be more complex than a big one. The problem size and complexity, as shows the increase of % error affect all approaches, where it is more effective on GA than MSA due to the possibility of MSA performs climbing hill, while genetic operators produces a disruptive effect in the solution. The processing time exponentially increases generating GA by GP because of the increasing the number of evaluation. The idle time and makespan are equivalent approaches for fitness function. The decision point-of-view for chromosome coding allows the construction of only feasible solutions, with the increase of processing efficiency.

Begin

⇨ *Initialise* the **population** of GP,

Repeat for each individual of GP:

- *pick* two individuals with Roulette search,
- *crossover* and *mutation* operators
- evaluate *fitness* for each individual applying the GA coded in individual chromosome: Repeat for 100 generations:
  - Initialise the population
  - Execute GA operators and obtain children
  - Evaluate the children;

  The best GA and its final fitness are used by GP;

End.

Fig. 3 Genetic control algorithm.

Table 1: results obtained from Genetic algorithms, Genertic Control and Simulating annealing.

	# Dec.	Opt.	GA makespan		GA Idle time		GC makespan		GC Idle time		MSA	
			Time	%	Time	%	Time	%	Time	%	Time	%
ABZ5	100	1234	1313	6.4	1314	6.4	1253	1.5	1244	0.8	1234.4	0
ABZ6	100	943	994	5.4	982	4.1	952	0.9	951	0.8	943.0	0
LA18	100	848	940	10.8	897	5.7	877	3.4	861	1.5	848.0	0
LA17	100	784	872	11.2	854	8.9	803	2.4	803	2.4	784.0	0
LA20	100	902	979	8.5	1006	11.5	939	4.1	925	2.5	902.0	0
LA16	100	945	1031	9.1	1036	9.6	979	3.9	978	3.4	945.0	0
LA19	100	842	945	12.2	965	14.6	883	4.8	877	4.1	842.0	0
ORB01	100	1059	1230	16.1	1229	16.0	1156	9.1	1118	5.5	1074.0	1.4
LA25	150	977	1129	15.5	1207	23.5	1100	12.5	1042	6.6	977.8	0
FT10	100	930	1032	10.9	1010	8.6	1017	9.3	1005	8.0	937.0	0.7
LA24	150	935	1091	16.6	1083	15.8	1040	11.2	1013	8.3	939.6	0.4
LA21	150	1046	1222	16.8	1260	20.4	1198	14.5	1164	11.2	1048.4	0.1
LA27	200	1235	1507	22.0	1538	24.5	1445	17.0	1415	14.5	1245.4	0.8
ABZ7	300	655	779	18.9	787	20.1	763	16.4	755	15.2	675.2	3.0
LA38	225	1196	1476	23.4	1485	24.1	1407	17.0	1382	15.5	1214.6	1.5
LA40	225	1222	1569	28.3	1540	26.0	1473	20.5	1426	16.6	1229.2	0.6
LA29	200	1130	1422	25.8	1501	32.8	1391	23.0	1346	19.1	1182.6	4.6
ABZ9	300	656	795	21.1	843	28.5	806	22.8	786	19.8	703.0	7.1
ABZ8	300	638	823	28.9	827	29.6	810	26.9	779	22.1	687.2	7.6

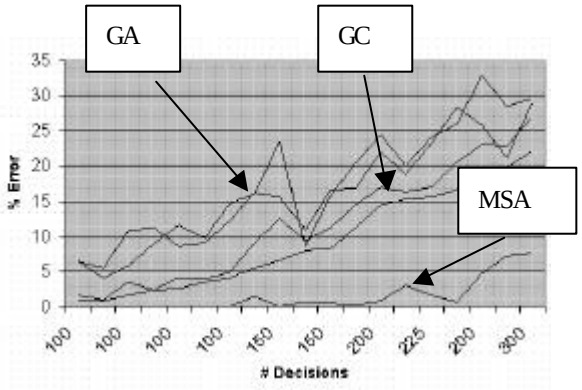

Fig. 4 Number of decisions against the percent error for each example.

CONCLUSION.

Algorithms generated by GP are competitive with specifics techniques of optimisation in JSSP. The limitations are the susceptibility of solution space increase and hill climbing limitation. The introduction of SA in replacement operator gave to GA the possibility to do hill climbing [10 and 11] and obtain betters results.

The next step of Evolvable Genetic Algorithms will be the study of other possible operators that improve the hill climbing freedom of GA and the study of conform transform that creates a mo re smooth and treatable landscape for JSSP problem.

REFERENCES:

1. BAKER, K. R., 1974, *Introduction to Sequencing and Scheduling*, John Wiley & Son.
2. BALAS, E., 1969, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm", Operations Research, 17, 941-957.
3. AYDIN M. E. AND FOGARTY, T. C, 2002, "Simulated annealing with evolutionary processes in job shop scheduling", in Giannakoglou, K., Tsahalis, D., Periaux, J., Papailiou, K. and Fogarty, T. C. (eds.) Evolutionary Methods for Design, Optimisation and Control, (Proc. of EUROGEN 2001, Athens, 19-21 September) CIMNE, Barcelona
4. VAZQUEZ,M.; WHITLEY, D., 2000, "A comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem", 6th Int. Conf. Parallel Problem Solving from Nature – PPSN VI, in Lectures notes in Computing Science vol. 1917, pg. 303-312.
5. HOLLAND,J.H., 1992, *"Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*." Cambridge: Cambridge press.

6. GOLDBERG,D.E., 1989, *"Genetic Algorithms in Search, Optimisation, and Machine Learning."* Reading,Mass.: Addison-Whesley.
7. KOZA,J.R., 1992, *"Genetic programming: On the programming of computers by means of natural selection."* Cambridge,Mass.: MIT Press.
8. RUDOLPH,G., 1994 *"Convergence Analysis of Canonical genetic algorithms."* IEEE Transactions on Neural Networks, v.5, p.96.
9. Imperial College Management School: OR Library http://www.ms.ic.ac.uk/info.html
10. AYDIN M. E. AND FOGARTY, T. C., 2002, "Modular Simulated Annealing Algorithm for Job Shop Scheduling running on Distributed Resource Machine (DRM)" Submitted to "Parallel Problem solving from Nature 2002" Granada, Spain Sept 7 to 11, 2002
11. AYDIN M. E. AND FOGARTY, T. C, 2002, "Modular Simulated Annealing: an empirical investigation with Job-Shop scheduling problems" South Bank University internal report.