Job-Shop Scheduling with Genetic Programming

Kazuo Miyashita
Electrotechnical Laboratory (ETL)
1-1-4, Umezono, Tsukuba,
Ibaraki 305-8568, JAPAN
(e-mail) miyasita@etl.go.jp
(tel & fax) +81-298-61-5963

Abstract

In order to solve a real-time scheduling problem, a computationally intensive search-based optimization method is not practical, but the efficient dispatching rule that is well-customized for the specific problem at hand can be an effective problem solving method. A dispatching rule is scheduling heuristics that decide the sequence of operations to be executed at each resource in the scheduling problem. However, developing a customized dispatching rule for specific scheduling problems is an arduous task even for domain experts or researchers in the scheduling problem. In this research, the author views scheduling problems as multi-agent problem solving and proposes an approach for synthesizing the dispatching rule by means of Genetic Programming (GP). In the preliminary experiments, the author got the results showing that GP-based multi-agent dispatching scheduler outperformed the well-known dispatching rules.

1 INTRODUCTION

In the research of production scheduling problems, plenty of search-based methodologies (e.g., constraint-based scheduling) have been developed and deployed in the real operational environment as well as in the research (Zweben & Fox 1994). Those methods search for an optimal or semi-optimal schedule in terms of multiple objectives, such as minimum tardiness or maximum machine utilization, while satisfying several constraints imposed on jobs and resources in a problem, such as temporal precedences and limited capacity. Since the scheduling problems tend to be large and complex, the efficiency of search process always becomes the biggest burden for those methods to be applied. The author has been applying machine learning technologies such as case-based reasoning and reinforcement learning to elicit search control knowledge from iterative schedule optimization process and shown that quality and efficiency of scheduling can be largely improved by exploiting the acquired knowledge (Miyashita & Sycara 1995a; 1995b).

In addition to building an optimal (or semi-optimal) schedule, there is another important requirement to the scheduling system: reaction to the unpredictiveness (Smith *et al.* 1990). The scheduling system must be able to control a complex production floor timely in response to the unpredictable events in manufacturing process such as machine breakdowns and material delays. To avoid interrupting continuous production in the face of those events, production schedule must be modified or re-built from scratch as quickly as possible. This is a strong demand especially in the semiconductor fabrication process because the interruption of production always causes a severe loss of product's quality and drasticly decreases its yield.

In such a dynamic and uncertain environment, search-based scheduling techniques are not practical because of their computational cost, but scheduling with simple but powerful rules that take advantage of a specific structure of the target problem are suitable and highly effective. For example, it is well known that in the one-machine problem such a simple rule that dispatches the jobs in the ascendant order of their processing time makes the optimal schedule in terms of the average lead time of the jobs.

Unfortunately simple rules such as the one described above are not always optimal to the more complicated real-world scheduling problems. Hence, in the realistic scheduling problems, more elaborated heuristic rules have been developed by domain experts or researchers in the field. However, developing the good rules that are well customized to the target problem is an arduous task and takes long time and huge efforts.

In this research, the author views the scheduling problem as a model of multi-agent problem solving and proposes a method for automated generation of each

agent's scheduling heuristics. There have been several research activities on inducing heuristics for scheduling, but those heuristics are the rules to classify a predefined set of simpler rules to the scheduling situations in which they are effective (Shaw 1989). The approach in this research is different from the previous research in that it synthesizes the scheduling heuristics from scratch using Genetic Programming (GP). In the next section, the author describes how GP is applied to the scheduling problems in this research. Then, multi-agent scheduling architecture is introduced for improving effectiveness of heuristic scheduling and several multi-agent models are proposed for experiments. Lastly, the results of those preliminary experiments are presented and analyzed.

2 APPLYING GP TO SCHEDULING PROBLEMS

In this research, the type of scheduling problems to be solved is *job-shop scheduling* that deals with allocation of a limited set of *resources* to a number of *operations* associated with a set of *jobs* (French 1982). The dominant constraints in job-shop scheduling are temporal operation precedence and resource capacity constraints. The operation precedence constraints along with a job's release time and due date restrict the set of acceptable start times for each operation. The capacity constraints restrict the number of operations that can use a resource at any particular point in time and create conflicts among operations that are competing for the use of the same resource at overlapping time intervals. Therefore, to proceed scheduling, the resource needs to select an operation for next execution among competing operations when the resource becomes available after finishing its current operation.

A rule for selecting the next operation to be processed at the resource is called as a *dispatching rule*. Basically the dispatching rule simply selects the operation with the highest priority. Hence, it is important to decide the priority of operations reasonably so that a good schedule is created. Several sophisticated dispatching rules have been developed for a variety of scheduling problems with different problem structures and objectives. But developing a good dispatching rule is very difficult even for domain experts because of complexity of scheduling problems. In this research, the heuristics to calculate the operation's priority are generated by *Genetic Programming* (GP) (Koza 1992).

GP extends the representation scheme of Genetic Algorithm (GA) into general, hierarchical computer programs of dynamically changing size and shape. In GP, the process of solving problems is reformulated as search for a highly fit individual computer program in the space of the possible computer program. Fitness of the program is determined based on quality and efficiency of the program in solving a target problem. In this research, fitness of the generated program is quality of a resultant schedule which is the biased combination of weighted tardiness and WIP [1].

2.1 Terminals and Functions

In GP, the search space is the space of all possible computer programs composed of *functions* and *terminals* appropriate to the problem domain. Therefore, when applying GP to the specific target problem, one needs to define carefully the function set and the terminal set that are useful for solving the problem. Especially, terminals should represent the idiosyncratic attributes of the problem to extract useful information for problem solving. In the complicated problems such as scheduling, the number of problem attributes — possible candidates of terminals — is extremely large : every detail of the schedule (i.e., any temporal relation among operations and resources) can be a useful terminal in some situation. Hence choosing the sufficient and smallest attribute set is important for reducing the search space and improving efficiency of GP. The terminals and functions used in this research are selected based upon the author's past work on scheduling and shown in Table 1.

Table 1: Terminals and Functions

Name	Description
DueDate	Due date of the job
CurrentTime	Current time
ProcessTime	Processing time of the operation
RemainingLeadTime	Remaining processing time till completion of the job
RemainingProcess	Remaining number of operations in the job
TimeSeverirty	Tardiness penalty of the job
WaitingTime	Waiting time of the operation
+	Addition
-	Subtraction
*	Multiplication
%	Division
iflte	If the first argument is less than or equal to the second argument, execute the third argument, else execute the fourth argument

In GP-based dispatching, whenever a resource be-

[1] WIP is an abbreviation of Work-In-Process inventory, which means inventory of materials and semi-finished products.

comes available for next operation, the priority of operations waiting for the resource is re-calculated using the GP-generated program and the operation with the highest priority is chosen for next execution on the resource. The priority of operation is not fixed over time but can change dynamically as scheduling proceeds. Among the terminals in Table 1, $CurrentTime$ represents simulation time of scheduling process and is used to calculate the change of operation priority as time elapses (for example, slack time of a job to the due date changes as time passes away). And, $WaitingTime$ stands for the elapsed time since the previous operation in the job is processed. This terminal might be effectively exploited for coordinating the interactions among resources that are in charge of adjacent operations in the job and reducing unnecessary waiting time intervals among operations.

Once effective heuristics for dispatching are generated by GP, they can be used for the following purposes in scheduling:

- **Real-time scheduler**: since heuristics are generated off-line in advance, they can be applied in real-time scheduling.
- **Initial scheduler**: since GP generated heuristics are expected to be able to produce a schedule of high quality, it can be used to make an initial schedule for succeeding iterative optimization and make the optimization process more efficient.

3 MULTI-AGENT DISPATCHING

In general, a dispatching rule can generate a schedule efficiently, but in most cases quality of the produced schedule is poor except in the very simple idealistic problems such as one-machine scheduling problems. To remedy the defect and deal with the complicated job-shop scheduling problems, *multi-agent dispatching* architecture is proposed as shown in Fig. 1.

In multi-agent dispatching, each resource or a group of resources can behave as an individual agent that has its own dispatching rule. The dispatching rule needs to be customized for each agent's specific requirement to produce an optimal schedule for each agent. Each dispatching rule makes a schedule for each agent (i.e., a resource) and schedules of the whole agents are integrated into a single schedule. But in the scheduling problem, combination of locally optimal sub-schedules does not always constitute a globally optimal schedule. To achieve a common goal (i.e., maximizing a global profit), agents are sometimes asked to sacrifice their own local goal and cooperate each other. In other words, dispatching rules also need to be coordinated among other agents. To pursue a consistent objective of scheduling such as minimizing WIP, agents are required to respond to the diverse situations without interfering other agents' behavior. Even in a small

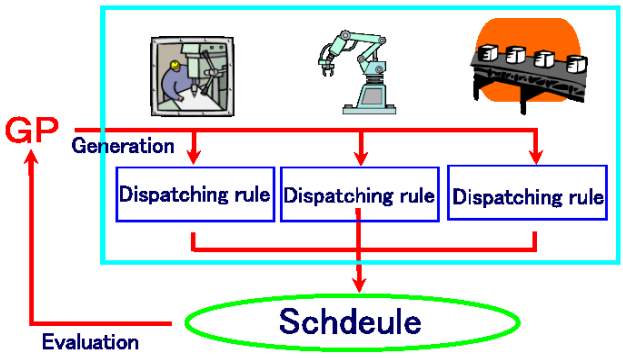

Figure 1: Multi-agent Dispatching

scheduling problem, the rule to generate such a coordinated action of agents easily gets too complicated to be hand-coded a priori by human-experts.

The author tries to solve this problem by learning a dispatching rule of each agent with GP. A group of dispatching rules that behave in a coordinated manner are expected to emerge through evolution of multiple agents. Local interactions of agents' dispatching rules create a whole schedule, whose objective value acts as an external constraint on agents' action and adjusts behavior (i.e., dispatching rules) of agents into more coordinated one. Through the cycle of these bottom-up and top-down interactions among agents and created schedules, more sophisticated dispatching rules for agents are developed and a better schedule can be generated.

3.1 Multi-Agent Learning by GP

In the past research on GP, several researchers applied GP to multi-agent learning problems: robot navigation problem (Iba 1997; 1998), RoboCup soccer problem (Luke 1998) and pursuit problem (Haynes & Sen 1997). In their research as well as this research, each agent uses GP to learn a program to control its behavior by exchanging some information among other agents and giving feedback to its GP process. In these problems, there are following difficulties to be solved:

- **Close cooperation among heterogeneous agents**

  Agents need to cooperate with other agents to produce a consistent result as a team. And at the same time, each agent's behavior should be customized to reflect the individual capability given as a problem specification and adapt dynamically to the situation around the agent. In scheduling problems, agent's idiosyncrasy comes from the the following facts: (1) each agent (i.e., a resource)

can process only specific types of operations, and (2) a load of each agent varies dynamically according to the jobs released in the job-shop and operation rate of other agents.

- **Incomplete information sharing**
  Each agent does not know much about the real-time situation of other agents due to the limited communication capacity among agents. Hence, the agent should reason other agents' behavior and adjust its own behavior to make a rational decision in uncertain situations. In scheduling problems, each resource does not know in advance what sort of operations are coming and when they are coming, since it depends on the behavior of other resources. Therefore the resource should always be able to respond to the unpredicted operations.

Learning the agent's control program that solves the above difficulties is itself very hard even in the simple toy problems in the past research. The scheduling problem differs from other problems in that it should realize more complete coordination among agents in order to obtain a rational result (i.e., a good schedule) since disharmony of any single agent's behavior (e.g., dispatching the last-in operation first while the other agents dispatch the most urgent operation first) drasticly damages the quality of a resultant schedule.

3.2 Models of Multi-Agent Structure

In this research, each individual tree evolved by GP acts as a program to give a priority to operations waiting for a resource (i.e., an agent). Because of tight interactions among agents in multi-agent dispatching, an action of any agent can make significant effects on a resultant schedule. Therefore, it is difficult to ascribe the achieved quality of the schedule to the behaviors of each agent. In this research, to avoid the problem of credit assignment, quality of an entire schedule, which is composed of each agent's sub-schedule, is fed back to each agent to evaluate the performance of its GP-generated dispatching rule. Evaluation of each agent's behavior with a global result promotes cooperation among agents.

In order to investigate effectiveness of multi-agent dispatching in job-shop scheduling problems, the author sets up 3 models of multi-agent structure.

- **Homogeneous agent model** (left in Fig. 2)
  Every resource utilizes the same heuristics evolved by a single GP process to calculate the priority of operations. The author assumes that in this model, coordination among agents (such as consistently giving a higher priority to an operation with earlier due date) can be easily realized since all agents take actions based on the identical heuristics. On the other hand, since every resource shares the same dispatching rule, the rule needs to become very complicated to deal with the idiosyncrasies among resources and produce a globally good schedule. Evolving such a rule with GP may take long time.
- **Distinct agent model** (center in Fig. 2)
  Each resource acts as a unique agent and learns distinct heuristics to evaluate the priority of operations on its queue. Each agent has its own population of individuals to evolve with GP. There is no immigration of individuals and crossover of individuals among populations in different agents. In the model, each agent can improve quality of its sub-schedules by customizing its behavior to its situated environment and optimize its throughput. But coordination among resources becomes difficult in this model.
- **Mixed agent model** (right in Fig. 2)
  In the scheduling problem, it is well known that the quality of overall schedule can be largely improved by optimizing the sub-schedules of *bottleneck* resources (Adams, Balas, & Zawack 1988). A bottleneck resource is a resource that has high demands from many operations. Hence, exploiting the bottleneck resource effectively is critical for producing a good schedule. In this model, 2 types of agents — bottleneck agent and non-bottleneck agent — learn independent heuristics for operation priority evaluation. Each agent has its own population of individuals to evolve with GP. There is no immigration of individuals and crossover of individuals among populations in different agents. This model is intermediate of the above two models and is expected to be effective in avoiding over-fitting and over-generalization problems in the above models and also successful in improving efficiency of evolution process by GP.

In realistic scheduling environment, the bottleneck resources are not fixed but may change dynamically while scheduling proceeds. Therefore, in the mixed agent model it is not practically valid idea that resources are classified into bottleneck or non-bottleneck in advance and cannot change their class during scheduling. But, the author's intention in this paper is to investigate empirically whether bottleneck can be dominant information to model the structure of scheduling agents. The dynamic structuring of agents is future research concern.

4 SCHEDULING PROBLEMS

The author evaluated the approach on a benchmark suite of job shop scheduling problems where parameters, such as the number of bottlenecks and the range of due dates and operation durations, were varied to cover a broad range of job shop scheduling problem instances. In particular, the benchmark problems have

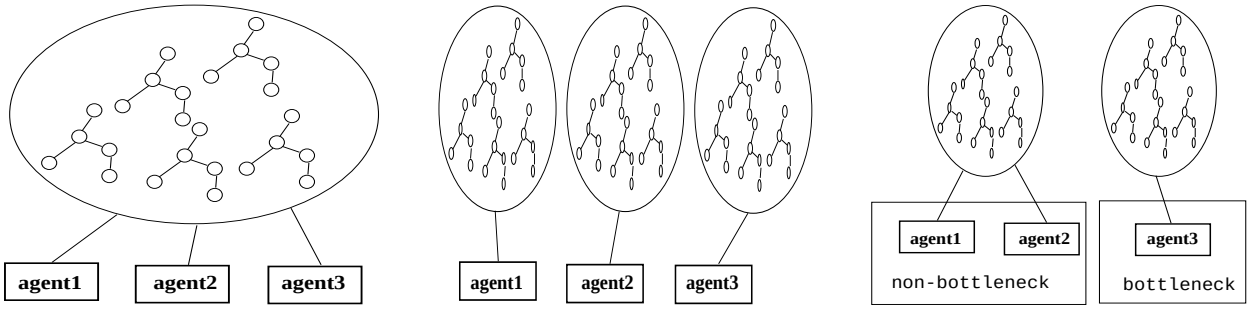

Figure 2: Three Types of Multi-agent Model

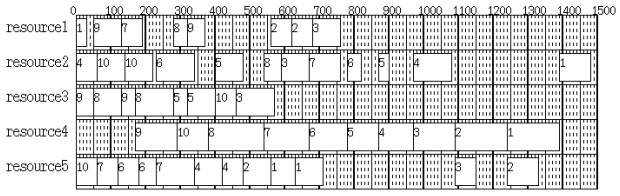

Figure 3: One Bottleneck Problem

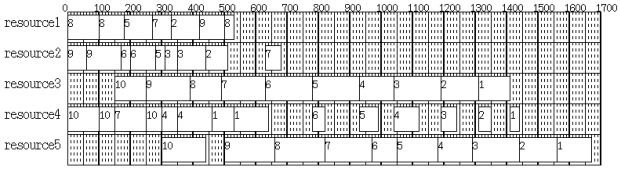

Figure 4: Two Bottlenecks Problem

the following structure: each problem has 10 jobs of 5 operations each. Each job has a linear process routing specifying a sequence where each job must visit bottleneck resources after a fixed number of operations, so as to increase resource contention and make the problem tighter. 2 parameters were used to cover different scheduling conditions: a range parameter controlled the distribution of job due dates and release dates, and a bottleneck parameter controlled the number of bottleneck resources. To ensure that knowledge of the problem had not been unintentionally hardwired into the solution strategies, the author used a problem generator function that embodied the overall problem structure described above to generate job shop scheduling instances where the problem parameters were varied in controlled ways. In particular, 6 classes of 10 problems each — in all, 60 problems — were randomly generated by considering 3 different values of the range parameter (static, moderate, dynamic), and 2 values of the bottleneck configuration (one and two bottleneck problems, see Figs. 3,4). The slack was adjusted as a function of the range and bottleneck parameters to keep demand for bottleneck resources close to a hundred percent over the major part of each problem. Durations for operations were also randomly generated. And the objective of scheduling in the experiments is minimizing the biased combination of weighted tardiness and WIP.

To evaluate the performance of GP-evolved heuristics, a 2-fold cross-validation method was used. Each problem set of one and two bottleneck problems was divided into 2 groups with the same size (i.e., each group has 15 problems). All problems in one group were used as a training set to evolve scheduling heuristics by GP. And the best individual of evolved heuristics were applied to the problems in the other group for validation. This process is repeated by swapping a training set and a validation set, and the average results are reported.

Table 2: Results by Typical Dispatching Rules

	1-BN problem	2-BN problem
EDD	8838.7	8569.3
SPT	8446.7	8677.3
MORTON	8372.3	8648.7

And also to compare the multi-agent dispatching results of various models with the typical dispatching rules, 3 types of the well-known dispatching rules (i.e., EDD (Earliest Due Date first), SPT (Shortest Processing Time first) and MORTON[2](Morton & Pentico 1993)) were applied to the problems. These rules are widely used in research and practice.

Table 2 shows the performance of 3 dispatching rules, which is the average result of 30 instances on one bottleneck problems and two bottlenecks problems. In the table, EDD is the worst dispatching rule for one

[2]MORTON dispatching rule is mixture of EDD and SPT: it dispatches the operation which belongs to the job with close due date and has short processing time on the resource.

bottleneck problems, while it is the best rule for two bottlenecks problems. From the result, it turns out that performance of the dispatching rule is heavily dependent on the structure of scheduling problems. In the realistic scheduling situations, the problem structure is not stable but flexible due to variance of the environment such as fluctuation of demands and unpredictive machine breakdown. Hence, it is desirable that one can always adapt scheduling heuristics to the current problem to produce a good schedule consistently.

5 EXPERIMENTAL RESULTS

The GP system used in the experiment was developed based on SGPC (Simple Genetic Programming in C) version 1.1 (Tackett & Carmi 1994). In the experiments, each agent breeds 1024 individuals and evolution process was continued until 200th generations. The details of SGPC parameters used in the experiments are shown in Table 3.

To examine relation between problem difficulty and types of multi-agent model in terms of problem solving performance (i.e., quality of resultant schedules in this experiment), the experiments were done using one-bottleneck problems and two-bottleneck problems separately. In general, problems with more bottleneck resources are more difficult to schedule than ones with less bottleneck resources because more bottleneck resources increase contention among operations and require more coordination among resources and operations for producing a good schedule.

5.1 One Bottleneck Problems

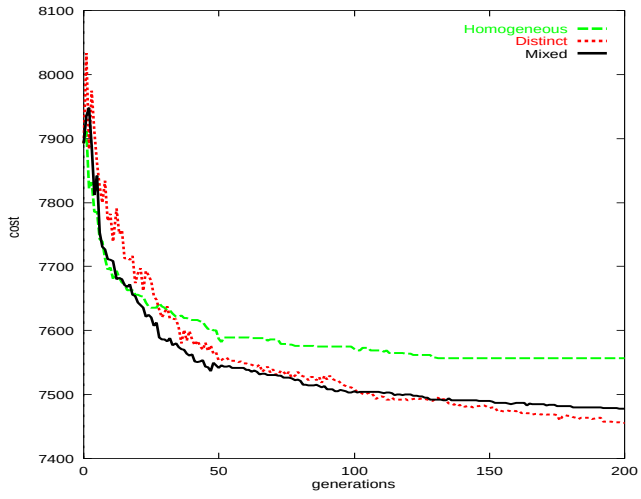

Figure 5: Result of Best Individual (1-BN problems)

These problems are designed so that the 4th resource becomes a bottleneck resource as shown in Fig. 3.

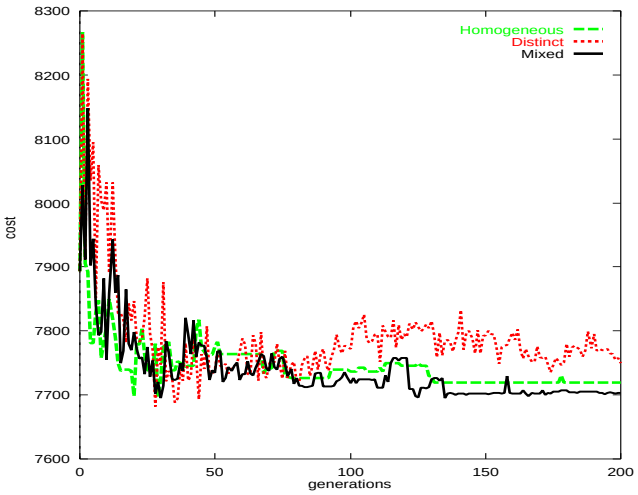

Figure 6: Validation Result (1-BN problems)

Fig. 5 is the graph showing the results of the best individual in the training set for 3 types of the agent model. The graph shows that in earlier generations the results of the homogeneous agent model is as good as the mixed agent model, but as the generation proceeds the results of the distinct and mixed agent models outperform the homogeneous agent model.

But as results of 2-fold cross validation Fig. 6 shows that the results of the distinct agent model are worse than the results of the homogeneous and mixed agent models in later generations. This explains that in the simple problems of one bottleneck resource, the flexibility of the distinct agent model causes over-fitting to the training data. Among 3 agent models, the mixed agent model succeeded to produce the best results both in training and validation sets. Around 200th generations the result of validation by the mixed agent model was better than the best dispatching rule in Table 2 by about 670.

5.2 Two Bottlenecks Problems

These problems are designed so that both 3rd and 5th resources become bottleneck resources in the schedule as shown in Fig. 4.

Fig. 7 is the graph showing the results of the best individual in the training set for 3 types of the agent model. The graph shows that in the two bottlenecks problems, the homogeneous agent model was not able to produce the effective scheduling heuristics because in these problems a scheduler needs to consider the complicated interactions among bottleneck resources to make a good schedule. And in the experiments, the mixed agent model could learn good heuristics faster than the distinct agent model. This is because more information of the problem structure (i.e., which re-

Table 3: GP Parameters

	Homogeneous model	Distinct model	Mixed model
No. of agents	1	5	2
No. of sub_populations	6120	1024	3060
No. of populations	6120	6120	6120
No. of Terminals	7	7	7
No. of Functions	5	5	5
max_depth_for_new_trees	4	4	4
max_depth_after_crossover	10	10	10
max_mutant_depth	2	2	2
grow_method	RAMPED	RAMPED	RAMPED
selection_method	OVERSELECT	OVERSELECT	OVERSELECT
crossover_fuct_pt_fraction	0.15	0.15	0.15
crossover_any_pt_fraction	0.65	0.65	0.65
fitness_prop_repro_fraction	0.1	0.1	0.1

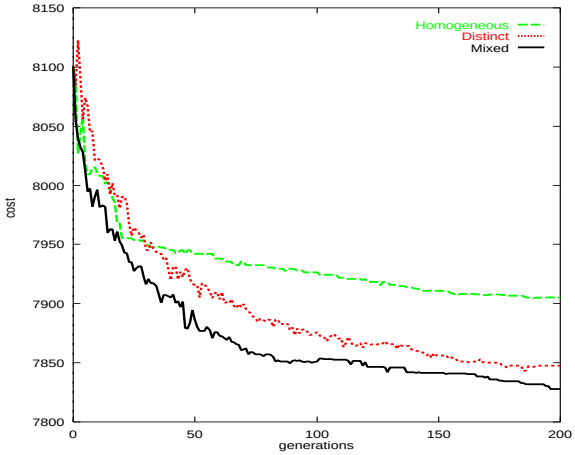

Figure 7: Result of Best Individual (2-BN problems)

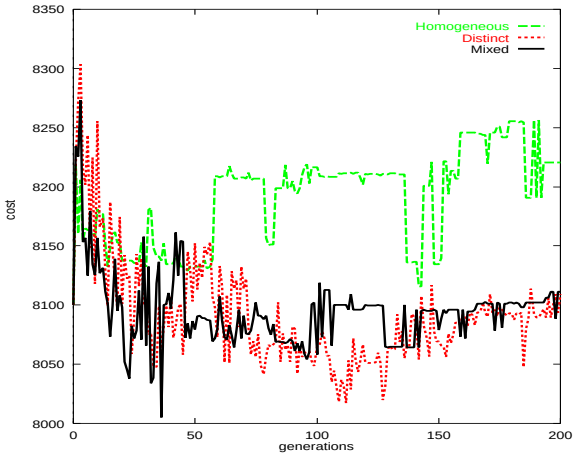

Figure 8: Result of Validation (2-BN problems)

sources are bottleneck and which resources are not) is embedded in the mixed agent model than in the distinct agent model.

The results of 2-fold cross validation Fig. 8 shows inadequacy of the homogeneous agent model for the two bottlenecks problems. On the other hand, in the mixed and distinct agent models the scheduling results were improved along generations. Around 200th generations the results of the mixed and distinct agent models were better than the typical dispatching rules in Table 2 by about 460.

6 CONCLUSION

This paper discussed the evolutionary synthesis of job-shop scheduling heuristics with Genetic Programming. In the paper the author empirically showed that the effective scheduling heuristics can be evolved by GP. The author proposed the multi-agent dispatching system where each agent dispatches the operations on the resources under its control. The rules for dispatching used by the agents are evolved by GP and a good schedule emerges as a result of their cooperation. 3 types of multi-agent models were proposed to examine the relation between problem difficulty and multi-agent structure in terms of problem solving capability. In th preliminary experiments, the scheduling heuristics evolved by GP in the mixed agent model outperformed the other agent models and the typical well-known dispatching rules. This shows that for better performance the multi-agent scheduling model should reflect problem specific information in its structure. With the appropriate model of multi-agent structure, GP can evolve good heuristics that produce a good result in difficult scheduling problems.

In general, finding an appropriate structure of multi-agent model for solving a given problem at hand is difficult without deep knowledge of the problem. In the scheduling problem, where each resource acts as an agent, the experiment results in this paper have shown that bottleneckness is a good metric to categorize resources into different groups of agents. However it is preferable to be able to make a structure of agent groups dynamically without prior knowledge of the problem. There are some research activities tackling to this problem (Hara & Nagao 1999). In the future research, the author is going to study such automatic agent group formation in the scheduling problem and compare the results with the findings in this research.

REFERENCES

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

French, S. 1982. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* London: Ellis Horwood.

Hara, A., and Nagao, T. 1999. Emergence of cooperative behavior using ADG; Automatically Defined Groups. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1039–1046.

Haynes, T., and Sen, S. 1997. Crossover operators for evolving a team. In *Genetic Programming 1997*, 162–167.

Iba, H. 1997. Multiple-agent learning for a robot navigation task by genetic programming. In *Genetic Programming 1997*, 195–200.

Iba, H. 1998. Multi-agent reinforcement learning with genetic programming. In *Genetic Programming 1998*, 167–172.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Luke, S. 1998. Genetic programming produced competitive soccer softbot teams for robocup97. In *Genetic Programming 1998*, 214–222.

Miyashita, K., and Sycara, K. 1995a. CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence* 76(1-2):377–426.

Miyashita, K., and Sycara, K. 1995b. Improving system performance in case-based iterative optimization through knowledge filtering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 371–376.

Morton, T. E., and Pentico, D. W. 1993. *Heuristic Scheduling Systems: With Application to Production Systems and Product Management.* New York, NY: John Wiley and Sons Inc.

Shaw, M. J. 1989. A pattern-directed approach to flexible manufacturing: A framework for intelligent scheduling, learning and control. *International Journal of Flexible Manufacturing System* 2:121–144.

Smith, S. F.; Ow, P. S.; Muscettola, N.; Potvin, J.-Y.; and Matthys, D. C. 1990. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society.*

Tackett, W. A., and Carmi, A. 1994. The donut problem: Scalability and generalization in genetic programming. In Kenneth E. Kinnear, J., ed., *Advances in Genetic Programming.* Cambridge, MA: MIT Press. 143–176.

Zweben, M., and Fox, M., eds. 1994. *Intelligent Scheduling.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.