

Using Local Search to Evaluate Dispatching Rules in Dynamic Job Shop Scheduling

Rachel Hunt¹(✉), Mark Johnston¹, and Mengjie Zhang²

¹ School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, PO Box 600, Wellington, New Zealand

² School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington, New Zealand

huntrach1@myvuw.ac.nz, {mark.johnston,mengjie.zhang}@vuw.ac.nz

Abstract. Improving scheduling methods in manufacturing environments such as job shops offers the potential to increase throughput, decrease costs, and therefore increase profit. This makes scheduling an important aspect in the manufacturing industry. Job shop scheduling has been widely studied in the academic literature because of its real-world applicability and difficult nature. Dispatching rules are the most common means of scheduling in dynamic environments. We use genetic programming to search the space of potential dispatching rules. Dispatching rules are often short-sighted as they make one instantaneous decision at each decision point. We incorporate local search into the *evaluation* of dispatching rules to assess the quality of decisions made by dispatching rules and encourage the dispatching rules to make good local decisions for effective overall performance. Results show that the inclusion of local search in evaluation led to the evolution of dispatching rules which make better decisions over the local time horizon, and attain lower total weighted tardiness. The advantages of using local search as a tie-breaking mechanism are not so pronounced.

1 Introduction

Scheduling is a difficult yet important decision making process in manufacturing and service industries [19], involving allocating resources to complete a set of tasks to optimise some objective subject to constraints [7]. Job shop scheduling (JSS) problems have been extensively studied in the academic literature over the past 60 years [7, 20], due mainly to the real-world applicability (there are many thousands of job shops world-wide [13]) and the computational complexity of scheduling in such environments (JSS is known to be *NP*-hard [3]).

In JSS problems, a set of jobs must be processed through a set of machines. Each job consists of a sequence of operations which must be processed in order, where each operation has a specified machine and processing time. The aim is to schedule jobs at each machine to optimise a measure of delivery speed or customer satisfaction. In *dynamic* JSS, jobs arrive according to a stochastic process; no information is available about a job until it is present in the shop system.

This makes the scheduling task very difficult, and optimisation methods which construct complete schedules in advance are not able to be used. Dispatching rules (DRs) are mathematical functions of attributes of the jobs, machines and the entire job shop. They have been applied consistently to complex scheduling problems as they provide good solutions in real-time [13]. DRs are popular because of their low time complexity and easy interpretability as well as their ability to cope with dynamic environments. DRs cope well with dynamic environments as decisions are made as and when a machine becomes available to process a job's operation. When a machine finishes processing a job, each job currently waiting in the machine's queue is evaluated and assigned a priority value by the DR. The job with the highest priority is selected to begin processing next. One disadvantage of DRs is that they take into account the local and current conditions of the shop, without considering the future implications of decisions made. Another disadvantage of DRs is it is a time consuming (and very hard) process to construct them manually.

Genetic Programming (GP) [14] is an evolutionary computation technique which has been shown to be sufficient in representation and search capability to automatically discover DRs successfully [9, 11, 16]. The mathematical representation of dispatching rules maps naturally to tree-based GP [14]. Each GP individual represents a DR, and through evolution the fitness of the individual is assessed through discrete-event simulations of job shops in which the individual is used to dispatch jobs. There has been work to encourage the evolution of "less-myopic" DRs through the inclusion of properties from the wider shop [10], which showed that the inclusion of additional "less-myopic" terminals improved the performance of DRs on scenarios with high utilisation, and reduced the expected queue length at machines. This paper investigates "less-myopic" in a different way, by including an assessment of how well a DR makes local decisions over a longer decision horizon in its fitness evaluation.

Local search algorithms try to improve on an initial solution by searching over a defined neighbourhood of the initial solution [1]. In static JSS environments, a DR can be used to create a schedule for processing the jobs, and local search can be applied to improve the schedule. This is *not* how we are using local search in this paper. The main goal of this work is to modify a GP based system for automatic generation of dispatching rules to incorporate a local search element into the fitness evaluation stage of the GP system to provide more feedback to DRs on their scheduling performance, to encourage a more global perspective in the evolved rules. In this paper we use local search only to evaluate the fitness of DRs, *not* to change the order in which jobs are dispatched.

When a machine becomes available, a DR builds an initial schedule of jobs at the current machine. This is evaluated over an *extended decision horizon* of the current machine and the next machine of each job. Local search is used to attempt to improve that sequence, and information on the possible improvement contributes to the fitness of the DR.

We want to find DRs that make good decisions, which we can do by looking at the wider local effect when dispatch decisions are made, and investigate whether this can encourage the evolution of DRs which are less-myopic, and *better* at

scheduling jobs in an order which is *better* beyond just the first job in the queue, with *better* generalisation performance on unseen problem instances.

The remainder of this paper is organised as follows. Section 2 describes background relating to JSS and GP. Section 3 describes how local search is incorporated into our GP method for automatic discovery of DRs. Section 4 describes the experimental design and Sect. 5 discusses the results. Section 6 concludes the paper and provides directions for future work.

2 Background

Dynamic Job Shop Scheduling Problem. In this paper we are interested in the dynamic job shop. Jobs arrive according to a stochastic process, and we have no knowledge about the job until it has entered the shop system. Each job j arrives at time r_j with a sequence of N_j operations, $O_j = (\sigma_{j,1}, \dots, \sigma_{j,N_j})$, in a predefined route, and where $1 \leq N_j \leq n$. The i th operation of job j , $\sigma_{j,i}$, has processing time $p(\sigma_{j,i})$ on machine $m(\sigma_{j,i})$. Each job joins the queue at the first machine on its route, $m(\sigma_{j,1})$, upon its arrival into the shop. Each job has a due date d_j and an importance weighting w_j . The eventual completion time of a job is denoted C_j and job tardiness is then $T_j = \max\{0, C_j - d_j\}$.

A DR is used to select which of the jobs currently waiting in the queue at a particular machine will be processed next using various properties of the job, machine and shop. Once an operation σ has completed its processing time on a machine it is moved to the next machine on its route, or if the entire job is completed then it is delivered to the customer. There are many different objectives that can be optimised in JSS; we are interested in minimising the total weighted tardiness:

$$TWT = \sum_j w_j T_j = \sum_j w_j \max\{0, C_j - d_j\}. \tag{1}$$

Existing DRs. There are many DRs in the literature. No DR is known to outperform others across all objective functions and shop environments. Some of the simplest rules are shortest processing time (SPT) which schedules the job with the shortest processing time next, and its weighted counterpart, weighted shortest processing time (WSPT), which schedules the job with largest $w/p(\sigma)$. These are components of the Apparent Tardiness Cost (ATC) [21] and the weighted version of COVERT (Cost over Time) [21] dispatching rules, which have good performance for dynamic job shops with the TWT objective. ATC assigns job j the priority

$$\frac{w_j}{p(\sigma_{ji})} \exp \left(- \left[\frac{d_j - t - p(\sigma_{ji}) - \sum_{q=i+1}^{N_j} ((b \times p(\sigma_{jq})) + p(\sigma_{jq}))}{kp_{avg}} \right]^+ \right)$$

where t is the current time and p_{avg} is the average processing time of the waiting jobs at the machine. Leadtime estimation parameter b is fixed at 2.0 and

lookahead parameter k is set to 3.0 as in [21]. The wCOVERT rule assigns priority values as the expected tardiness cost per unit of imminent processing time:

$$\frac{w_j}{p(\sigma_{ji})} \left[1 - \frac{(d_j - t - \sum_{q=i}^{N_j} p(\sigma_{jq}))^+}{k \sum_{q=1}^{N_j} (b \times p(\sigma_{jq}))} \right]^+$$

where the lookahead parameter $k = 2$. The $[A]^+$ notation takes the maximum of A and 0, this means wCOVERT assigns priority 0 if the slack “exceeds some generous ‘worst case’ estimate of the waiting time” [21].

GP for Automatic Generation of Heuristic Dispatching Rules. Heuristics are methods that seek to find good quality solutions in reasonable computational time; optimality cannot be guaranteed [4]. Dispatching rules can very naturally be represented as trees, as in tree-based GP. Recent research has focused on developing new DRs using approaches such as genetic programming [4]. GP searches the space of heuristics rather than searching the solution space directly [4], as the GP trees do not represent schedules, but heuristic DRs.

GP has been used to develop DRs in a range of scheduling environments, from the simpler static two-machine flow shop [6] and job shop [9], to complex dynamic environments, e.g., in semi-conductor manufacturing [18], and varying objectives such as makespan [11,12], mean flow time [8], and total weighted tardiness [12]. Rules evolved by GP are frequently reported to compete with rules from the literature [11,16].

Local Search. Local search is one heuristic approach often used on static scheduling problems. Local search provides a “robust approach to obtain high-quality solutions to problems of a realistic size in a reasonable time” [1]. Local search methods start with an initial solution and try to find better solutions by searching neighbourhoods. Some basic neighbourhood definitions for local search that can be applied to schedules are transposition, insertion and swap [1]. Let n be the number of jobs in the queue of the machine. *Transpose* swaps two adjacent jobs in the queue (neighbourhood has size $O(n)$). *Insert* moves a job from one position to another and *Swap* swaps two jobs that can be anywhere in the queue (both have neighbourhood size $O(n^2)$). With dynamic scheduling problems, the jobs available to be scheduled are frequently changing with the arrival of jobs from outside the shop system and from other machines. This makes the application of local search difficult, as we do not know when another job will arrive, or any of its properties, therefore an optimised queue order is unlikely to still be optimal when more jobs arrive, or when we consider the final objective value function.

Instead of applying local search to find better DRs with a different order of jobs, this paper will use local search in the fitness evaluation process to improve the performance of the DRs. Details will be discussed in the next section.

Table 1. Terminal set used in GP system.

Feature	Symbol	Feature	Symbol
<i>Job Properties</i>		<i>Machine Properties</i>	
Processing time of operation	PR	Ready time of machine	RM
Remaining processing time of job	RT	Number of jobs in queue	NQ
Remaining number of operations	RO	Average wait time of last	QW
Ready time of job	RJ	Five jobs processed	
Due date of job	DD	Number of jobs in queue	NNQ
Weight of job	W	At the next machine job visits	
Next operation's processing time	NPR	Average wait time of last five jobs processed at next machine the job visits	NQW
<i>Shop Properties</i>			
Current time	CT	Average wait time of last five jobs processed across all machines	AQW

3 The New Method

Here we describe our new method, which uses local search as an additional evaluation of the fitness of a given DR, across an extended decision horizon. We use local search to evaluate potential queue orders, and compare these to the priority sorted queue. We are interested to see if the increase in computational time is a reasonable trade off for better evolved DRs.

3.1 GP System

GP individuals are DRs. The function set is $\{+, -, \times, \%, \text{if}>0, \text{max}, \text{min}\}$. All arithmetic operators take two arguments. $+$, $-$ and \times , have their usual meanings and the $\%$ operator is protected division, returning one if dividing by zero. The $\text{if}>0$ function takes three arguments; if the first argument is greater than 0 then it returns the second, else the third is returned. The max and min functions take two arguments and return the maximum and minimum of their arguments respectively. The properties of jobs, machines and the job shop that are used as terminals of the GP system are given in Table 1. The properties, NPR, NNQ and NQW all return zero if the job's current operation is its last. If fewer than five jobs have visited a machine, then WQ, NQW and AQW return the average wait time of the jobs which have visited, and if the machine has not yet processed any jobs then 0 is returned [10].

Fitness. The fitness of a DR (individual) in the current GP population is evaluated using discrete-event simulations of problem instances of a job shop. In each method the fitness value is calculated on four training problem instances, and the mean of the fitness values across these problem instances used is the fitness of the DR. In the benchmark GP method, the objective of interest to be minimised, TWT, is normalised by the expected utilisation and this is used as the fitness value. Each of our new methods use a different fitness value.

3.2 Local Search for Evaluating Dispatching Rules

We consider four different ways of using local search as a contributor to the fitness of an individual. We take the priority sorted queue of jobs waiting at the time the machine needs to make the scheduling decision as the initial solution. The neighbourhood search operator used is *SwapFront*, which swaps the job at the front job with each other job. This is used as it is a simple operator with small neighbourhood size, $(n - 1)$ for n jobs in the queue of the machine, and therefore low computational cost. The evaluation process is shown in Fig. 1.

We compare neighbourhoods by calculating the expected contribution to total weighted tardiness (TWT) of the queued jobs, which is the sum of weighted expected tardiness of each of the jobs in the queue. We calculate this by taking the expected completion time of each job given its current position in the queue at the current machine, and where it would fit in the queue of jobs at the next machine the job visits (assuming the current state of the job shop as static, with no new arrivals except those moving from the current queue). For each job j in the queue, this is calculated as

$$E(C_j) = CT + QC_j + PR_j + QN_j + NPR_j + QR_j + (RT_j - PR_j - NPR_j) \quad (2)$$

For a given job d_j , CT, PR, NPR, RT are constants as in Table 1. QC_j is the time remaining waiting (queuing) at the current machine (M_C), i.e., the sum of the processing times of jobs ahead of job j in the queue under the current ordering of jobs. QN_j is the time spent waiting (queuing) at the next machine on the job's route, (M_N). This is determined by treating the next machine's queue as a one machine problem with arrivals only from the current machine and using the DR to dispatch jobs until all jobs that join this machine from the current machine have been dispatched. This gives us a lower bound on the length of time each job from machine M_C that next visits M_N is expected to be in the queue at M_N , as in the full simulation jobs may arrive into the shop and from the other machines. QR_j is the sum of the average expected waiting times at each remaining machine on the job's route after machines C and N. If a job does not have operations remaining after the current or next machine then QN_j and QR_j will have value 0. Further we are only interested in changing the order of jobs at the first machine, and calculating the predicted time in the next queue, therefore for each job QR_j is also a constant. As our objective is to minimize the total weighted tardiness, we are seeking to minimize Eq. (3) across the neighbourhood being searched.

$$Total = \sum_{j=1}^J w_j \times \max\{0, E(C_j) - d_j\}. \quad (3)$$

Each time the DR is applied to select the next job, the expected contribution of the original queue order, $Total_0$, is calculated. If $Total_0 = 0$ then we cannot improve on the current queue ordering, so we do not apply local search. For each neighbouring solution, $Total$ is calculated, so we can find the minimum of all neighbourhoods searched, $Total_{min}$, which is used in the next stage.

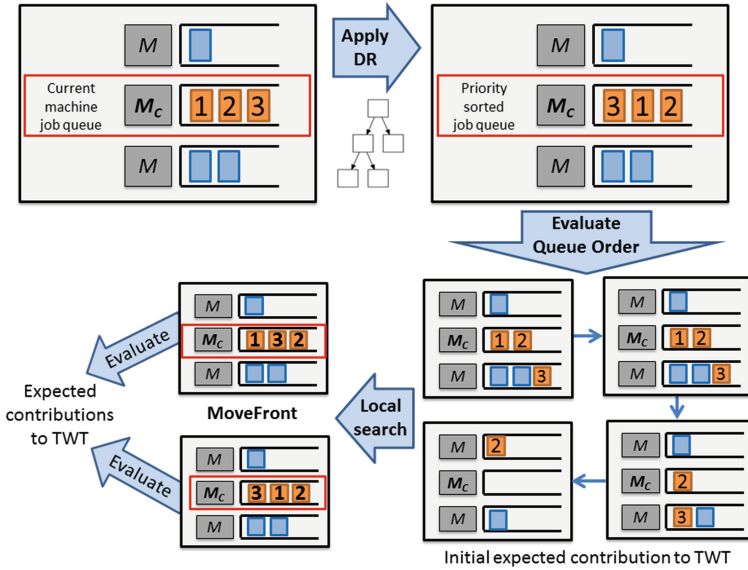


Fig. 1. Diagram of evaluation process when a machine becomes available to process a job.

Local Search to investigate Local Decision Making. When we apply local search and the new job queue order has an improved (smaller) expected contribution to TWT than the original, we calculate the difference $penalty = Total_0 - Total_{min}$. We sum all the penalties incurred during the discrete event simulation, and average over the number of times the local search was applied; this gives $penalty_{mean}$.

- *Local Search Single Objective (LS-SO).* We use local search to evaluate the job queue every twentieth time the DR is called to select a job for dispatch. The overall fitness of a DR for a problem instance is the sum of TWT and the penalty, $TWT + penalty_{mean}$.
- *Local Search Multi Objective (LS-MO).* Local search evaluates the job queue every twentieth time the DR is called to select a job for dispatch. This is a multi-objective method based on LS-SO, using the NSGA-II algorithm [5]. The penalty is used as a distinct second objective. The first objective is TWT and the second objective is $penalty_{mean}$.

Local Search to investigate Tie Breaking (TB). These methods investigate how often a DR assigns the same priority value to different jobs, and whether the default tie break, or using the SPT rule, is selecting the best job in those situations. Local search is used among the sub-schedule of jobs which have been assigned the same highest priority value in the queue. Sub-schedules are compared based on their expected contribution to the objective function. We run two discrete-event simulations for each problem instance. In the first we do not use local search, and ties are broken using SPT, this gives TWT_0 . In the second

we use local search to alter which of the jobs that are tied for top priority is dispatched, dispatching the job which leads to the lowest expected contribution to TWT, and the resulting final value of the objective function is TWT_{LS} . Due to computational restraints we use local search only every second time a tie occurs.

- *Tie Breaking Single Objective (TB-SO)*. This variant uses single objective GP, combining the TWT fitness with an added penalty if the performance of the DR was better with local search, i.e., if $TWT_{LS} < TWT_0$. The fitness of an individual for a problem instance is $TWT_0 + \max\{TWT_0 - TWT_{LS}, 0\}$.
- *Tie Breaking Multi Objective (TB-MO)*. This variant uses a multi-objective approach. The first fitness objective is TWT_0 , the second fitness objective is the penalty for possible improvement on the schedule the DR produced, $\max\{TWT_0 - TWT_{LS}, 0\}$. We use the NSGA-II algorithm [5].

We want to find out if TB-SO and TB-MO can reduce the number of ties in test cases, and if the TWT performance is improved.

4 Experiment Design

We implement a GP system using ECJ20 [15] for evolving DRs, which are represented by the GP trees.

Problem Instances. We randomly create problem instances of job shops with ten machines. We present results on one test and one extreme test scenario. The extreme testing to see how well the rules generalise to a job shop with different distribution and priority assignment. We are dealing with dynamic JSS, so jobs arrive stochastically according to a Poisson process with rate λ for all problem instances. The settings for these are shown in Table 2. The desired expected utilisation of machines (ρ) is achieved by setting $\lambda = \frac{\rho}{(\mu \times p_M)}$, where p_M is the expected number of machines a job will visit (i.e. the expected number of operations in each job). Job due dates are set by [2], $d_j = r_j + h \times \sum_{l=1}^{N_j} p(\sigma_{j,l})$, where h is a due date tightness parameter, randomly chosen with equal probability from the choices available for each job. Jobs are given weight 1, 2 or 4, with probability (0.2, 0.6, 0.2) [16]. Four training scenarios are used. The processing times at each machine follow a discrete uniform distribution with mean μ , i.e., $U(1, 2\mu - 1)$. A warm up period of 100 jobs is used, and we collect data from the next 200 jobs to arrive ($N = 200$), however new jobs keep arriving in the system until the 300th job is completed. This is a very low number of jobs due to the increase in computational time required by local search. In testing, we increase the number of jobs we collect data to $N = 1000$. In testing scenario T1, all jobs have due date tightness of 4, and utilisation is 0.95. In extreme testing, XT1, the processing times follow a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$), and utilisation is 0.95. We also change the weights given to jobs, including an additional weight for *very* important jobs; jobs are now given weight 1, 2, 4 or 8, with probability (0.2, 0.5, 0.2, 0.1). Due date tightness is equally likely from $\{2, 2.5, 3\}$; these are the same or tighter than in training.

Table 2. Simulation properties for training and testing.

		μ	ρ	h	Operations			μ	ρ	h	Operations
Training	TR1	25	0.90	{2, 3, 4}	Full	Training	TR3	25	0.90	{2,3,4}	Missing
	TR2	25	0.95	{2, 3, 4}	Full		TR4	25	0.95	{2, 3, 4}	Missing
Testing	T1	25	0.95	4	Full	Extreme testing	XT1	25	0.95	{2, 2.5, 3}	Full

GP System. The initial population is generated using the ramped-half-and-half method [14]. The population size is 50, to restrain the computational time due to the incorporation of local search, and evolution is for 50 generations, a standard setting. GP trees have a maximum depth of six. Genetic operators crossover, mutation and elitism, use rates of 85 %, 10 % and 5 % respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. This is a common setting that has been previously used [17].

5 Experimental Results

We performed 50 independent GP runs for each method, with the same seed. From these the best-of-run individual (LS-SO or TB-SO), or final non-dominated front (LS-MO or TB-MO) are tested on test and extreme test instances.

5.1 LS-SO and LS-MO

Figures 2 and 3 present the test results of the evolved DRs on T1 and XT1. These methods are compared to a benchmark method which only used the local search penalty calculation during testing. In particular the zoom-in on the overall best front of DRs in Fig. 2 shows that the LS-MO method was able to find a large number of DRs which attain lower TWT on both test instances. Four DRs form the front on this scenario. The lowest TWT attained by LS-MO and LS-SO methods are lower than the lowest attained by the benchmark.

Including local search in the evaluation process has found DRs which attain good performance. There are a large number of DRs from LS-MO and LS-SO that achieve lower penalty than the benchmark for similar TWT values, particularly for the lower TWT values. This shows that including local search has improved the local performance of DRs.

5.2 TB-SO and TB-MO

Figures 4 and 5 present the performance of TB-SO and TB-MO on two test scenarios. These methods are compared to a benchmark method which only used the tie breaking penalty calculation during testing. The TB-SO method produced the DRs which attained the lowest TWT value on both scenarios. Most of the best evolved DRs have a penalty of 0, this supports our hypothesis that classifiers which are better at separating jobs by assigning distinct priorities

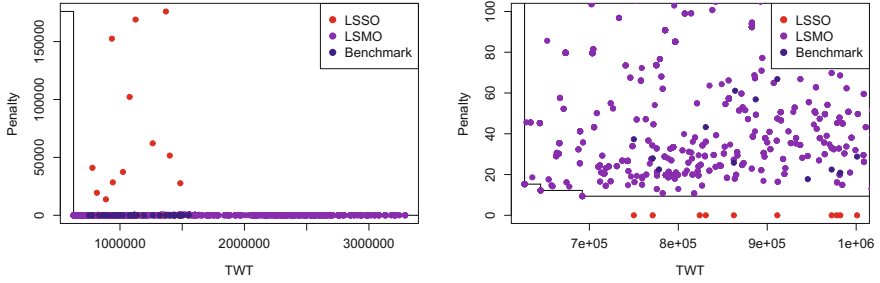


Fig. 2. Graph of the combined non-dominated front from non-dominated fronts from LS-MO and best-of-run individuals from LS-SO on scenario T1 with close up of lowest TWT attained on right.

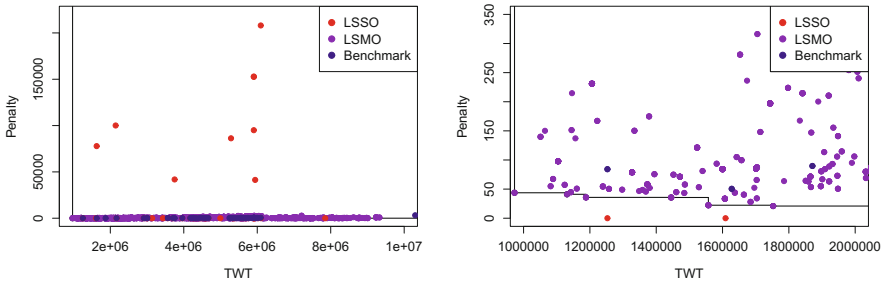


Fig. 3. Graph of the combined non-dominated front from non-dominated fronts from LS-MO and best-of-run individuals from LS-SO on scenario XT1 with close up of lowest TWT attained on right.

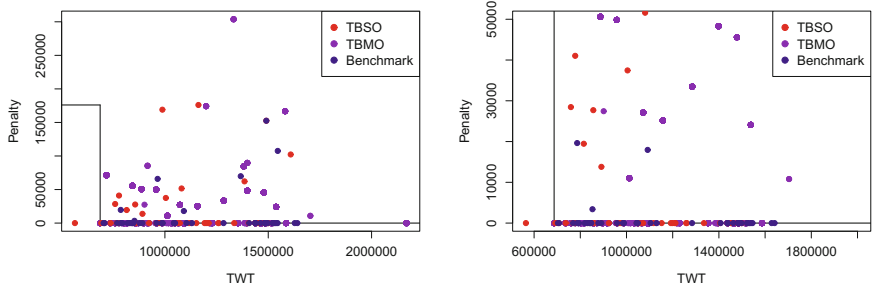


Fig. 4. Graph of the combined non-dominated front from non-dominated fronts from TB-MO and best-of-run individuals from TB-SO on scenario T1 with close up of lowest TWT attained on right.

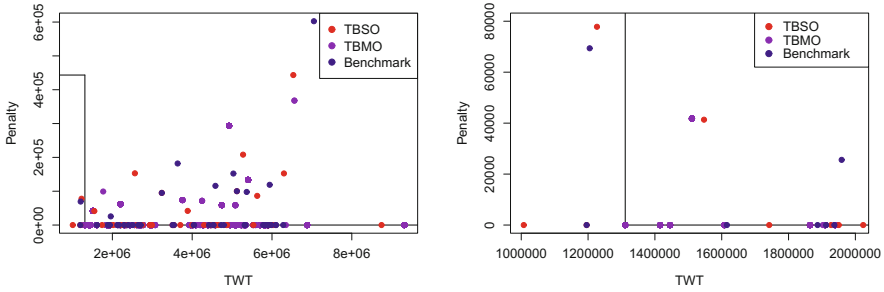


Fig. 5. Graph of the combined non-dominated front from non-dominated fronts from TB-MO and best-of-run individuals from TB-SO on scenario XT1 with close up of lowest TWT attained on right.

are more effective. The spread of TWT and penalty results on both scenarios is similar, which suggests that performing tie-breaking with local search does not offer enough improvement to justify the additional computational cost incurred.

6 Conclusions

The goal of this paper was to investigate the possible improvements to training of DRs for the dynamic ten-machine job shop in GP through the use of local search. We implemented a local search based penalty, which punished DRs which were not scheduling jobs in the best order based on the current projected contribution to the TWT objective function. Results show that the inclusion of the local search penalty in evaluation has led to the evolution of DRs which have better local performance, and achieve some better TWT values. This is worth further investigation. The inclusion of local search for tie-breaking did not offer enough improvement in TWT in its current implementation to warrant its use.

Due to the computational cost we have investigated small examples, with a small GP population and short problem instances. This has significantly reduced the number of scheduling decisions made in training, and with the shorter warm up period the system may not have reached steady state. In future we would like to increase the warm up period and reduce the frequency in which local search is applied. We will also consider other simple neighbourhood search operators, such as *Transpose* (swaps two adjacent jobs in the queue) and *MoveFront* (inserts each job at the front of the queue).

References

1. Aarts, E., Lenstra, J. (eds.): *Local Search in Combinatorial Optimization*. Wiley, Chichester (1997)
2. Baker, K.R.: Sequencing rules and due-date assignments in a job shop. *Manage. Sci.* **30**(9), 1093–1104 (1984)

3. Blazewicz, J., Domschke, W., Pesch, E.: The job shop scheduling problem: conventional and new solution techniques. *Eur. J. Oper. Res.* **93**(1), 133 (1996)
4. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Mumford, C., Jain, L. (eds.) *Computational Intelligence, Intelligent Systems Reference Library*, vol. 1, pp. 177–201. Springer, Heidelberg (2009)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
6. Geiger, C., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *J. Sched.* **9**(1), 734 (2006)
7. Hart, E., Ross, P., Corne, D.: Evolutionary scheduling: a review. *Genet. Program. Evolvable Mach.* **6**, 191–220 (2005)
8. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pp. 257–264 (2010)
9. Hunt, R., Johnston, M., Zhang, M.: Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 618–625 (2014)
10. Hunt, R., Johnston, M., Zhang, M.: Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 927–934 (2014)
11. Jakobović, D., Jelenković, L., Budin, L.: Genetic programming heuristics for multiple machine scheduling. In: Ebner, M., O’Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) *EuroGP 2007. LNCS*, vol. 4445, pp. 321–330. Springer, Heidelberg (2007)
12. Jakobović, D., Marasović, K.: Evolving priority scheduling heuristics with genetic programming. *Appl. Soft Comput.* **12**(9), 2781–2789 (2012)
13. Jones, A., Rabelo, L.C.: *Survey of Job Shop Scheduling Techniques*. Technical report, National Institute of Standards and Technology, Gaithersburg (1998)
14. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge (1992)
15. Luke, S.: *Essentials of Metaheuristics*, 2nd edn. Lulu (2013). <http://cs.gmu.edu/~sean/book/metaheuristics/>
16. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3261–3268 (2012)
17. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Trans. Evol. Comput.* **17**(5), 621–639 (2013)
18. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *Int. J. Prod. Econ.* **145**(1), 67–77 (2013)
19. Pinedo, M.L.: *Scheduling: Theory, Algorithms, and Systems*, 3rd edn. Springer, Heidelberg (2008)
20. Potts, C., Strusevich, V.: Fifty years of scheduling: a survey of milestones. *J. Oper. Res. Soc.* **60**(S1), 41–68 (2009)
21. Vepsäläinen, A., Morton, T.: Priority rules for job shops with weighted tardiness costs. *Manage. Sci.* **33**, 1035–1047 (1987)