Applied Soft Computing 63 (2018) 72–86

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling

John Park[a,*], Yi Mei[a], Su Nguyen[a,b], Gang Chen[a], Mengjie Zhang[a]

[a] Evolutionary Computation Research Group, Victoria University of Wellington, PO Box 600, Wellington, New Zealand
[b] La Trobe University, Melbourne, Australia

ARTICLE INFO

Article history:
Received 8 March 2017
Received in revised form 7 November 2017
Accepted 10 November 2017
Available online 21 November 2017

Keywords:
Combinatorial optimisation
Job shop scheduling
Genetic programming
Hyper-heuristic
Ensemble learning

ABSTRACT

Genetic programming based hyper-heuristic (GP-HH) approaches that evolve ensembles of dispatching rules have been effectively applied to dynamic job shop scheduling (JSS) problems. Ensemble GP-HH approaches have been shown to be more robust than existing GP-HH approaches that evolve single dispatching rules for dynamic JSS problems. For ensemble learning in classification, the design of how the members of the ensembles interact with each other, e.g., through various combination schemes, is important for developing effective ensembles for specific problems. In this paper, we investigate and carry out systematic analysis for four popular combination schemes. They are majority voting, which has been applied to dynamic JSS, followed by linear combination, weighted majority voting and weighted linear combination, which have not been applied to dynamic JSS. In addition, we propose several measures for analysing the decision making process in the ensembles evolved by GP. The results show that linear combination is generally better for the dynamic JSS problem than the other combination schemes investigated. In addition, the different combination schemes result in significantly different interactions between the members of the ensembles. Finally, the analysis based on the measures shows that the behaviours of the evolved ensembles are significantly affected by the combination schemes. Weighted majority voting has bias towards single members of the ensembles.

1. Introduction

Job shop scheduling (JSS) problems are types of combinatorial optimisation problems that model manufacturing environments [1]. In a JSS problem, there is a shop floor with machines that are used to process arriving jobs. Although both academics and industry experts have interest in JSS problems, there has always been a gap between the classical research to JSS (from an academic perspective) and its application (from an industrial perspective) [2]. In classical research to JSS, many approaches handle *static* JSS problems, where the properties of the shop are known a priori [3]. However, in practice the properties of the shop are extremely variable and it is commonly believed that any change to the shop floor can cause ripple effects [2]. To bridge the gaps between the static JSS problems that have been handled by academics (where the problems are predictable and can be optimised in advance) and unpredictable real-world scenarios encountered in the industry, researchers have focused on matching the problem more closely with real-world manufacturing environments by incorporating unforeseen events into the problem [4]. JSS problems that have real-time unforeseen events that affect the properties of jobs, machines and shop floor are called *dynamic* JSS problems [4]. Examples of unforeseen events include dynamic job arrivals, where job arrivals are unknown until they reach the shop floor, and machine breakdowns [4–6]. In real-world manufacturing environments, it is likely that last minute (and potentially urgent) jobs can arrive that require attention [4,5]. In general, dynamic JSS problems are much more difficult than static JSS problems, and conventional optimisation methods cannot solve dynamic JSS problems due to the unpredictable changes in the shop floor [7]. Instead, *dispatching rules* [3] are studied by both academics and industry experts for dynamic JSS problems due to their interpretability [6], short reactions times and their ability to cope well with the unforeseen events in dynamic JSS problems [8]. To automate the design of effective dispatching rules, many genetic programming based hyper-heuristic (GP-HH)

* Corresponding author.
E-mail addresses: John.Park@ecs.vuw.ac.nz (J. Park), Yi.Mei@ecs.vuw.ac.nz (Y. Mei), Su.Nguyen@ecs.vuw.ac.nz (S. Nguyen), Aaron.Chen@ecs.vuw.ac.nz (G. Chen), Mengjie.Zhang@ecs.vuw.ac.nz (M. Zhang).

https://doi.org/10.1016/j.asoc.2017.11.020
1568-4946/

approaches to dynamic JSS problems have been proposed in the literature [6]. GP-HH approaches have successfully evolved dispatching rules for various dynamic JSS problems which are more effective than the man-made counterparts [6].

In addition to the primary motivation of automatically generating effective dispatching rules for dynamic JSS problems [6], many GP-HH approaches have focused on evolving *robust* dispatching rules for the dynamic JSS problems [9], i.e., rules that function reliably and effectively despite noise and unexpected changes in the problem domain. However, many approaches focus on evolving dispatching rules with a single constituent component, and are often not sufficiently robust for dynamic JSS problems. This issue was addressed by evolving *ensembles* [10] of dispatching rules [11–13]. Ensemble learning has been shown to be effective at training robust high quality rules for JSS problems [11–13] and problems outside of JSS (e.g. classification problems [10]) because ensemble members are able to minimise errors made by other ensemble members [10]. This makes ensemble approaches a promising direction to improve the robustness of rules evolved by GP-HH for dynamic JSS problems. However, the existing ensemble GP-HH approaches to JSS [11–13] only use majority voting combination scheme [10] to combine the outputs of the subcomponents of the ensembles together. Design of interactions between the ensemble members is an important factor in ensemble learning [10]. It is clearly evidenced on some classification problems that different combination schemes, such as linear combination and weighted combination schemes, can be more effective than majority voting [10]. Therefore, it may be possible that better rules can be evolved by GP using combination schemes besides majority voting. In addition, by analysing the rules evolved by the different combination schemes, one can observe the behaviour of ensembles that are applied to dynamic JSS problem instances. This allows for future ensemble GP-HH approaches which may generate higher quality and more robust rules than the current state-of-the-art GP-HH approaches for dynamic JSS problems.

1.1. Goal

For the scope of this paper, the analysed combination schemes are majority voting, weighted majority voting, linear combination and weighted linear combination [10]. The goal of this paper is to investigate and analyse the combination schemes to further improve the robustness of ensemble GP-HH approaches for dynamic JSS problems. Majority voting was previously used by the existing ensemble GP-HH approaches to dynamic JSS problems [11–13]. However, linear combination, weighted majority voting and weighted linear combination, which have extensively been used in the classification literature [10], have not been explored in any existing research on dynamic JSS. The members of the evolved ensembles are analysed by their behaviours and interactions on complex decision situations [14]. For this paper, we propose new analysis measures to compare specific behaviours between the evolved ensembles from the different combination schemes: the diversity in the decisions made by ensemble members, the bias towards specific ensemble members, and how the different members of the ensembles rank in the ensembles. Overall, this investigation into the combination schemes for GP-HH to dynamic JSS problems is broken down into the objectives given below.

(a) Extend an existing ensemble GP-HH approach by incorporating the four different combination schemes.
(b) Investigate the performances of evolved ensembles against each other and the state-of-the-art results [8,15].
(c) Analyse in detail the behaviour of the evolved ensembles against complex decision situations from different perspectives.

1.2. Organisation

The organisation of the paper is as follows. Section 2 includes a background into dynamic JSS and related work which provide approaches to dynamic JSS problems. This section also includes a description of ensemble learning proposed in the literature. Section 3 describes the combination schemes investigated and modifications made to an existing ensemble GP approach to incorporate the combination schemes. Afterwards, Section 4 provides a description of the analysis procedure to measure how the rules behave in the problems. Section 5 covers the experimental design, Section 6 covers the evaluation procedure, the experimental results, the discussions of the results and the analysis. Section 7 gives the conclusions and the future works.

2. Background

This section gives the dynamic JSS problem definition investigated in this paper, and approaches in the literature for tackling dynamic JSS problems, including GP-HH approaches. It also gives a brief description of ensemble learning and their applications to JSS problems.

2.1. Problem definitions for dynamic JSS

In a dynamic JSS problem instance, there are M machines on the shop floor. An arriving job j has a sequence of N_j operations denoted as $\sigma_{1j}, \ldots, \sigma_{N_j j}$. The ith operation of job j, denoted as σ_{ij}, needs to be processed at machine $m(\sigma_{ij})$. The operations need to be processed in the order of their indices, e.g., operation σ_{2j} cannot start until operation σ_{1j} has been processed and completed on machine $m(\sigma_{1j})$. The duration of time operation σ_{ij} is processed on the machine is called the *processing time* [3], and is denoted as $p(\sigma_{ij})$ (abbreviated to p_{ij}). In addition, a machine can only process one operation at a time. There is no re-entry (job has two or more operations on the same machine) and preemption (a job's operation being processed on a machine can be interrupted) [3]. The time when job j arrives at the relevant machine that the job's operation is currently up to is called the *operation ready time* [3], and is denoted as $r(\sigma_{ij})$ (abbreviated to r_{ij}). The time when the job j arrives on the shop floor is the operation ready time of the first operation of job j (denoted as r_j), and is called the *release time* of job [3]. The goal of JSS is to complete all arriving jobs by processing the operations of the jobs on the machines. The sequence of times and jobs that are processed on the machines is called a *schedule*, and the goal is to generate a schedule that is optimal given an *objective function* [3]. For this paper, we focus on the objective of minimising the mean tardiness (MT) of the schedule. In a problem instance with a tardiness related objective, a job j also has a due date d_j. The time when all of a job's operations have been completed is called the job *completion time*. If a job j's completion time C_j is greater than its due date d_j, then the job is considered *tardy* and has a tardiness $T_j = C_j - d_j$. Otherwise, a job completed before its due date has a tardiness value of zero, i.e., $T_j = 0$ if $C_j \leq d_j$. Afterwards, the mean tardiness of a schedule which has processed N jobs arriving on the shop floor is given by $\frac{1}{N}\sum_{j=1}^{N} T_j$, i.e., is the average tardiness over the N jobs completed in the schedule. JSS problems with tardiness related objective has been extensively investigated in the literature as they are strongly NP-hard [16]. In addition, we focus on dynamic JSS problems with dynamic job arrivals. This means that a job j's properties such as its operations and due date are unknown until it arrives on the shop floor at time r_j. In other words, a scheduling algorithm has limited