A Novel Heuristic Generator for JSSP Using a Tree-Based Representation of Dispatching Rules

Kevin Sim
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
k.sim@napier.ac.uk

Emma Hart
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
e.hart@napier.ac.uk

ABSTRACT

A previously described hyper-heuristic framework named NELLI is adapted for the classic Job Shop Scheduling Problem (JSSP) and used to find ensembles of reusable heuristics that cooperate to cover the heuristic search space. A new heuristic generator is incorporated that creates novel heuristics, formulated as GP-like tree structures, by combining problem specific information formulated as a large set of terminal nodes. The new heuristics operate as dynamic dispatching rules, selecting at each iteration the highest priority operation available for scheduling. The new system is trained and tested on a large set of 1400 newly generated problem instances, using both makespan and weighted tardiness as fitness metrics. Results on unseen test instances show that relatively small ensembles of evolved heuristics significantly outperform any individual *one size fits all* heuristic and a greedy selection from a large set of existing rules.

Categories and Subject Descriptors

Computing methodologies [**Machine learning**]: Machine learning algorithms

Keywords

Hyper-heuristics; Artificial Immune Systems

1. INTRODUCTION

Amongst the most successful approaches for solving the widely studied Job Shop Scheduling Problem (JSSP) are metaheuristics that provide excellent results when tailored to the problem instances that they are used to solve. However such approaches are often infeasible in real world applications due to their cost in terms of computational effort and the need to adapt the techniques to the constraints imposed by different job-shop environments. It is therefore often the case that simple dispatching rules are used to solve real world scheduling problems due to their simplicity, understandability, and speed of execution. Recent approaches using Genetic Programming techniques to automate the design of dispatching rules have shown that automated approaches can outperform human designed equivalents [3]

We adapt a previously developed hyper-heuristic framework dubbed NELLI, described fully in [2], that has been used to generate ensembles of simple heuristics that collectively exhibit good performance across very large problem sets in the domains of bin packing and job shop scheduling. One of the core elements of NELLI is a *heuristic* generator that in NELLI's previous application to the JSSP [1] created new ''heuristics' by combining existing, well-known dispatching rules into variable length sequences that were iteratively applied to a problem instance until all operations were scheduled. Although published results improve on the performance of the component rules, performance is limited by the size and diversity of the fixed set of hand crafted dispatching rules. Two improvements to the heuristic generator are presented that increase the number and diversity of available rules that are sequenced by NELLI into heuristics. Firstly, the set of pre-defined rules is increased in size to incorporate additional and more dynamic information about a schedule. Secondly, by using a tree-based rule representation, typical to that used in Genetic Programming, we are able to generate completely novel composite dispatching rules. The new system is trained and tested on a large set of JSSP generated as described in the following section.

2. PROBLEM DESCRIPTION

Two variations of JSSP are investigated, denoted using the conventional $\alpha|\beta|\gamma$ notation as $Jm||C_{max}$ and $Jm||\sum\omega_i T_i$ where the term J_m translates as a Job Shop environment and the last term corresponds to the objective (Makespan (C_{max}) and Summed Weighted Tardiness (SWT) respectively). Due to the lack of any widely used *large* sets of benchmark problems, two new sets of problems are generated comprising of 500 and 200 instances that are solved separately using the two different objective fitness functions effectively making 1400 problems in total. 1000 are used during training and the remaining 400 are used to evaluate the evolved heuristics. Problems are generated using all combinations of machines m and jobs n where $m \in [5, 10, 15, 20, 25]$ and $n \in [10, 25, 50, 100]$. For each of the 20 parameter combinations 25 problems are generated for the training set and 10 for the test set. Each problem is considered using both fitness metrics, doubling the number of training and test instances. The processing time for an operation is selected randomly from a uniform distribution using $p_{i,j,k} = U[m/2, m2]$ Release dates are drawn randomly

GECCO '15 July 11-15, 2015, Madrid, Spain
© 2015 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764697

from one of two distributions depending on the number of jobs in the problem instance. For instances with less than 50 jobs $r_i = U[0, 20]$ and for those with 50 or more $r_i = U[0, 40]$ Due dates and job weights, used by the SWT objective, are added using $d_i = r_i + c \times \sum_{j=1}^{n} p_{ij}$ with c is fixed at 1.3. and by the 4:2:1 rule respectively. The later is informed by research suggesting that 20% of a company's customers are the most important (weight 4), 60% of medium importance(2) and 20% less important (1).

This process results in 1400 problems containing a total of 64750 jobs and 971250 operations.

3. HEURISTIC GENERATOR

The main focus of the paper is to improve the diversity of the dispatching rules (DR) that are combined into heuristics (linear sequences of rules) and used by NELLI. We increase the set of 13 DRs used previously to 28 and implement these as Terminal Nodes that can be further combined into GP-like tree structure enabling substantially greater numbers of novel composite rules to be generated. The 28 rules, not described here due to space limitations, include simple static and dynamic DR taken from the literature that return information about an operation, the associated job or the associated machine. Also included are a number of composite rules such as Job Apparent Tardiness Cost (JATC) which encapsulates features of both the weighted shortest processing time and minimum slack rules and has been shown to outperform its component parts [4] for the SWT objective.

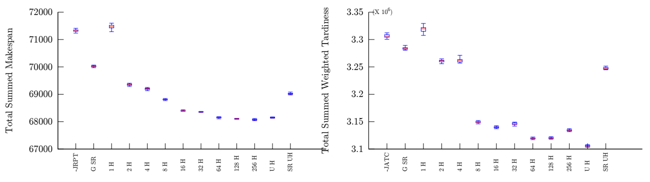
The function set includes the mathematical operators +,-,X,/, abs,max,exp and one 3 operand conditional operator which evaluates the second operand if the first is less than or equal to zero otherwise the third operand is evaluated.

Each of the 28 terminal nodes may be used in isolation or may be combined into a tree structure along with nodes from the function set and used as a DR. Using NELLI we generate sets of heuristics (strings of rules) that cooperate to cover the training problem sets. Rules are initialised using the standard GP ramped half and half method and combined into sequences by NELLI as described in[2, 1]. The rules that make up the heuristics sustained by NELLI act as the GP population. Each iteration a new heuristic is generated and added to the system by either randomly initialisation or by mutating an existing heuristic. Where a heuristic is mutated (rules added, mutated or removed) existing rules may undergo mutation. Mutation of an existing rule is conducted using a replace operator that replaces a random branch with a newly initialised sub-tree. The system is evaluated using the following empirical investigation.

4. EXPERIMENTS AND RESULTS

For each of the 2 objectives considered the maximum number of heuristics allowed in the system was limited to $1, 2, 4, 8, 16, 32, 64, 128, 256, \textit{Unrestricted}$ NELLI is run 30 times on the corresponding 500 training problems for each scenario for a total of 5000 iterations. In addition, each individual Terminal node was executed 30 times on each of the 1400 problems in order to allow a comparison between evolved rules and their component parts. Figures 1a and 1b show results on the unseen test problems for C_{max} and SWT respectively. The first box on each plot shows the single Terminal Node (DR) that performed best on the corresponding test problems. The next box labelled GSR shows the total summed objective (C_{max} or SWT) achieved following a greedy selection of the best component DR for each individual problem instance. These 2 boxes allow a comparison between evolved rules and the component rules from which they are created.

The plots labelled $1 \ldots 256H$ show how NELLI performs when the number of heuristics allowed in the system is restricted. The plots labelled UH and SR UH allow a comparison between heuristics composed of evolved DR and those composed of only single Terminal Node DRs respectively for the case where the number of heuristics is unrestricted.



(a) Makespan test problems (b) Tardiness test problems

Figure 1: Results on the unseen test problems using C_{max} and SWT as objectives

5. CONCLUSIONS AND FUTURE WORK

The paper has described a heuristic generator that uses a tree based representation in conjunction with a large set of terminal nodes to evolve new dispatching rules that are combined using NELLI into ensembles of linear sequences of rules defined as heuristics. Results on 1400 new problems, from the JSSP domain, using two objective functions show that the new system outperforms dispatching rules from the literature, and improves upon the previous application that created heuristics composed only of sequences of existing dispatching rules. The reusable heuristic ensembles generated can be used to provide quick and high-quality solutions to unseen problems or to provide a set of diverse solutions as seeds for metaheuristic approaches. We show that by combining relatively *weak* heuristics into an ensemble the system performance is greater than the sum of its parts.

6. REFERENCES

[1] K. Sim and E. Hart. An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In *GECCO '14: Proceeding of the sixteenth annual conference on Genetic and evolutionary computation conference*, 2014.

[2] K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation Journal*, In Press, January 2014.

[3] J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

[4] A. P. J. Vepsalainen and T. E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047, 1987.