Applied Soft Computing 12 (2012) 2781–2789

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Evolving priority scheduling heuristics with genetic programming

Domagoj Jakobović*, Kristina Marasović

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia

ARTICLE INFO

Article history:
Received 5 July 2011
Received in revised form 13 January 2012
Accepted 10 March 2012
Available online 4 May 2012

Keywords:
Genetic programming
Priority scheduling
Scheduling heuristics

ABSTRACT

This paper investigates the use of genetic programming in automated synthesis of scheduling heuristics for an arbitrary performance measure. Genetic programming is used to evolve the priority function, which determines the priority values of certain system elements (jobs, machines). The priority function is used within an appropriate meta-algorithm for a given environment, which forms the priority scheduling heuristic. The evolved solutions are compared with existing scheduling heuristics and found to perform similarly to or better than existing algorithms. We intend to show that this approach is particularly useful for combinations of scheduling environments and performance measures for which no adequate scheduling algorithms exist.

1. Introduction

Scheduling is a decision-making process concerned with allocating scarce resources to tasks over given time periods, aiming to optimize one or more objectives (performance measures) [1]. Scheduling problems are usually modeled based on a quantitative mathematical approach. The *scheduling environment* includes the *resource set* (the amount and the type of each resource), and the *nature of the tasks*. A model may contain one or more machines available in unit amounts or in parallel. Tasks may be described by their resource requirement, duration, and the start and due times. Precedence restrictions that may exist among the tasks and other processing characteristics and constraints are also considered as parts of task modeling. The manufacturing vocabulary is usually employed, as early developments in the scheduling field were related to the problems arising in manufacturing. Hence, resources are usually called machines and tasks are referred to as jobs.

Decision-making goals are modeled as objective functions. The objective to be minimized is a function of the completion times for the jobs. For example, the objective could be to minimize the completion time of the last job or the number of jobs completed after their respective due dates.

A solution to a scheduling problem includes resolving allocation and sequencing issues: which resource should be allocated to perform each task, and when each task should be performed. There are cases when scheduling is narrowed down to pure allocation resolution, and in other cases scheduling is purely sequencing. Depending on the model, the solution usually involves developing, applying and evaluating combinatorial methods, simulation techniques and heuristic approaches.

The combinatorial nature of most scheduling problems allows using *search-based* and *enumerative* techniques [2], including genetic algorithms, branch and bound, simulated annealing. These methods usually offer good quality solutions, but at the cost of a large amount of computational time required by a solution algorithm. Furthermore, search-based techniques are not applicable in dynamic or uncertain conditions, which may require frequent schedule modification or reaction to changing system requirements (i.e. resource failures or job parameter changes). Scheduling with simple but fast heuristic algorithms that directly build schedules (not searching the solution space) is therefore highly effective, and the only feasible solution, in many instances.

Due to the inherent problem complexity and variability, many scheduling systems employ such heuristic scheduling methods. Among the available heuristic algorithms, the question arises of which heuristic to use in a particular environment, given different performance criteria and user requirements. The problem of selecting the appropriate scheduling policy is an active research area [2,3], and considerable effort is needed to choose or develop the algorithm best suited to the scheduling problem at hand. Recent approaches addressing this issue include inferring fuzzy rules to assign the appropriate scheduling strategy for parallel jobs on identical machines [4] and building a fuzzy rule base for a hypothetical manufacturing system [5]. Considering the vast variety of environments that may be encountered, an effective solution to this problem may be provided using machine learning, particularly genetic programming, to create *problem-specific* scheduling algorithms.

* Corresponding author. Tel.: +385 1 6129967.
E-mail addresses: domagoj.jakobovic@fer.hr (D. Jakobović), kristina.marasovic@gmail.com (K. Marasović).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.03.065

Genetic programming (GP) is rarely employed in scheduling [6,7], mainly because it is unpractical to use it to search the space of potential solutions (i.e. schedules). It is, however, suitable for searching the space of *algorithms* that provide solution to the scheduling problem. Previous work in this area includes evolving scheduling policies for the single machine unweighted tardiness problem [8–10] and single machine scheduling subject to breakdowns [11]. Classic job shop tardiness scheduling is also addressed [12,13], as well as airplane scheduling in air traffic control [14,15]. In most cases, the authors observe performance comparable to human-made algorithms. The scheduling procedure is however defined only implicitly for a given scheduling environment, which leaves space for misunderstandings and affects reproducibility. Moreover, the scheduling paradigm is mostly reduced to list scheduling, where job ordering is determined only at the beginning of the process, which reduces the space of usable heuristics.

In this paper, we structure the scheduling algorithm in two components: a *meta-algorithm*, which uses priority values to perform scheduling, and a *priority function*, which defines priorities for different system elements. The priority function is evolved for a given scheduling environment using genetic programming, while the meta-algorithm is defined manually. This allows creating of various heuristics tailored to an arbitrary scheduling environment.

To illustrate this methodology, we address the problems in single machine and job shop scheduling environments, combined with several real-world properties that complicate the application of existing heuristics. The properties include job weights, dynamic job arrivals, precedence constraints, sequence dependent setup times and combinations of those. Other problems (not covered here) were also investigated, such as multiple proportional machine scheduling [16], dynamic identification of bottleneck resources [17] and resource constrained project scheduling [18].

The remainder of this paper is organized as follows: Section 2 describes the approach in more detail, and Sections 3 and 4 show its application on the single machine and job shop problems, respectively. Section 5 covers the choice of relevant parameters. Section 6 discusses the results, followed by a brief conclusion.

2. Priority scheduling with genetic programming

A natural representation of the solution to a scheduling problem is a sequence of jobs to be performed on each machine. This sequence presents only a solution to the specific *problem instance*, i.e. a given set of jobs that need to be scheduled. For a different set of jobs (different initial conditions), a new solution must be found. With genetic programming, we can produce an *algorithm* that will be used to generate schedules for all possible problem instances in a scheduling environment. Algorithms evolved by genetic programming are usually in the form of a *tree*, where tree nodes represent problem specific functions, variables (terminals) or commands.

The scheduling paradigm applied in this work is priority scheduling, in which certain elements of the scheduling system (i.e. jobs, machines) are assigned priority values. The choice of the next job run on a certain machine is based on job's priority value, which may be determined dynamically. This type of scheduling algorithm is also called, variously, 'dispatching rule', 'scheduling rule' or just 'heuristic'. The term *scheduling rule*, in a narrow sense, often represents only the priority function that assigns values to system elements (jobs in most cases). For instance, a scheduling process may be described with the statement 'scheduling is performed using the *shortest processing time* (SPT) rule', where jobs are sequenced in increasing order of processing times.

While the method of assigning jobs to machines based on priority values is trivial in most cases, in some environments it is not. This is particularly true in dynamic conditions, where jobs arrive over time or may not be run before another job finishes. A *meta-algorithm* must therefore be defined for each scheduling environment, dictating how jobs are scheduled based on their priorities and possible system constraints. The meta-algorithm encapsulates the priority function, but the same meta-algorithm may be used with different priority functions and vice versa. In virtually all related literature, the meta-algorithm part is never explicitly expressed but implicitly presumed – only the priority function is given, without defining when the priority is calculated, which jobs are taken into account, etc. This can lead to different results and misunderstandings between projects.

In this work, the meta-algorithm is defined manually for a specific scheduling environment, such as single machine or job shop. The priority function, on the other hand, is evolved with genetic programming and represented as a tree structure using appropriate functional and data structures. This way, using the same meta-algorithm, different scheduling algorithms best suited for various criteria can be devised.

A simple example may be given for a single machine environment with static job availability (all jobs ready to begin at time zero), where the meta-algorithm is trivial:

while there are unscheduled jobs **do**
  wait until machine is ready;
  calculate priorities π_j of all unscheduled jobs;
  schedule job with highest priority;
end while

The task of genetic programming is to find a priority function that yields the best results considering given user requirements. In the evolution process, the candidate priority functions – GP individuals – are used within the meta-algorithm to generate schedules for the learning set of scheduling instances. Once evolved, the priority function can be used with an existing meta-algorithm to generate schedules for unseen problem instances. The described scheduling algorithm structure allows modular development and the possibility of iterative refinement, which is particularly suitable for machine learning methods.

The priority function depends on the given performance measure; for instance, the total weighted flowtime (the amount of time all the jobs spend in the system) is minimized by the weighted shortest processing time (WSPT) rule which is defined with the following priority function: $\pi_j = w_j/p_j$, where w_j is relative weight and p_j is the processing time of job j. In this work, the actual priority functions are evolved using GP, given some objective as a fitness measure.

The time complexity of priority scheduling algorithms depends on the meta-algorithm, but it is in most cases negligible compared to search-based techniques, which allows using this method for on-line scheduling [19] and dynamic conditions. For instance, a common meta-algorithm simply finds the best priority value among all available jobs, which takes $O(n)$ time. It is obvious that the priority function must be previously evolved with genetic programming, which we assume is always performed offline, before the actual scheduling occurs. The evolution process itself takes several hours on average and does not vary greatly for the scheduling problems included in this work. All heuristics presented in this paper, both the existing and evolved ones, produce schedules for several hundred problem instances in less than a second.

3. Single machine scheduling

3.1. Problem statement

In a single machine environment, n jobs are processed on a single resource. In a *static* problem, each job is available at time zero, whereas in a *dynamic* problem, each job j has a release date r_j. The processing time of the job is p_j and its due date is d_j. The relative