Repository Istituzionale dei Prodotti della Ricerca del Politecnico di Bari

Speeding up a Rollout algorithm for complex parallel machine scheduling

This is a pre-print of the following article

Original Citation:
Speeding up a Rollout algorithm for complex parallel machine scheduling / Ciavotta, Michele; Meloni, Carlo; Pranzo, Marco. - In: INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH. - ISSN 0020-7543. - 54:16(2016), pp. 4993-5009. [10.1080/00207543.2016.1157276]

Availability:
This version is available at http://hdl.handle.net/11589/89887 since: 2022-06-07

Published version
DOI:10.1080/00207543.2016.1157276

Publisher:

Terms of use:

(Article begins on next page)

12 October 2024

To appear in the *International Journal of Production Research*
Vol. 00, No. 00, 00 Month 20XX, 1–18

RESEARCH ARTICLE

Speeding up a Rollout Algorithm for Complex Parallel Machine Scheduling

Michele Ciavotta[a], Carlo Meloni[b,d] and Marco Pranzo[c]

[a]*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy*;
[b]*Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Bari, Italy*
[c]*Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena, Siena, Italy*
[d]*Istituto per le Applicazioni del Calcolo 'M. Picone', Consiglio Nazionale delle Ricerche, Bari, Italy*

(*July 17, 2015*)

The practice of operations management calls to tackle different scheduling problems. Often the environment to consider can be modeled with a set of parallel resources to be scheduled and is characterized by one or more application-specific constraints or objectives. To tackle these problems, this paper addresses the use of multi-heuristic Rollout algorithmic procedures. Main characteristics of the considered methodology are its modularity, the adaptability to different objectives and constraints, and the easiness of implementation. Moreover, the multi-heuristic Rollout is able to easily incorporate human experience inside its research patterns to fulfill complex scheduling requirements as those of interest in services and manufacturing applications. A drawback is often represented by the required computation time. This paper proposes some alternatives of the full multi-heuristic Rollout algorithm aimed at improving the efficiency by reducing the computational effort while preserving the effectiveness. Namely, we propose Dynamic Heuristics Pruning and Candidates Reduction strategies. As illustrative case studies, we analyze deterministic complex parallel machine scheduling problems showing how Rollout procedures can be used to tackle several additional constraints arising in real contexts. An extensive campaign of computational experiments shows the behavior of the multi-heuristic Rollout approach and the effectiveness of the different proposed speed-up methods.

Keywords: job scheduling, parallel machines, Rollout algorithm, Pilot method

1. Introduction

Nowadays, the stiff international crisis embitters the competition and enforces to continuously improve and innovate the production techniques in terms of both machinery and methodologies. Therefore, production managers are ceaselessly challenged to enhance planning methods and scheduling activities striving for better utilization of available resources in order to improve the reactiveness and reduce production costs. Producers need, indeed, to increase responsiveness and flexibility and, at the same time, achieve a reduction of manufacturing costs preserving the overall Quality of Service (QoS). In this context, the automation of planning and scheduling activities allows the management to focus on critical aspects of the manufacturing processes and to contribute to a plant-wide responsiveness increase. In this paper we move in this direction introducing a general framework to address scheduling problems arising in manufacturing production systems. Specifically, we consider a metaheuristic strategy often referred to as *Rollout* method (Bertsekas et al. 1997) or *Pilot* method (Duin and Voß 1999). This approach, which allows generating high quality solutions for a class of discrete optimization problems, has been independently developed by Bertsekas et al. (1997) and Duin and Voß (1994).

The rationale of this technique (henceforth named Rollout) is to exploit the good behavior of one or more fast pilot heuristics by embedding them into a general constructive framework in order to achieve near-optimal results in short times. The main merit of this methodology consists of being easily adaptable to handle different constraint families and objectives since it is suitable for

incorporating practitioners' experience into a computerized logic.

The aim of this work is threefold; first we demonstrate that Rollout is easy to implement and results effective to solve even very complex scheduling problems. Then we present a real-life hard-constrained case study where it has been satisfactory applied. We face a complex parallel machine environment that includes batch production with family setups, due dates, deadlines and other realistic constraints, for more details see Ciavotta et al. (2009).

To tackle this problem a Rollout algorithm encompassing multiple pilot heuristics at once has been developed (i.e., a multi-heuristic Rollout) and it is, to the best of our knowledge, the first attempt in the deterministic combinatorial optimization literature to realize such a complex scheme.

Finally, we propose several speed-up techniques (such as a dynamical heuristic pruning and state reduction mechanisms) to apply on the classical Rollout scheme in order to reduce computation times while preserving the quality of the solutions. We assess the effectiveness of the Rollout scheduling framework by showing the improvements with respect to the basic Rollout behaviour, and the fast heuristics that we expressly developed for this problem.

The paper is organized as follows. In Section 2 the proposed algorithmic approaches are described. The case studies are illustrated in Section 3 whereas computational experiments are described and discussed in Section 4. Finally, conclusions are drawn in Section 5.

2. The algorithmic approach

In this section we describe a general approach to designing combinatorial algorithms that is easily adaptable, modular and suitable for incorporating human experience in reusable and robust software components for heuristic-driven search. As solution technique we use a metaheuristic strategy known as Rollout method (Bertsekas et al. 1997) or Pilot method (Duin and Voß 1999). Notwithstanding the fact that they started from different considerations (dynamic programming for Bertsekas et al. (1997) and discrete branch-and-bound for Duin and Voß (1999)) the authors ended up developing essentially the same algorithm, which tries to overcome or, at least, mitigate the limits of a heuristic procedure by means of a look-ahead strategy (Voß and Duin 2003). The Rollout scheme usually does not exploit strong problem-related properties, but let the search process to be heuristically guided by one or more basic or pilot procedures. These underlying procedures may, however, employ some problem specific properties or may incorporate rules or policies based on the decision-maker expertise. This is especially helpful when dealing with problems and models that suffer for a lack of known properties to be exploited to design an ad-hoc solution method.

The literature reports several applications of this algorithmic approach including bus driver scheduling problems (D'Annibale et al. 2007), Steiner tree problems (Duin and Voß 1999, 1994), sequencing problems with stochastic routing (Secomandi 2003), sequential fault diagnosis problems (Tu and Pattipati 2003), job shop scheduling problems (Meloni et al. 2004), scheduling in pharmaceutical industry (Pacciarelli et al. 2011; Ciavotta et al. 2009), and variants of spanning tree problems (Golden et al. 2008). Nonetheless, similar concepts and ideas have been proposed over the years (e.g. see Néron et al. (2008); Glover, and Taillard (1993); Amberg et al. (1999)), and look-ahead features may also be found in connection with various optimization schemes. A survey on the Rollout methods for deterministic optimization applications is given by Bertsekas (2013).

2.1 *The Rollout algorithm*

Behind the Rollout algorithm there is the idea of iteratively splitting the search space into progressively smaller search spaces and analyze each one of them by means of some heuristic procedures. The most promising sub-space is then selected and the process is repeated until a near-optimal solution is generated. We next illustrate more in detail the general scheme of this method.

Given an optimization problem P, let Σ be the finite set of its feasible solutions, solving P means finding a solution $\varsigma \in \Sigma$ that minimizes a given objective function Φ. In order to apply the Rollout

scheme to P a solution ς must be representable as a collection of m components or decision variables, i.e. $\varsigma = (c_1, c_2, \ldots, c_{m-1}, c_m)$. A partial solution in which the value of only k components has been assigned is called k-state and referred to as s_k. The 0-state is a dummy state, corresponding to the situation in which no component has been assigned. From each $(k-1)$-state it is possible to move to a k-state by fixing one of the unassigned components. Finally, according to the nature of problem, the component assignment order can be predetermined and fixed or be dependent on the choices the algorithm made for each k-state.

Rollout is a metaheuristic developed in three phases namely *Branching*, *Look-ahead* and *Selection*. The aim of the *Branching* is to create a set of feasible state candidates. To be more precise, consider that at the k-th iteration the algorithm has at its disposal a state s_{k-1} and a set $\mathcal{U}_k$ of $m-k+1$ unassigned components. Not all of them, however, are suitable for being assigned to s_{k-1}, hence a set $\mathcal{F}_k \subseteq \mathcal{U}_k$ is defined as the set of components $c \in \mathcal{U}_k$ that can safely be assigned to the state s_{k-1} to obtain a feasible new state $s_{k,c} = (s_{k-1}, c)$. The branching phase can be described, therefore, as the procedure in charge of appoint each unassigned component to the current state s_{k-1} and check the feasibility in order to generate a set of candidates for the k-state $\mathcal{C}_k = \{s_{k,c} | c \in \mathcal{F}_k\}$. The second phase is referred to as *Look-ahead* and employs one or more pilot procedures able to construct a complete solution for the problem starting from any candidate state in $\mathcal{C}_k$. In the combinatorial optimization literature, only applications of Rollout implementing the simplest scheme with only one pilot heuristic are reported. In this paper, however, we consider the more general strategy reported in the Algorithm 1. More precisely, let $\mathcal{H}$ be the set of q pilot procedures, the i-th method $H^i(\cdot) \in \mathcal{H}$ is a constructive algorithm that, starting from the candidate k-state $s_{k,c}$, produces a complete (or pilot) solution $H^i(s_{k,c})$ for the problem. Thereby, the look-ahead applies all the methods in $\mathcal{H}$ on each candidate k-state in $\mathcal{C}_k$, provided by the branching phase, and generates a set of complete solutions referred to as $\Lambda_c = \left\{\varsigma_c^i = H^i(s_{k,c}) | H^i(\cdot) \in \mathcal{H}, c \in \mathcal{F}_k\right\}$. Therefore, the aim of this phase is to use a set of methods in order to heuristically explore the space of all solutions that can be generated starting from a certain state. At each iteration of the algorithm, the *Selection* phase is responsible for driving the search towards the most promising direction and to achieve this goal it chooses one candidate state from $\mathcal{C}_k$ based on the quality of the pilot solutions generated during the look-ahead phase. Selection procedure assigns to each k-state $s_{k,c}$ a value of a *scoring function* φ. In order to evaluate the candidate components in the case of multiple pilot procedure, two main approaches are suggested in Bertsekas et al. (1997). They consist of calculating the value of the fitness function $\phi^i(\cdot) = \phi(H^i(\cdot))$ for each pilot heuristic and computing the scoring function at each state as:

- the weighted sum of the fitness values $\varphi(\cdot) = \sum_i w_i \phi^i(\cdot)$
- the minimum fitness value $\varphi(\cdot) = \min_i \phi^i(\cdot)$

We decided to implement the second policy because the minimum value returned by the pilot procedures represents a upper bound for the final solution and, therefore, by choosing it a no-worsening search process is assured. Furthermore, this policy has no parameters to adjust. The most promising component, i.e. the one having the best score is selected.

Algorithm 1: Multi-Heuristic Rollout

```
begin
    s_1 = (·)                                          ← a new solution is generated as a
    for k = 1, ..., m do                                 sequence of m states
        φ_best = +∞
        c_best = Null
        F_k = GetUnassignedComponents(s_k)             ← select feasible unassigned components
        forall the c ∈ F_k do                            for each component a candidate k-state
            s_{k,c} = (s_{k-1}, c)                     ← is genereated
            Λ_c = ∅
            for i = 1, ..., q do                         a set of complete solutions is
                ς_c^i = H^i(s_{k,c})                     generated applying q pilot heuristics
                Λ_c = {Λ_c, ς_c^i}                       to the k-state s_{k,c}
            φ(c) = min_{Λ_c} Φ^i(ς_c^i)
            if φ(c) ≤ φ_best then                        the state candidate linked to the best
                φ_best = φ(c)                            complete solution is selected
                c_best = c
        s_{k+1} = (s_k, c_best)                        ← the k-state is consolidated
```

The value of the scoring function $\varphi(\cdot)$ for $s_{k,c}$ is $\varphi(c) = \min_i\{\phi(H^i(s_{k,c}))\}$. Then, let $c_{best} = \arg\min_{c\in\mathcal{F}_k}\{\varphi(c)\}$, we set $s_k = (s_{k-1}, c_{best})$ fixing the k-th component of the solution. Note that the feasibility of the candidate component c at iteration k-th does not guarantee the feasibility of the complete solution reachable from s_k, at this aim we set $\phi^i(\varsigma_c^i) = \phi(H^i(s_{k,c})) = +\infty$ if the ith pilot heuristic fails in finding a feasible solution. These three phases are iteratively repeated until a complete solution is generated.

As the exploration of the whole search tree related to the procedure described via pseudo-code in Algorithm 1 can be time consuming (even when a single pilot procedure is used), suitable speed-up strategies have been designed and developed. This aspect is addressed in the next section.

In Figure 1 is reported a a schematic description of the multi-heuristic Rollout procedure. The algorithm starts from a dummy 0-state and analyzes four possible 1-state candidates (*Branching*), namely $s_{1,1}$, $s_{1,2}$, $s_{1,3}$ and $s_{1,4}$. For each candidate three pilot heuristics are executed (*Look-ahead*) and the value of the fitness ϕ^i is calculated. The candidate state associated with the lowest fitness (in this case $\phi^1(s_{1,2}) = 7$) is selected (*Selection*) and the 1-state $s_{1,2} = [2]$ is used as starting point for the next iteration. Following the same *Branching-Look-ahead-Selection* pattern states $s_{2,4}$, $s_{3,1}$ and $s_{4,3}$ are successively generated leading to the complete (and sub-optimal) solution $\varsigma = [2, 4, 1, 3]$.

2.2 *Speeding up basic and multi-heuristic Rollout algorithms*

As reported in the seminal works (Bertsekas et al. 1997; Duin and Voß 1994) and discussed by several authors (see Voß et al. (2005); Meloni et al. (2004); Guerriero et al. (2002); Guerriero and Mancini (2005)) one of the main drawback of Rollout metaheuristic is that, especially for large instances, it might require large computation times even when only a single pilot procedure is implemented. In fact, mainly during the first iterations, several alternatives must be evaluated yielding to a possible high execution time.

In this context, Guerriero and Mancini (2005) resorted to parallel processing in order to reduce Rollout's time requirements. Nevertheless, a different approach that does not call for more complex parallel implementations is possible. To this aim, several mechanisms have been proposed in the literature to improve the basic Rollout scheme and they can be classified according to the following criteria:

***Filter*:** At each step of the algorithm a speed-up can be achieved if only a reduced number of

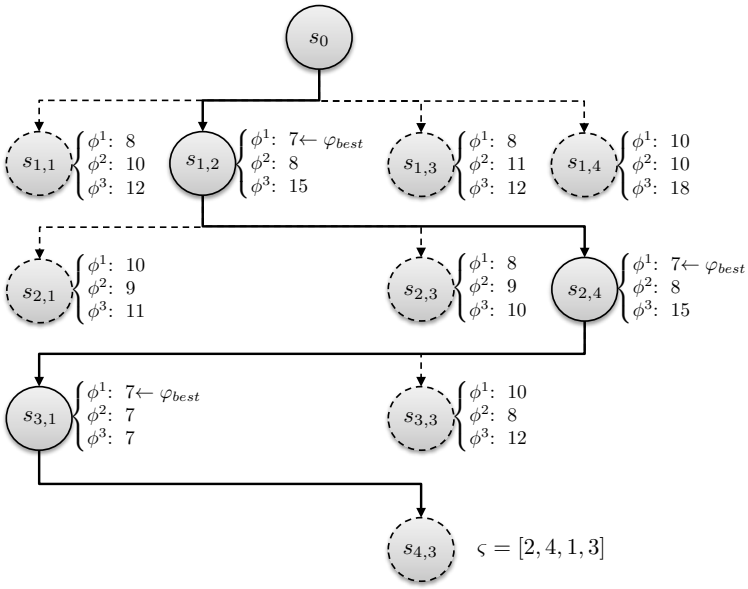

Figure 1. Example of Multi-Heuristic Rollout paradigm

alternative components c are evaluated, i.e. if a filter is applied in order to select only the most promising candidates. Different evaluation filters have been proposed by Meloni et al. (2004) and Voß et al. (2005).

***Incomplete run*:** In this case, the idea is to limit the algorithm's execution time by stopping it after a suitable number of iterations and returning the best pilot solution found so far as final outcome. Incomplete runs have been proposed by Meloni et al. (2004).

***Evaluation depth*:** Since the most consuming operation within Rollout is, by far, the execution of the pilot heuristics, a considerable speed-up is achieved by reducing the evaluation depth, that is by designing pilot procedures that return incomplete solutions. This approach has been applied in (Voß et al. 2005) for several combinatorial problems.

***Heuristic pruning*:** When a set of pilot heuristics is at the Rollout's disposal a speed-up is achieved by dynamically reducing their number to only the most promising ones.

All these mechanisms are aimed at reducing the Rollout's computational requirement at the expense of a, hopefully negligible, reduction of the solution quality. To compensate this unsought side effect, however, some expedient might be considered, we mention here the use of several pilot heuristics and the hybridization with a fast local search procedure.

To the best of our knowledge, no Rollout's implementation has been presented that uses more than one pilot heuristic. Notwithstanding, the use of a collection of pilot procedures is meant to explore a larger part of the solution space and this, therefore, should allow the method to find higher quality solutions. Moreover, this approach is particularly promising when different classes of instances have to be solved as each heuristic could be effective in solving one or more of them. However, for each pilot heuristic a remarkable overhead in terms of time has to be paid, for this reason such approach can be really beneficial only if it is associated with a mechanism for the dynamic selection of the most suitable heuristic.

In the next sub-sections we describe two speed-up strategies introduced to accelerate the search process, namely *Dynamic Heuristics Pruning* and *Candidate Reduction*.

Dynamic Heuristics Pruning (DHP)

This strategy applies only when several pilot heuristics are considered. The rationale of this speed-up mechanism is to dynamically reduce the number of pilot procedures during the algorithm execution, thus allowing an overall reduction of the running time trying, nonetheless, to not affect too much the solution quality. This mechanism is aimed at identifying and pruning the less effective heuristics,

i.e. those that most rarely return the best scoring solution. In order to implement such a pruning mechanism, the algorithm associates a counter $count_H(i)$ to each heuristic $H^i(\cdot)$ that is incremented every time the heuristic is *selected*. We say that a pilot heuristic is selected if its outcome is the solution that minimizes the scoring function within the pilot set for a certain stage. In case of ties all the related counters are incremented. Since many pruning paradigms are possible we implemented and analyzed in a preliminary study a variety of different strategies. Among those the three pruning policy reported below demonstrated to overcome all the others.

Multi Never-win **($\mathbf{P}_A$):** this policy considers the counters associated with the heuristics and at every iteration it removes those that were not selected, that is they did not return the best solution of at least one candidate k-state, namely their counter is equal to zero. The counters are reset at every iteration. Clearly, possibly no heuristics are eliminated.

Single Never-win **($\mathbf{P}_B$):** this policy acts in a similar manner to P_A, with the difference that at each iteration only one pilot heuristic is eliminated. In case of tie the first heuristic of the list is pruned. Also in this policy possibly no heuristics are eliminated.

Linear-race **($\mathbf{P}_C$):** every $\lfloor m/q \rfloor$ iterations (where m is the number of components and q is the number of pilot heuristics, with $m \geq q$) the worst pilot heuristic is removed. Every time one heuristic is discarded the counters are reset. Please notice the cardinality of $\mathcal{H}$ decreases linearly and that at the last iteration the pilot set contains exactly one heuristic.

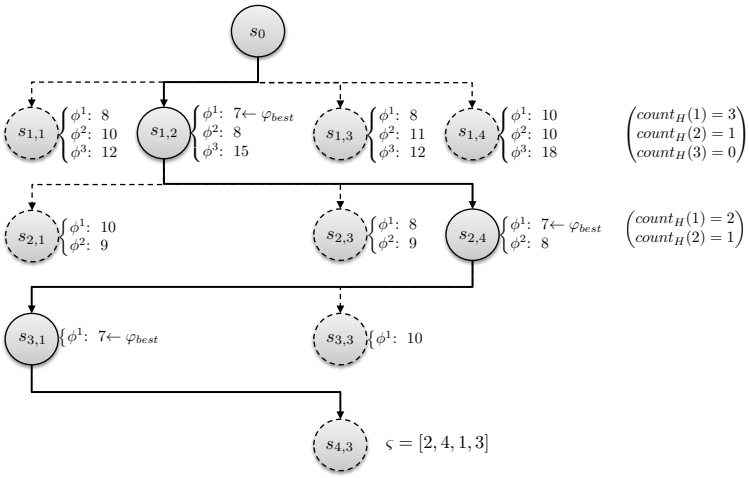

Figure 2. Example of Dynamic Heuristic Pruning (P_C)

Figure 2 shows a sketch of Rollout algorithm using the heuristic pruning technique P_C. In this simple four-iteration example the reader, helped by the reported counters $count_H(i)$, can observe that heuristic $H^3(\cdot)$ is never selected in the first iteration. Thus indicating that it could not be adequate in this particular context and for this reason it is dropped from the optimization process according to the adopted DHP. In the second iteration $H^1(\cdot)$ is selected twice while $H^2(\cdot)$ only once and hence it is pruned. Eventually, only $H^1(\cdot)$ is exploited to generate the optimized solution.

Dynamic Candidates Reduction (DCR)

As detailed in Section 2.1, at the k-th iteration the Rollout algorithm starts from a $(k-1)$-state, s_{k-1}, and a set of feasible candidate components $\mathcal{F}_k$, and returns a set of complete (hopefully feasible) solutions $\left\{\varsigma_c^i = H^i(s_{k,c}) | H^i(\cdot) \in \mathcal{H}, c \in \mathcal{F}_k\right\}$. The k-state is then created by choosing the component $c \in \mathcal{F}_k$ associated to at least one pilot heuristic returning the best candidate solution, that is the one that minimizes ϕ^i. Consequently, for a generic iteration $|\mathcal{F}_k| \times |\mathcal{H}|$ solutions are generated.

The aim of the previously introduced DHP is to reduce the algorithm's complexity by progressively pruning the less effective heuristics. Whereas, the goal of DCR is to identify the more promising

candidate components in order to have a smaller set of alternatives $\mathcal{F}_k^{'} \subset \mathcal{F}_k$ to evaluate. Obviously, the candidate reduction can be performed according the several possible policies. The one implemented in this work is based on a *proximity* principle. More precisely, consider that at the k-th iteration in the look-ahead phase the heuristic pool generates a set of full solutions and, to a closer inspection, not only the set of k-state candidates $\mathcal{C}_k$ but also sets of possible candidates for next remaining states ($\mathcal{C}_{k+1}, \mathcal{C}_{k+2}$, etc.).

DCR enacts a proximity-based policy by selecting $\mathcal{C}_{k+1} \subseteq \mathcal{F}_{k+1}$ ($\mathcal{C}_{k+1}$ is trivially a subset of all feasible unassigned components) as suitable set of candidates for iteration $k+1$. In a nutshell, the rationale behind this choice is that if one or more heuristics have elected a component to be the $k+1$ state in their solutions before, it is likely to be selected again since the heuristics (often stateless) tend to return similar solutions in two subsequent iterations. As a consequence, it is worth restricting the branching process to these elements alone discarding the others. In order to implement such a speed-up mechanism, the algorithm associates a counter $count_c(i)$ to each suitable candidate for the next iteration.

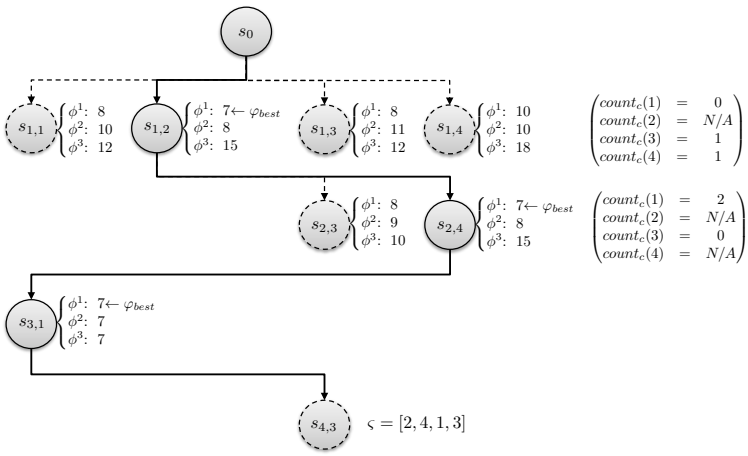

Figure 3. Example of Dynamic Candidate Reduction

In Figure 3 we depict the effects of applying this mechanism. Note that, in the second iteration $\mathcal{F}_2$ contains only components 3 and 4, meaning that none of the three heuristics considered in the example has returned a complete solution having component 1 placed in the second position (i.e., $count_c(1) = 0$). A similar consideration can be drawn as regarding the third iteration. In this case component 3 is discarded because it has been never selected as the third component of a complete solution at iteration 2 (i.e., $count_c(3) = 0$, while $count_c(1) = 2$).

3. Case study: Complex Parallel Machine Scheduling

In this paper, we deal with a multi-objective parallel machines scheduling problem under a number of both standard and realistic constraints, such as release times, due dates and deadlines, particular sequence-dependent setup times, machine unavailabilities, and maximum campaign or lot size. This problem has been introduced by Pacciarelli et al. (2011) and it is based on a real-world problem arising in a pharmaceutical manufacturing plant. The two considered case studies address the operations scheduling of dispensing and counting departments, respectively.

Following the Graham notation (Graham et al. 1979) the considered problems can be classified as

$$P_2|r_i, d_i, D_i, s_{ij}, MCS, unavail|Lex(L_{max}, C_{max}, U),$$

and

$$P_3|r_i, d_i, D_i, unavail|Lex(L_{max}, C_{max}, U).$$

in which, P_2 and P_3 indicate identical parallel machines production environment with 2 and 3 machines; r_i, d_i, and D_i indicate that the jobs have release times, due dates, and deadlines, respectively; s_{ij} indicates the presence of sequence-dependent setup times; MCS represents the constraint on the maximum campaign size; *unavail* represents the possible machine unavailability constraint; the objectives are (in lexicographic order): the minimization of L_{max} the maximum lateness, C_{max} makespan U the number of tardy jobs. The maximum campaign size constraint imposes that a major setup is incurred after a maximum number of consecutive operations belonging to the same family. Besides, there may be planned temporary machine unavailability, which must be taken into account when scheduling the production. This constraint allows to interrupt a setup operation and resume it at the end of the unavailability without penalties, whereas ordinary processing operations cannot be interrupted.

Parallel machine scheduling problems are widely studied (see for instance, Cheng and Sin (1990)) both with single and multiple objective functions (T'kindt et al. 2006). These kind of problems arise quite frequently in industrial applications as described by Ovacik and Uzsoy (1997). Scheduling problems with setups have been often addressed (Allahverdi 2015) in the literature, and a parallel machine problem with sequence dependent setup times is addressed by Lee and Pinedo (1997) while Tang (1990) also considers the case with minor and major setups. The latter case is particularly frequent in chemical and pharmaceutical production processes in which minor setups are related to successive processing of jobs belonging to the same family, while major setups occur when the families of two successive jobs are different and a more accurate time/work consuming changeover operation is required. A survey on algorithmic approaches to address family-based setups in parallel machines environments is given in Van der Zee (2015). Not all the constraints in these problems have been extensively studied, in fact the maximum size of a campaign (MCS) and the particular machine unavailability are not frequently addressed in the scheduling literature (Stefansson et al. 2006).

3.1 *Pilot heuristics*

In this subsection we briefly introduce a set of pilot heuristics used in our experiments with different Rollout schemes. These basic heuristics dispatch the pre-ordered jobs (one at a time) among the available machines preferring the machine with the shortest completion time. The pre-ordering of jobs is obtained in different ways determining an overall different behaviour of the respective pilot heuristic, as described in the following:

EDD: two versions of the Earliest Due Date (EDD) heuristic, proposed in (Jackson 1955), have been realized. The first, referred to as EDD_1, favors deadlines over due dates. For this reason, the jobs are preliminarily divided into three sets; to the first set are assigned the jobs having a deadline, the second one, instead, contains the jobs with only a due date, whereas the remaining jobs are gathered to create the third set. Eventually, a permutation is created by considering the jobs of the first set, ordered according to their deadlines, followed by the jobs of the second set, ordered according to their due dates, while the jobs of the third set, taken in the order they are loaded into the algorithm, completes the solution. The second version of this heuristic (EDD_2), due dates and deadlines are considered equivalent. Consequently, only two job set are created, the one containing jobs with either deadlines or due dates and the set of the residual jobs. Again, the permutation is created by concatenating the jobs of the

first set ordered according to their deadline or due date with the elements of the second sets

Algorithm 2: Modified Jackson Schedule (MJS)

```
input: 𝒥, the set of jobs
input: 𝒰, the set of machine unavailabilities
input: S, the current schedule
input: Δ ≥ 0, the selection threshold
begin
    ς = (·)
    repeat
        t = getMinStartingTime(S,𝒰,𝒥)                ← get the minimum starting time
        𝓜^t = getIdle(𝒮,𝒰)                            ← get machines idle at t
        𝒥^t = {i ∈ 𝒥 : r_i ≤ t}
        𝒥^t_D = {i ∈ 𝒥^t : D_i > 0}
        𝒥^t_d = {i ∈ 𝒥^t : d_i > 0}
        if 𝒥^t_D ≠ ∅ then                            select candidate jobs having deadline
            D_min = min_{i∈𝒥^t_D} D_i                in the range [D_min, D_min + Δ]
            𝒥_Cand = {j_i : D_i ≤ D_min + Δ}
        else if 𝒥^t_d ≠ ∅ then                       select candidate jobs having duedate in
            d_min = min_{i∈𝒥^t_d} D_i                the range [d_min, d_min + Δ]
            𝒥_Cand = {j_i : d_i ≤ d_min + Δ}
        else
            𝒥_Cand = {i : i = arg min_{i∈𝒥^t} r_i}   ← select candidate jobs with the smallest
                                                      release time
        j, m = findBestMatch(𝒥_Cand, 𝓜^t, S)         select the job and the machine with the
        S = updateSchedule(S, m, j)                  smallest setup and update the schedule
        𝒥 = 𝒥 \ {j}                                  S, the set 𝒥 and solution ς
        ς = (ς, j)
    until 𝒥 = ∅
    return S, ς
```

MJS: the rationale of this heuristic rule, which is a modified version of the Jackson Schedule (Jackson 1955), is to extend the EDD_1 rule by taking into account the state of the machine and of the job campaigns. Details on the MJS are illustrated in Algorithm 2. The reader can easily notice that, again, the a set of jobs is divided into three sorted groups (jobs with deadline, due date and others, respectively). However, unlike the EDD_1 rule only the jobs released before t (set $\mathcal{J}^t$) are eligible to be scheduled. Time t, in turn, is the smallest completion time calculated over all the machines considering the current partial schedule and the unavailabilities. If $\mathcal{J}^t_D \neq \emptyset$, that is if there is at least one candidate job $j \in \mathcal{J}^t$ with deadline associated, the smallest deadline D_{min} is found and all the jobs with deadline in the range $[D_{min}, D_{min} + \Delta]$ are selected. Δ is a selection threshold that allows the application to consider a variable number of jobs as possible candidates. A similar candidate selection procedure is implemented as far as the set of jobs with due dates is concerned whereas when it comes to consider the remaining jobs, the ones with minimum release time are selected. Finally, the candidate jobs are analyzed individually and the one that can complete the earliest, taking into account sequence-dependent setup times, job campaigns and machine availabilities is selected. The heuristic executes m step in total, being m the number of components in a complete solution; each time t and the current partial schedule S are updated accordingly.

SPT : this heuristic is the simplest one among the one realized to be part of the Rollout pilot heuristic. This approach implements the classical Shortest Processing Time ordering rule for single machine scheduling ordering the candidate jobs in non-decreasing order of their processing times.

Name	Heuristic	Name	Heuristic
H_0	EDD_1	H_5	MJS($\Delta = 7$ hours)
H_1	EDD_2	H_6	MJS($\Delta = avg(p_{i,j})$)
H_2	MJS($\Delta = 0$)	H_7	MJS($\Delta = \frac{avg(p_{i,j})}{2}$)
H_3	SPT	H_8	LLF($\Delta = 7$ hours)
H_4	LLF($\Delta = 0$)	H_9	LLF($\Delta = avg(p_{i,j})$)

Table 1. Pilot heuristics implemented for the case study

LLF: this pilot heuristic, whose name is the acronym for Largest Lateness First, implements a threshold mechanism similar to that described for the MJS. In particular, it considers the current schedule, and for each candidate job and each machine calculates the its lateness values; for each job the minimum possible lateness is considered and using such values the maximum lateness L_{max} is computed. Finally, the jobs with lateness in the range $[L_{max} - \Delta, L_{max}]$ are considered and, among those, the job with minimum completion time is selected and the schedule is updated accordingly.

4. Computational Experiments

The proposed algorithms have been tested on randomly generated realistic instances first introduced in Ciavotta et al. (2009), each instance representing a two-week production plan of two departments in a pharmaceutical plant. The instances can be divided into two groups according to the number of machines (two or three parallel machines). The number of jobs in the two machine group ranges from 40 to 60, whereas the instances with three parallel machines have from 400 to 480 jobs. A further classification divides the instances in three categories according to the presence of deadlines. In a first category jobs do not have deadline. In the second group 20% of the jobs have a deadline (i.e., are urgent or high priority jobs) and finally in the last category 40% of the jobs have a deadline. Moreover, instances can be further classified into two groups according to the number of random machine unavailabilities. More specifically, in the first group there is only one random machine downtime, in which a machine cannot process any jobs. More unavaliabilities are present in the second group of instances. On the whole there are 12 set of instances and each set includes 15 instances, for a total of 180 instances in the test set.

In this section we present the results of the experimental campaign obtained by varying each factor that defines the configuration of the algorithm in the two considered complex parallel machine scheduling problems. Five factors have been considered, namely:

Heuristic: A first factor specifies the heuristic adopted by the algorithm. Eleven levels have been considered. Ten levels are obtained by using each heuristic alone (H_0–H_9), while the last level (indicated as 10H) runs all the ten heuristic and selects the best resulting solution.

Rollout: This factor describes the use of the Rollout algorithm. A first level implies that no Rollout is applied to improve the solutions (NRH), whereas the second level implies the use of the rollout framework (RH).

Candidate Reduction: The candidate reduction factor assumes two levels, according to the use of the Dynamic Candidate Reduction speed-up introduced in Section 2.2 (DCR) or not (NCR).

Heuristic Pruning: The heuristic pruning factor describes the pruning strategies effects. It assumes 4 levels, one for each pruning described in Section 2.2 (P_A, P_B, P_C) and one for the reference case (NP), i.e., the no pruning case. Recall that, the pruning strategies can be applied only in the case of Rollout with multiple pilot heuristics.

Therefore, for example, the single stand-alone heuristic 6 is denoted simply as H_6, whereas H_6-RH is the Rollout using heuristic H_6 and a Rollout algorithm using all the ten heuristics with pruning strategy P_B and dynamic candidate reduction strategy is denoted as 10H-RH-DCR-P_B. Observe that the total number of configurations to be tested is 39.

Since each algorithm is deterministic there is no need of multiple runs to extract meaningful

Algorithm	Time	L_{max}	C_{max}	U	%feas
H_0	0.00	11101.64	23170.28	30.13	1.00
H_1	0.00	3614.37	20610.92	14.62	0.53
H_2	0.00	3618.37	20075.81	14.56	0.96
H_3	0.00	22521.33	25996.24	42.66	0.33
H_4	0.00	13419.49	25224.14	49.41	0.33
H_5	0.00	3210.57	20116.37	12.82	0.72
H_6	0.00	3210.57	20116.37	12.82	0.72
H_7	0.00	3210.57	20116.37	12.82	0.72
H_8	0.00	9776.22	20549.61	11.86	0.46
H_9	0.00	10020.23	20674.76	11.47	0.46
10H	0.00	3210.57	20116.37	12.82	0.72
H_0-RH	0.07	9860.73	22794.12	27.52	1.00
H_1-RH	0.07	2906.40	19799.06	9.47	0.70
H_2-RH	0.44	2781.99	19531.49	9.48	1.00
H_3-RH	0.09	9825.82	22012.80	28.33	0.37
H_4-RH	0.42	5297.37	21227.98	32.26	0.39
H_5-RH	0.48	2650.66	19586.27	7.59	0.90
H_6-RH	0.47	2650.66	19586.27	7.59	0.90
H_7-RH	0.49	2650.66	19586.27	7.59	0.90
H_8-RH	0.41	4717.06	20031.93	14.98	0.58
H_9-RH	0.41	4653.71	20107.48	15.17	0.60
10H-RH	3.28	2722.32	19552.49	9.11	1.00

Table 2. Performance of heuristics and Rollout on two-machine instances

results. A full factorial design of experiments resulting in 39 algorithm configurations has been applied to 180 instances for a total 7020 runs. All the algorithms are single threaded and written in C. They run on an Intel XEON E5420 processor at 2.5 GHz with 8 Gb of RAM under Linux operating systems.

The results are organized as follows: *(i)* first, in Section 4.1, we show the influence of the multi-pilot Rollout. In other words we are going to compare the effect of the single-pilot rollout algorithms over the simple heuristics and then we introduce the multi-pilot and analyze its effect; *(ii)* next, in Section 4.1.1, we evaluate effects of the candidate reduction speed-up technique; *(iii)* in Section 4.1.2 we show the effect of the second proposed speed-up (DHP) which can be applied only to multi-pilot Rollouts; *(iv)* finally, in Section 4.1.3, we report on the best configurations obtained by the test campaign for both the two- and three-machine problem.

4.1 *Effects of Multi-Heuristic Rollout*

In this section we study the performance of the Rollout algorithm over the simple heuristics and when the multiple heuristics are employed. The results are, therefore, restricted to 22 configurations without speed-ups:

- $\{H_0, \ldots, H_9\}$: the 10 single heuristics
- 10H: the best among the 10 stand alone heuristics
- $\{H_0\text{-RH}, \ldots, H_9\text{-RH}\}$: ten single-pilot Rollouts
- 10H-RH: the multi-pilot Rollout

The results are shown in Tables 2-3 for the two-machine and three-machine instances respectively. For each algorithm we present the computation time (Time) in seconds, the objective functions values of maximum lateness L_{max}, makespan C_{max} (both expressed in minutes), number of tardy jobs U (observe that the number of tardy jobs is computed over all the jobs having deadline or duedate), and the percentage of feasible instances (%feas), i.e., percentage of instances in which the deadline constraints are not violated. All the values in each row of the tables are the average over a set of 90 instances for each algorithm.

From Table 2 and Table 3 the single heuristic algorithms $H_0, \ldots, H_9$ have a very different behaviours with some heuristics being not able to obtain good average solutions while others only rarely are able to produce a feasible solution. Among them only H_2, H_5, H_6 and H_7 consistently

Algorithm	Time	L_{max}	C_{max}	U	%feas
H_0	0.00	12432.71	22618.09	220.67	1.00
H_1	0.00	3600.32	20637.89	45.23	0.33
H_2	0.03	1087.16	17802.61	9.62	1.00
H_3	0.00	23160.33	24851.32	390.63	0.33
H_4	0.05	17634.91	22408.68	225.61	0.33
H_5	0.03	965.49	17766.94	6.90	1.00
H_6	0.03	811.81	17727.78	5.30	0.96
H_7	0.03	811.81	17727.78	5.30	0.96
H_8	0.05	11676.60	18099.52	36.78	0.33
H_9	0.05	10950.60	18095.93	37.31	0.33
10H	0.30	811.81	17727.78	5.30	0.96
H_0-RH	100.44	11335.67	22406.21	200.04	1.00
H_1-RH	92.80	3480.08	20513.79	38.73	0.34
H_2-RH	1879.98	806.46	17517.51	5.61	1.00
H_3-RH	102.50	17971.97	23539.84	340.71	0.33
H_4-RH	3142.94	7527.69	20184.67	132.69	0.33
H_5-RH	1956.70	735.59	17545.09	4.17	1.00
H_6-RH	1948.26	584.11	17535.23	2.87	0.98
H_7-RH	1953.89	584.11	17535.23	2.87	0.98
H_8-RH	3122.44	6854.29	17831.09	31.32	0.40
H_9-RH	3132.90	5438.40	17905.19	31.72	0.42
10H-RH	17454.51	580.69	17531.00	3.76	1.00

Table 3. Performance of heuristics and Rollout on three-machine instances

yield to good performance both in terms of solution quality ($L_{\max}$) and feasibility (%feas). We observe that 10H (i.e., the algorithm selecting the best among the ten considered heuristics) is fast and able to obtain frequently a feasible solution.

When considering the simple Rollout algorithms (H_0-RH, ...,H_9-RH) we observe three very distinct behaviours.

In first place, the Rollout is always able to consistently improve the performance of each single pilot heuristic.

Secondly, we observe that some Rollout algorithms (i.e., H_1-RH, H_3-RH, H_4-RH, H_8-RH and H_9-RH) perform even worse than 10H. This clearly implies that the pilot heuristic on which they are based performs extremely poorly on average, and that the Rollout framework is not able to salvage the performance. In practice, some pilot heuristic could be designed to address a specific class of instances and this, in general, leads to worsening its average behaviour. It can be noted that by removing these heuristics also the multi-heuristic Rollout could improve its average performances. This observation is at the basis of DHP speed-up technique which dynamically tries to exclude a heuristic when it appears to be not effective on the given instance.

Third, there are some Rollouts (H_2-RH, H_5-RH, H_6-RH and H_7-RH) that improve over the simple 10H configuration even if they rely on a single pilot algorithm. The best simple Rollout algorithms turn out to be H_6-RH and H_7-RH, although their are not able to find a feasible solution in a few cases (about 3%). Observe that, H_6 and H_7 are two variations of the same basic greedy heuristic (MJS) obtaining the same results, and being the Rollout scheme deterministic, in this tests they obtain the same improvements. The computation times of the basic Rollouts for the two machine instances (Table 2) are always under half second, whereas for the larger and more complex three machine instances the CPU times range from 100 to 3000 seconds (Table 3).

However, when we compare these algorithms with the multi-heuristic Rollout (10H-RH) proposed in Section 2.1 we observe that the multi-pilot effect is evident since the quality of the obtained solutions is improved over all the single Rollouts. This benefit has a drawback of an increased computation time. When considering $C_{\max}$ objective we observe that the differences among the algorithms are evident when comparing the full multi-heuristic Rollout 10H-RH with the worst performing single greedy H_3. The improvement of the makespan $C_{\max}$ of 10H-RH over H_3 is almost 25% and 30% in the two-machine and three-machine, respectively. On the other hand, when considering the number of tardy jobs the ranges between the best performing and worst performing algorithms are even more remarkable (especially in the three-machine instances) since they range from 390 tardy

Algorithm	Time	L_{max}	C_{max}	U	%feas
H_*-RH	0.33	4799.50	20426.37	16.00	0.73
H_*-RH-DCR	0.03	6702.29	21342.76	20.52	0.66
10H-RH	3.28	2722.32	19552.49	9.11	1.00
10H-RH-DCR	0.80	2913.17	19612.59	9.88	1.00

Table 4. Effect of dynamic candidate reduction speed-up on two-machine instances

Algorithm	Time	L_{max}	C_{max}	U	%feas
H_*-RH	1743.29	5531.84	19251.39	79.07	0.68
H_*-RH-DCR	19.30	5978.03	19608.28	104.81	0.67
10H-RH	17454.51	580.69	17531.00	3.76	1.00
10H-RH-DCR	580.80	555.67	17458.38	4.63	1.00

Table 5. Effect of dynamic candidate reduction speed-up on three-machine instances

Algorithm	Time	L_{max}	C_{max}	U	%feas
10H-RH	3.28	2722.32	19552.49	9.11	1.00
10H-RH-P_A	0.44	3104.08	19750.87	11.06	1.00
10H-RH-P_B	1.06	2908.49	19602.99	9.66	1.00
10H-RH-P_C	2.82	2862.41	19539.08	9.47	1.00
10H-RH-DCR	0.80	2913.17	19612.59	9.88	1.00
10H-RH-DCR-P_A	0.06	3226.58	19851.88	11.63	1.00
10H-RH-DCR-P_B	0.27	3228.68	19849.57	11.66	1.00
10H-RH-DCR-P_C	0.61	2960.56	19658.61	10.09	1.00

Table 6. Effect of heuristic pruning speed-up on two-machine instances

jobs on average to less than 3. Of course the worst performing heuristics are the ones like H_3 that are based on the SPT rule, thus completely disregarding the presence of due dates and deadlines. Additionally, also the feasibility is not an issue since 10H-RH is the only algorithm able to always obtain a feasible solution on all the 180 instances. The large computation time required by the multi-pilot Rollout 10H-RH calls for the use of speed-up techniques that are the main focus of this paper.

4.1.1 Effects of candidate reduction

We now assess the effects of the candidate reduction speed-up for the two- and three-machine problems (Tables 4–5), respectively. Rows H_*-RH and H_*-RH-DCR show the average results of all the single Rollout algorithms grouped over the candidate reduction factor over a set of 90 instances. Whereas, 10H-RH and 10H-RH-DCR show the results of the multi-pilot Rollout configurations with and without dynamic candidate reduction and without applying pruning speed-up over a set of 90 instances.

From Tables 4 and 5 the average effects of the dynamic candidate reduction can be easily observed. Its adoption cuts the computing time up to two orders of magnitude with only a small degradation on the different objective functions. On the three-machine instances the 10H-RH-DCR even yields to a reduction of the maximum lateness over the 10H-RH algorithm.

4.1.2 Effects of heuristic pruning strategies

In order to assess the effects of heuristic pruning strategies we show detailed results of the multiple pilot heuristics Rollout configurations grouped by varying the heuristic pruning and candidate reduction factors. In Tables 6 and 7 each row refers to a specific configuration and the presented results are an average over 90 instances.

The effects of the three pruning strategies (P_A, P_B and P_C) are rather different. In fact, while P_A and P_B cut the computing times up to two order of magnitude at the expense of a small increase of the maximum lateness, P_C causes only a small reduction of the computing times but at almost no expense in term of solution quality. More in details, when considering the two-machine case (Table 6) we observe that the no-pruning option produces the best solution but employing more CPU time.

Algorithm	Time	L_{max}	C_{max}	U	%feas
10H-RH	17454.51	580.69	17531.00	3.76	1.00
10H-RH-P_A	318.89	693.92	17627.82	5.07	1.00
10H-RH-P_B	1033.03	678.36	17614.21	4.51	1.00
10H-RH-P_C	14243.42	575.47	17500.59	4.79	1.00
10H-RH-DCR	580.80	555.67	17458.38	4.63	1.00
10H-RH-DCR-P_A	147.80	705.47	17647.32	5.31	1.00
10H-RH-DCR-P_B	158.63	694.65	17637.38	4.92	1.00
10H-RH-DCR-P_C	467.84	567.41	17484.35	4.76	1.00

Table 7. Effect of heuristic pruning speed-up on three-machine instances

While adopting the pruning strategy one could reduce the CPU time at expense of solution quality: P_A and P_B are the more aggressive pruning options resulting in 10 times speed-ups at the expense of 6-14% of loss in solution quality, while P_C results in a smaller speed-up and a loss limited to 5%. Interestingly, no option results dominated. When considering three-machine instances (Table 7) the situation is similar. P_A and P_B are extremely aggressive having a 100 times speed-up and a 16-19% of loss, whereas P_C dominates the no pruning option since it is able to produce solution having a slightly better quality and being slightly faster. This could happen since the use of heuristic pruning may lead the search process on a different area of the search space thus finally yielding to different final solutions. Among the three proposed pruning strategies we observe as P_A and P_B have a similar behavior and when compared to P_C both lead to a larger reduction in computing times but at the expense of a worse solution quality (L_{max}), while smaller relative differences are observed in terms of C_{max}.

When considering the combined effects of dynamic candidate reduction and heuristic pruning speed-ups we observe that the computation times are further reduced, and the difference between the faster pruning strategies P_A and P_B and the slowest strategy P_C is reduced.

When considering the secondary objectives ($C_{\max}$ and U) the use of the speed-ups does not influence their values too much. This is not surprising considering that in the lexicographic order they are evaluated after the maximum lateness.

With respect to the feasibility, it is interesting to observe how the use of the proposed speed-ups does not affect the feasibility of the solutions and that all the proposed speed-ups when applied to the multi-pilot Rollout lead to a reduction of computing times but without impacting on the feasibility.

4.1.3 Non-dominated configurations

As it often happens in practice, when selecting the "best" algorithm for a given problem the decision makers have to find a balance between the available computational time and the solution quality. To this aim, we now analyze the trade-off faced by decision makers when evaluating the adoption of a multi-pilot and the proposed speed-ups described in this paper.

In Figures 1 and 2 we show the Pareto front obtained by all the considered configurations for the two- and three-machine instances, respectively. More precisely, Figure 1 (a) and 2 (a) show all the non-dominated Rollout configurations, while 1 (b) and 2 (b) restrict the analysis only to the non-dominated multi-pilot Rollout configurations. Each dot on the plots shows the computational time/solution quality (expressed in terms $L_{\max}$) of a given algorithm. In order to maintain the clarity, only the non-dominated algorithms are represented in the plot. An algorithm results dominated if there exists at least another algorithm which, on average, obtains better solutions (lateness objective function) in smaller computational time.

From Figure 1 (a) there are five non-dominated Rollout algorithms. The fastest option is the Rollout (H_3-RH-DCR) using only H_3 with DCR enabled. This configuration results in a negligible computation time although the solution quality, expressed by the L_{max}, is extremely poor. Configuration H_1-RH-DCR only requires 0.01 seconds and it is able to greatly improve over the previous configuration. Then configurations H_5-RH-DCR and H_6-RH-DCR both require only 0.03 seconds to attain a 2859 of maximum lateness. Finally the last two non-dominated configurations (H_2-RH and

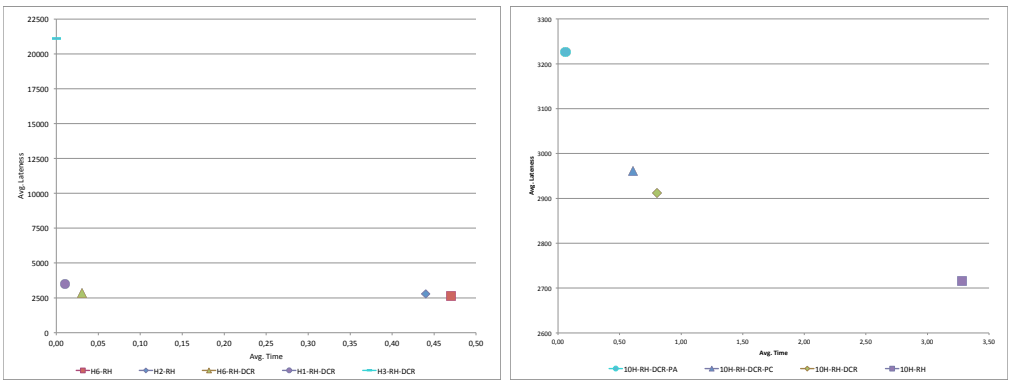

Figure 1. (a) Non-dominated Rollout configurations on two-machine instances. (b) Non-dominated multi-pilot configurations on two-machine instances.

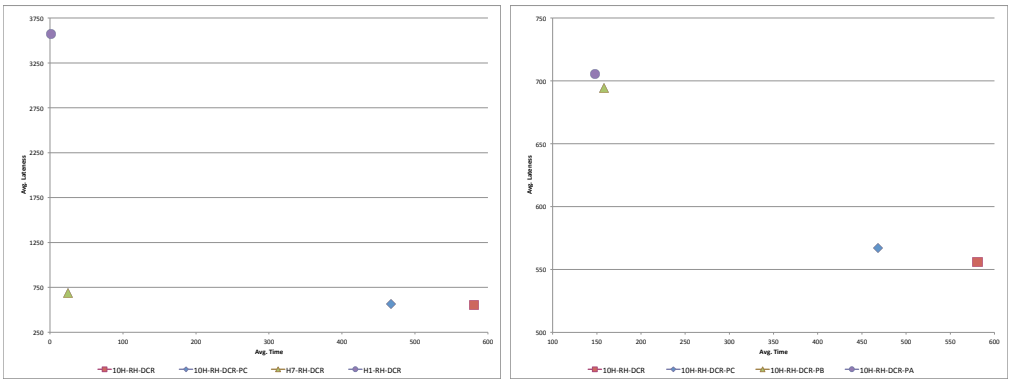

Figure 2. (a) Non-dominated configurations on three-machine instances. (b) Non-dominated multi-pilot configurations on three-machine instances.

H_6-RH) require more CPU time (about 0.4 seconds) but they improve over the previous alternatives in terms of solution quality. From this figure we observe as the candidate reduction produces faster algorithms, while best results are obtained when no speed-up is adopted.

When restricting the analysis to only multi-pilot configurations on the two-machine instances (Figure 1 (b)) the faster options uses both speed-ups (10H-RH-DCR-P_A and 10H-RH-DCR-P_C). The fastest configuration requires 0.06 seconds and the maximum lateness is 3226, while using P_B as heuristic pruning strategy requires 0.6 seconds for a 2960 maximum lateness. The next configuration does not adopt any pruning strategy (10H-RH-DCR), it still requires less than 1 seconds to be solved and slightly improves over the previous configuration. The last configuration instead does not include any speed-up and results in a substantially slower algorithm (3.28 seconds) but with $L_{max} = 2715$. When comparing this set of non-dominated solutions with the one shown in Figure 1 (a) we observe that the multi-pilot Rollouts are dominated by their single pilot counterparts.

When considering the three machine cases Figure 2 (a) the non-dominated configurations are as follow. The fastest algorithm is a Rollout with H_1 and dynamic candidate reduction. This algorithm requires about 1 second but produces poor solutions ($L_{max} = 3567$). Switching from H_1 to H_7 the resulting algorithm requires 25 seconds but slashes the maximum lateness to 690. Further lateness reductions can be obtained by using a multi-pilot Rollout with dynamic heuristic reduction and P_C pruning but at the expenses of greater CPU requirements (467 seconds). The slowest non-dominated configuration (580 seconds), although yielding to the better solutions ($L_{max} = 555$), is 10H-RH-DCR. From this plot we can observe as all the non-dominated configurations make use of candidate reduction speed-up, and the multi-pilot is used to achieve better performance. This finding is of interest since it shows that the effects of multi-pilot can be better than just using one single highly performing heuristic. In other words, the combined use of different heuristics yield to better performance.

While restricting the analysis to only multi-pilot Rollouts (Figure 2 (b)) the non-dominated

configurations are, as in Figure 2 (a), 10H-RH-DCR and 10H-RH-DCR-P$_C$ and two additional configurations. Namely, 10H-RH-DCR-P$_B$ and 10H-RH-DCR-P$_A$. These two "fast" configurations require 158 and 147 seconds respectively, and their solutions have, on average, about 700 minutes of maximum delay. Also in this case all the non-dominated configurations adopt the dynamic candidate reduction speed-up.

5. Conclusions

In this paper we extended the Rollout framework by using multiple pilot heuristics and introducing some speed-up techniques. We evaluated the proposed extensions on some complex parallel machine scheduling problems. Rollout algorithms are known to be, an easy to implement, general constructive metaheuristic. However one of their main drawbacks is the high computing time requirements. The proposed speed-ups strategies have been applied to two pharmaceutical manufacturing problems involving both standard and uncommon constraints. From an extensive campaign of computational experiments turns out that the use of multiple heuristics in the Rollout scheme yields to better results overcoming the single heuristic approaches and that the proposed speed-up techniques result in a remarkable reduction of computing times at the expense of a minor reduction in solution quality. Overall the resulting algorithm, when compared to the standard Rollout approach, produces better quality solutions while maintaining acceptable computation times.

The main finding of the paper is that multi-heuristic Rollout with speed-ups can be an easy to apply algorithm ables to accomodate several heuristics into a single adaptive metaheuristic framework enabling improvements over the performances of the single heuristics. These methods are inherently simple and easy to code. Moreover, their results are easy to reproduce, adopt and extend for industrial practitioners enabling a learning-by-improving process. This approach is particularly promising when different classes of instances have to be solved as each basic heuristic could be effective in solving one or more of them.

Future research directions include the development and test of different speed-up methods, applying these techniques to other scheduling issues and more general combinatorial optimization problems, analyze the performance ratios of the Rollout schemes (e.g. see Bertazzi (2012); Mastin and Jaillet (2014), and exploring the potentialities offered by parallel and cloud computing frameworks.

References

Allahverdi, A. 2015. The third comprehensive survey on scheduling problems with setup times/costs *European Journal of Operational Research* 246 345–378.

Amberg, A., L. Gouveia, P. Martins, and S. Voß. 1999. Iterative heuristics metastrategies. In *Proceedings of the Third Metaheuristics International Conference MIC'99, Rio de Janeiro, Brazil,* 13–16. July.

Bertazzi, L. 2012. Rollout Minimum and Worst-Case Performance Ratios of Rollout Algorithms. *Journal of Optimization Theory and Applications*, 152: 378–393.

Bertsekas, D. P., J. N. Tsitsiklis, and C. Wu. 1997. Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics* 3 (3): 245–262.

Bertsekas, D. P. 2013. *Rollout Algorithms for Discrete Optimization: A Survey.* In P.M. Pardalos et al. (eds.), *Handbook of Combinatorial Optimization,*. 2989–3013. Springer, New York.

Cheng, T. C. E., and C. C. S. Sin. 1990. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research* 47 (3): 271–292.

Ciavotta, M., C. Meloni, and M. Pranzo. 2009. Scheduling Dispensing and Counting in Secondary Pharmaceutical Manufacturing. *AIChE Journal* 55 (5): 1161–1170.

D'Annibale, G., R. De Leone, P. Festa, and E. Marchitto. 2007. A new meta-heuristic for the Bus Driver Scheduling Problem: GRASP combined with Rollout. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007),* 192 –197. April.

Duin, C., and S. Voß. 1994. Steiner Tree Heuristics: A Survey. In *Proceedings of the 22nd Annual Meeting of DGOR,* 485–496.

Duin, C., and S. Voß. 1999. The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks* 34 (3): 181–191.

Glover, F., and E. Taillard. 1993. A user's guide to tabu search. *Annals of Operations Research* V41 (1): 1–28.

Golden, B., S. Raghavan, and D. Stanojević. 2008. The prize-collecting generalized minimum spanning tree problem. *Journal of Heuristics* 14 (1): 69–93.

Graham, R.L., E.L. Lawler, J.K. Lenstra, and A.H.G Rinnooy Kan. 1979. Optimization and approximation in deterministic machine scheduling: a survey. *Annals of Discrete Mathematics* 5: 287–326.

Guerriero, F., and M. Mancini. 2005. Parallelization Strategies for Rollout Algorithms. *Computational optimization and applications* 31 (2).

Guerriero, F., M. Mancini, and R. Musmanno. 2002. New Rollout Algorithms for Combinatorial Optimization Problems. *Optimization methods and software* 17 (4): 627 – 654.

Jackson, J.R. 1955. *Scheduling a Production line to Minimize Maximum Tardiness.* Research Report 53. Los Angeles, CA: Management Science, University of California.

Lee, Y. H., and M. Pinedo. 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research* 100 (3): 464–474.

Mastin, A., and P. Jaillet. 2014. Average-Case Performance of Rollout Algorithms for Knapsack Problems. *Journal of Optimization Theory and Applications* 165 (3): 964–984.

Meloni, C., D. Pacciarelli, and M. Pranzo. 2004. A Rollout Metaheuristic for Job Shop Scheduling Problems. *Annals of Operations Research* 131 (1-4): 215–235.

Néron, E., F. Tercinet, and F. Sourd. 2008. Search tree based approaches for parallel machine scheduling. *Computers & Operations Research* 35 (4): 1127–1137.

Ovacik, I.M., and R. Uzsoy. 1997. *Decomposition methods for complex factory scheduling problems.* Boston/Dordrech/ London: Kluwer Academic Publishers.

Pacciarelli, D., C. Meloni, and M. Pranzo. 2011. *Models and Methods for Production Scheduling in the Pharmaceutical Industry.* Vol. 152 of *International Series in Operations Research and Management Science.* 429–459. Springer New York.

Secomandi, N. 2003. Analysis of a Rollout Approach to Sequencing Problems with Stochastic Routing Applications. *Journal of Heuristics* 9 (4): 321–352.

Stefansson, H., N. Shah, and P. Jensson. 2006. Multiscale planning and scheduling in the secondary pharmaceutical industry. *AIChE Journal* 52 (12): 4133–4149.

Tang, C. S. 1990. Scheduling batches on parallel machines with major and minor set-ups. *European Journal of Operational Research* 46 (1): 28–37.

T'kindt, V., J.-C. Billaut, and H. Scott. 2006. *Multicriteria Scheduling: Theory, Models and Algorithms.* Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Tu, F., and K. R. Pattipati. 2003. Rollout strategies for sequential fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 33 (1): 86–99.

Van der Zee, D.-J. 2015. Family-based dispatching with parallel machines. *International Jorurnal of Production Research*, in press.

Voß, S., and C. Duin. 2003. Look Ahead Features in Metaheuristics. In *MIC2003: The Fifth Metaheuristics International Conference, 79-1,* .

Voß, S., A. Fink, and C. Duin. 2005. Looking Ahead with the Pilot Method. *Annals of Operations Research* 136 (1): 285–302.