Intl. Trans. in Op. Res. xx (2021) 1–25

Pattern mining-based pruning strategies in stochastic local searches for scheduling problems

Arnaud Laurent[a,*], Damien Lamy[b], Benjamin Dalmas[c] and Vincent Clerc[b]

[a]*INRIA Lille, France*
[b]*Mines Saint-Etienne, Univ Clermont Auvergne, CNRS, UMR 6158 LIMOS, Institut Henri Fayol, F - 42023 Saint-Etienne France*
[c]*Mines Saint-Etienne, Univ Clermont Auvergne, CNRS, UMR 6158 LIMOS, Centre CIS, F - 42023 Saint-Etienne France*
E-mail: [a]*name.surname@inria.fr;* [b,c]*name.surname@emse.fr*

Received DD MMMM YYYY; received in revised form DD MMMM YYYY; accepted DD MMMM YYYY

Abstract

Scheduling problems are a subclass of combinatorial problems consisting of a set of tasks/activities/jobs to be processed by a set of resources usually to minimize a time criterion. Some optimization methods used to solve these problems are hybridized with knowledge discovery techniques to extract information during the optimization process and enhance it. However, most of these hybrid techniques are custom-designed and lack generalization. In this paper a module for knowledge extraction in Stochastic Local Searches is designed, aiming to be problem independent and plugged into optimisation methods that relies on multiple Stochastic Local Search replications. The objective is to prune parts of the search space for which the exploration is likely to lead to poor solutions. This is performed through the extraction of high-quality patterns occurring in locally optimal solutions. Benchmarked on two well-known scheduling problems, the Job-shop Problem and the Resource Constrained Project Scheduling Problem, the results show both a speed up in the convergence and the reaching of better local optima solutions.

Keywords: Stochastic Local Search; Pattern Mining, Machine Learning; Scheduling Problem, Job-shop, Resource Constrained Project Scheduling Problem

1. Introduction

Scheduling problems are a subclass of combinatorial problems that have drawn a lot of attention. A scheduling problem consists in a set of tasks/activities/operations (”task” will be used in the remaining of this paper) that have to be processed using a set of given processors and some additional resources [Blazewicz et al. (1996)]. The objective function to minimize is usually related to time functions such as the total processing time, also called makespan.

*Author to whom all correspondence should be addressed (e-mail: arnaud.laurent@inria.fr).

There exist several approaches to solve such problems. This paper is focused on Stochastic Local Search (SLS), a widely used optimization method. As stated in the book written by Hoos and Stützle (2004), SLS techniques consist in "selecting an initial candidate solution, and then proceeds by iteratively moving from one candidate solution to a neighbouring candidate solution, where the decision on each search step is based on a limited amount of local information only". Accordingly, these methods can range from simple iterative improvement algorithms to more complex methods, such as metaheuristics. Referring to the authors' description of SLS methods, this paper focuses on iterative improvement SLS. The principle of such an SLS is to randomly explore the neighborhood of a solution (i.e. a set of solutions), accepting only better or equal quality neighbors as the " strong randomisation of local search algorithms [...] can lead to significant increases in performance and robustness" Hoos and Stützle (2004). This exploration process allows reaching solutions that are locally optimal. Classical operators for metaheuristics consist in perturbation mechanisms and improvement mechanisms. Yet, the latter are generally done through multiple SLS phases that, in most cases, do not take into account the potential information gained during the previous iterations.

To address these limitations, the combination of optimization algorithms and knowledge discovery algorithms has drawn a lot of attention these past years. The goal of such hybrid approaches is to use the solutions generated by the optimization algorithm to extract useful insight and improve the optimization process. In many applications, incorporating knowledge discovery techniques have been proven to perform better in terms of convergence speed and solution quality, as shown in Zhang et al. (2011) for machine learning and evolutionary algorithms.

However, these hybrid techniques are specifically designed to solve a specific problem with a specific algorithm, which lacks generalization. In this paper, a stand-alone learning mechanism is introduced, based on patterns, to extract information from previous local optimum solutions. The objective is to speed up the convergence of SLS toward good solutions. To do so, it is proposed to identify the patterns that lead to good solutions and use them to influence the neighboring systems. These patterns represent a precedence relation between tasks in the schedules. Patterns are extracted according to their quality, i.e., their impact on the optimized objective function and their frequency. During local searches, a lot of moves are performed, and some of these moves can be accepted whereas they are not presumably good ones to reach efficient solution. The approach aims at reducing the search space of local searches by prohibiting some moves that would break a pattern that is likely to occur in good solutions. In other words, the goal is to remove parts of the search space for which the exploration would probably not improve the solutions or even lead to a bad quality local minimum. Figure 1 is an exemplified demonstration of these pruning strategies. If AB and CD are identified to be good patterns, we aim to prevent searching for solutions where these patterns are not present (hatched squares).

This method has the ambition to be easily adaptable to methods with several SLS replications and effective in several scheduling problems. To assess its effectiveness, two well-known scheduling problems are investigated: the Job-shop Scheduling Problem (JSP) and the Resource Constrained Project Scheduling Problem (RCPSP). The paper is a preliminary study and aims at exploring the usage of pattern mining as a way to improve SLS for scheduling problems. At the best of our knowledge this has never been presented in other works. However, it is not yet appropriate to improve state-of-the-art approaches in scheduling, which would require further investigations. The remaining of the paper is as follows. Section 2 introduces the related works on knowledge discovery for scheduling problems. In section 3, the two scheduling problems addressed in this paper are presented with their specific features. In section

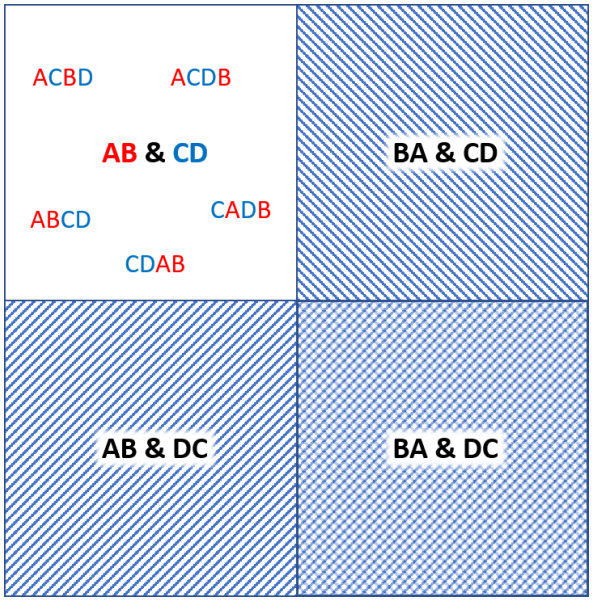

Fig. 1: Reduction of the search space with two good patterns AB and CD

4, patterns are defined and the methodology to extract them is provided. Then, section 5 exposes the quality of the patterns on the accessibility of solutions through the exact solving of some JSP instances. Then the focus is given to the performance of SLS, where pattern removal is prohibited. The final section concludes and gives future research prospects.

2. Related work on Knowledge discovery for scheduling problems

As can be stressed from the following literature, combinatorial problems can benefit from learning schemes. Knowledge discovery and operations research, in general, have shown some synergies by integrating data mining into optimization techniques [Aytug et al. (1994); Olafsson et al. (2008); Meisel and Mattfeld (2010); Corne et al. (2012); Talbi (2016)]. In this section, the focus is given to the integration of knowledge discovery techniques for scheduling problems. However, this section does not aim at being exhaustive in a broad sense. To that end, the reader is referred to Lin and Gen (2018) for a complete review of optimization techniques improved with knowledge discovery. This section also does not focus on metaheuristics containing a learning aspect on parameters on the probabilities of applications of neighborhood systems. Instead, the literature is investigated with a problem-specific or instance-specific perspective and our contribution is positioned accordingly. In section 2.1, *Problem-specific learning* is addressed, that aims at extracting information from already solved instances in order to solve new instances. In opposition, the objective of *instance-specific learning* is to solve an instance of a problem without any prior knowledge of the problem structure and is discussed in section 2.2.

2.1. Problem-specific learning

Problem-specific techniques usually rely on *a priori* knowledge learned on the problem (e.g., problem features, specificities,...) or on the features shared among the best solutions known (e.g., size, structure,...) regardless of the instances being solved. It means that different new instances will be solved using the same *a priori* information.

Chiu and Yih (1995) study dynamic scheduling in distributed manufacturing systems. They propose a tolerance-based learning algorithm to deduce dispatching rules from obtained solutions. The dynamic scheduling scheme significantly outperforms the static version with a single dispatching rule in a distributed manufacturing system.

Koonce and Tsai (2000) extract knowledge from a large set of Job-shop schedules with data mining. They used an attribute oriented induction method to explore the relationship between an operations' sequence and a set of rules. The obtained rules can provide solutions that outperform classical dispatching heuristics. On the same problem Kumar and Rao (2009) propose a similar approach to extract a rule set scheduler based on data obtained with an ant colony optimization technique. They extract the rules with a decision tree obtained with the See5 software.

In their work Priore et al. (2001, 2006, 2010), the authors evaluate the performances of different combinations between an optimization method and several machine learning techniques: neural networks, decision trees, support vector machines (SVM) and nearest neighbors algorithms. The different learning techniques give similar results, the SVM performing slightly better. It is to be noted that in addition to using more data and computing power, neural networks do not perform better than the other techniques.

Zare and Fakhrzad (2011) uses the extracted data to build a knowledge base and inject it to solve new instances of the JSP and improve the performance of genetic algorithms. Shahzad and Mebarki (2012) aims at a similar objective but using a tabu search-based approach for the JSP with maximum tardiness. For the single machine problem, Jain et al. (2012) also consider a data mining-based approach to discover previously unknown priority dispatching rules.

Karthikeyan et al. (2012) investigate a multi-objective Flexible Job-shop Scheduling Problem where the makespan, the total machine workload, and critical machine workload are optimized. A Particle Swarm Optimization algorithm and a data mining approach are considered. The mining approach consists in learning from a sample set of solutions in which the zone of sequence operations should be added.

Wang et al. (2015) also proposes dispatching rules for the JSP based on learning from a set of solutions using timed Petri net. They also provide a Petri net-based branch and bound algorithm to solve this problem.

In Sun et al. (2015) the authors proposed a two-layer surrogate-assisted Particle Swarm Optimization (PSO), in which a global and several substitute models are employed for fitness approximation. The global model is processed beforehand and used as a cheap approximation substitute to replace in part the computationally expensive exact fitness evaluations known to be a major drawback of PSO. Park and Kim (2017) also developed a method to approximate the fitness function in PSO but using a Generalised Regression Neural Network (GRNN).

Feng et al. (2017) proposes three types of approaches to reuse knowledge from past search experiences to enhance the search performance of their algorithm. These approaches are namely exact storage and reuse of past solutions, the reuse of model-based information and the reuse of structured knowledge

captured from past optimized solutions.

Clustering and association rule mining algorithms are other well-known discovery techniques used in Gholami and Hafezalkotob (2018) to schedule maintenance tasks. The authors use prior knowledge as input for these algorithms to extract helpful information. This allows them to detect frequent relations indicating warning signs and guide their algorithm toward better scheduling of future maintenance.

2.2. Instance specific learning

In opposition to problem-specific learning techniques, instance-specific learning techniques dynamically extract information about a specific instance during its optimization process. It means the information used is only related to the current instance and the mining procedure has to be embedded in the optimization process. In many examples, the developed methods strongly aim at a specific problem class.

In Lee et al. (1997), a machine learning and genetic algorithm approach in the context of dynamic Job-shop scheduling is studied. Machine learning is applied for releasing jobs for production according to dispatching rules and the genetic algorithm is used to schedule jobs at each machine. Genetic algorithms (GA) have been hybridized with data mining for instance-specific learning.

In Mashhadi et al. (2003), the authors improve the convergence of the genetic algorithm with a new local optimizer based on Lamarck's theory of evolution.

Ali et al. (2013) proposes two different schemes to improve a genetic algorithm. The authors focused on using information about the instance to bias the generation of the initial population toward the global optimum. The authors use nonlinear simplex and quadratic interpolation to generate better starting points than random initialization. This technique is applied to different benchmark functions, showing an improvement in results.

Guerine et al. (2016) coupled data mining with metaheuristics to guide the solution toward better cost heuristics and applied their method to a 1-commodity Pickup-and-Delivery Traveling Salesman Problem. The authors designed an algorithm to select the patterns present in elite solutions and define the most frequent binary arcs and connect them. This application differs from most existing techniques in the sense it aims at dealing with sequences, in which the order of tasks matters, instead of sets in which only the presence or absence of a task is evaluated.

In Wang and Tang (2017) a machine learning based multi-objective memetic algorithm is designed for the discrete permutation Flow-Shop Scheduling Problem. A multi-objective local search is developed, where the analysis of historical data stored during the search process is used to determine which non-dominated solutions should be selected for local search.

In Thevenin and Zufferey (2019), the authors propose a Variable Neighborhood Search algorithm enhanced with a learning mechanism that enhances the search toward promising areas of the search space. This technique is applied to a single machine scheduling problem with rejections, setups, and earliness and tardiness penalties and seems to outperform existing techniques.

2.3. Discussion and contribution position

The different techniques discussed show interesting results in comparison to traditional techniques, and emphasize the benefits to include knowledge discovery approaches into optimization methods. However, as is, they cannot be applied to meet our requirements.

In this paper, the idea of a technique-and-problem-independent optimization is strongly defended. This raises two main limitations among existing techniques:

- Problem-specific learning techniques have to be discarded since the objective of this research work is to improve the convergence of SLS regardless of the scheduling problem under study. Therefore the only accessible information is related to the current instance being solved. This major limit leads to consider other types of techniques, in this case, instance-specific learning techniques.
- Even though some authors claim their technique to be flexible, they are mostly custom-designed for a specific algorithm to solve a specific type of problem.

Our work aims to speed up local searches, avoiding the poor quality solutions neighborhood. Existing methods use the information extracted in current solutions to enforce the knowledge in the next solutions that are expected to be in promising areas of the search space. In this paper, the objective is not to have an early guess on where the good solutions are and enforce this information into the solution. Instead of restricting the solutions into some parts of the search space, forbidden moves are considered. This way, the search space is pruned, while maintaining the accessibility to optimal solutions through the characteristics of the initial solutions.

Following the different limitations highlighted in this section, the contribution of this paper is twofold. First, a knowledge discovery module to enhance Stochastic Local Search-based optimization techniques is proposed. Secondly, a module that aims at improving the Local Searches of most problems from the class of scheduling problems is investigated. This approach is tested on JSP and RCPSP instances.

For ease of reference, all notations used in following sections are summarized in Table 1.

3. Scheduling problems

In this section, the scheduling problems and the methods that are used to solve them are introduced. To evaluate the consistency and performance of the provided methods, two case studies are investigated: the Job-shop Scheduling Problem and the Resource Constrained Project Scheduling Problem. For both problems, the representation of solutions and the neighboring systems used are presented.

3.1. The Job-shop Problem

The Job-shop Scheduling Problem (JSP) is a well known NP-hard scheduling problem that allows modeling a large number of industrial production systems. In the context of such a manufacturing system, a set J of jobs has to be processed on a set M of machines. Each job $j \in J$ has its own process plan (i.e. list of ordered operations) which consists in visiting a subset $M' \subseteq M$ of machines in a predefined order. Each operation belongs to one job only and is noted $\{\pi_i\}_{i\in[1,N]}$, with N the total number of oper-

Table 1: Frequent notations used throughout the paper

Scheduling Problems	J	Set of jobs
	M, M'	Set of machines
	N	Total number of operations/tasks
	p_i	Duration of operation/task i
	Π	Sequence of operations/tasks
	π_i	i^{th} operation/task to be performed
	K	Number of types of resources
	P_j	Set of predecessors of operation/task j
	$r_{j,k}$	Amount of resources of type k required by task j
	R_k	Amount of resources of type k available
	d_j	End time of task j
	av_r	Availability period of resource r
Patterns	T	Set of tasks to schedule regardless the problem addressed
	$I = \langle t_1, t_2 \rangle$	The pattern $t_1 t_2$, where $\langle t_1, t_2 \rangle \in T^2$
	$\neg I$	The reverse pattern of I
	P	Set of sequences of operations
	P_I	Set of sequences containing the pattern I
	sup_I^P	Support of the pattern I in P
	$score_I^P$	Average objective function value of solutions containing the pattern I
	obj_Π	Value of objective function of vector Π (e.g. makespan)
	IS_I^P	Improvement score of a pattern I.
	$minSup$	The minimum support for a pattern to be considered
	$Freq^P$	The set of patterns I with a support $\geq minSup$ and $\neg I$ with a support $\geq minSup$
	σ^P	The standard deviation of improvement score for the patterns in $Freq^P$
	GP	Set of selected patterns

ations. Each operation π_i must be processed on a predetermined machine $m \in M'$ with duration p_i. A machine m can process only one operation at a time, and an operation cannot be processed by more than one machine at a time. In this research project, the deterministic case of a job-shop is addressed where the number of jobs to be processed is known ahead of the planning. No preemption is allowed, and no machine-failure is expected.

3.1.1. The solution encoding and decoding scheme

For the JSP, an effective representation of solutions has been introduced in Bierwirth (1995) in the context of a genetic algorithm. This representation consists in a repetition vector, where an operation is modeled with its job number. Hence, each occurrence of a job number in the list corresponds to an operation to schedule. For instance, consider the vector [2 1 2 3 2 3 1 1 3] which represents a schedule for a problem with 3 jobs and 3 machines. Suppose that all jobs must be processed by each machine once. All operations can be numbered $\{\pi_i\}_{i\in[1,9]}$. The first appearing job in the list is job 2, which implies to schedule the first operation of this job, then the first operation of job 1 is scheduled. In the third position, job 2 has to be scheduled again, which corresponds to the second operation in its process plan. The decoding of the vector continues until all operations are scheduled. The main advantage of this representation is that it characterizes a topological order which means that no cycle nor infeasible vector

can be encountered. During decoding, the vector can be transformed to its equivalent operation list Π which is $[4\ 1\ 5\ 7\ 6\ 8\ 2\ 3\ 9]$, and where each value i corresponds to a π_i.

3.1.2. The neighborhood system

Let S be a solution of a JSP. The neighborhood of S is defined as the set of solutions that are obtained from S by applying one or several specific operators (called the neighborhood system). If classical local searches for the JSP and its extensions usually rely on the exploration of the critical path [Bürgy (2017); Kemmoé-Tchomté et al. (2017)], the current research work considers a larger neighborhood system that enables to reach better solutions, but at the cost of higher convergence time. In the case of the present study, a simple $swap$ operator is considered.This operator consists in selecting two random operations that belong to different jobs, and to exchange their position in the vector. Empirical experiments have demonstrated that this approach yields better results in a standalone context (i.e. this local search is not embedded in a metaheuristic).

3.2. The Resource-Constrained Project Scheduling Problem

The Resource-Constrained Project Scheduling Problem (RCPSP) is one of the most complex scheduling problems. This problem is a generalization of several classical scheduling problems, notably the JSP. In this section, only the single mode RCPSP is considered, which means that every task has only one way to be executed.

The objective is to schedule a set of N tasks using a set of resources of K different types. The tasks $j = 1, N$ have a duration of p_j and have to be executed with no preemption. Each task j has a set of other transactions P_j that has to be ended before the start of the task j. Each task needs an amount $r_{j,k}$ of resources of type $k = 1, K$ assigned during all its duration. When a task is completed, all resources assigned are available again. The total amount of resources available is noted R_k for the resources of type k.

3.2.1. The solution encoding and schedule generation scheme

The solution encoding used is a global order Π of all the tasks to schedule, respecting the precedence relations, such that:

- $\Pi = \langle \pi_1, \pi_2, ..., \pi_N \rangle$ is an order of the N tasks to schedule which respects the precedence constraints. Thus, a task i linked by a precedence constraint $i \in P_j$ implies that $i \prec j$ in Π. This order also implies that for any resource that is used by the tasks i and j, if i precedes j in the Π order, the resources must process i before j.

A schedule is also defined by a starting date d_j for all the tasks $j = 1, N$. For each order Π an infinity of solutions can be considered. To obtain these dates, a Schedule Generation Scheme (SGS) is applied on the ordered list Π, inspired by the work of Carlier (1984).

The principle of the SGS used is: "schedule as soon as possible all the tasks, in the Π order". To do that, the algorithm needs to compute the set of resources that will process the task j during the time p_j. For the first task, the period of availability $av_{k,r}$ for a resource r of type k is equal to 0. In the Π order,

the $r_{j,k}$ earliest available units r (with available time av_r) of type k resources are determined to process the task j. The ending date of the task j is computed by the equation 3.

$$A = \max_{h \in P_j}(d_h) \tag{1}$$

$$B = \max_{r \in R}(av_r) \tag{2}$$

$$d_j = max(A, B) + p_j \tag{3}$$

Then, the new availability date av_r for the resources $r \in R$ are set to the value d_j.
The Schedule Generation Scheme is presented in algorithm 1.

Algorithm 1 Schedule Generation Scheme for the RCPSP

Require: Π
  $av := \{0, ..., 0\}$;
  $d := \{0, ..., 0\}$;
  for $j = \pi_1$ to π_N **do**
    Computation of the set of resources $r \in R$ with the lowest av_r
    Computation of the completion time d_j (Eq. 3)
    Update of the av_r date to d_j for the resources $r \in R$
  end for

3.2.2. The neighborhood system

The neighborhood system used is an insertion of a random task in the Π order. An insertion move can be valid only if no precedence constraint is violated. It means that if two tasks are linked by a precedence constraint, these two tasks cannot be inverted in Π.

4. Pattern detection and usage

In this section, the learning mechanism proposed and the usage of this knowledge is introduced. At first, the notion of global order is introduced, relying on previously stated solution structures for the JSP and RCPSP. In section 4.2, the concept of pattern is defined as well as the proposed methodology to extract and use them in stochastic local searches.

4.1. Global order for scheduling problems

In the remaining of this paper, Π represents a global order for a solution of a scheduling problem, based on the definition introduced in section 3.2:

- $\Pi = \langle \pi_1, \pi_2, ..., \pi_N \rangle$ is a global order of the N tasks to schedule that respects the processing order by the resources. This implies that for any resource that is used by the tasks i and j, if i precedes j in

the Π order, the resource must process i before j.

For every valid solution of a scheduling problem, at least one global order Π exists on the tasks. The only way that no Π order would exist for a solution is the presence of a cycle in the processing order of the tasks on resources. However, this cycle implies a deadlock on the resource processing order and results in a solution S not valid.

It is to be noted that a solution can often be represented by many different vectors Π.

4.2. Patterns

In this section, the methodology to extract relevant patterns is introduced starting with background definitions related to patterns. Then the methodology to extract these patterns is addressed in section 4.2.2 while their use in a stochastic local search-based method is given in section 4.2.3.

4.2.1. Pattern definition

Regardless of the scheduling problem addressed, let T be the set of tasks to execute. A pattern $I{=}\langle t_1, t_2\rangle, \forall t_1, t_2{\in}T$ is a sequence of two tasks in which the task t_1 is scheduled before the task t_2. This implies that if these two tasks share a common resource, t_1 will be processed before t_2. A vector Π contains a sequential pattern I, noted $I \in \Pi$, if and only if I is a subsequence of Π; i.e., the tasks contained in I occur in the same order in Π:

$$I = \langle t_1, t_2\rangle \in \Pi \iff \exists i, \exists j \mid i < j \wedge (t_1{=}\pi_i \in \Pi) \wedge (t_2{=}\pi_j \in \Pi) \tag{4}$$

Conversely, $I{=}\langle t_1, t_2\rangle{\notin}\Pi$ denotes the situation where a pattern I is not contained in a vector Π, i.e., if t_2 precedes t_1 in Π. The reverse pattern of $I{=}\langle t_1, t_2\rangle$ is noted $\neg I{=}\langle t_2, t_1\rangle$.

$$I \notin \Pi \iff \neg I \in \Pi \tag{5}$$

For simplicity reasons, in the remaining of this paper we will refer to a pattern as the concatenation of the tasks involved; i.e., $I{=}\langle t_1, t_2\rangle$ will be referred as: t_1t_2.

As stressed in the above definition, this preliminary work only considers size-2 patterns. Besides this study's aim, which is to investigate the feasibility of the approach with small patterns first, this restriction can be justified as follows:

- The validity of a solution when a neighborhood system as defined in section 3 is applied over a pattern of size larger than two seems to be very time-consuming;
- In theory, the transitivity property does not hold in pattern mining; i.e., having the patterns t_1t_2 and t_2t_3 respecting a specific criterion does not ensure the pattern t_1t_3 to respect it as well. However, in practice, it seems this property tends to be true considering solutions close to local/global optima.

4.2.2. Pattern extraction

In this section, the methodology to extract the patterns defined in section 4.2.1 is given. This preliminary work focuses on stochastic local searches, therefore let $P = \{\Pi_1, \Pi_2, ..., \Pi_{|P|}\}$ be the set of $|P|$ lists used in a such a procedure. P_I is defined as the set of lists containing the pattern I, $P_I = \{\Pi | \Pi \in P \wedge I \in \Pi\}$. We also define the complement $P_{\neg I}$ as the set of lists not containing the pattern I, $P_{\neg I} = P \backslash P_I$.

The investigated methodology aims at extracting "good" patterns, i.e., the patterns that appear to be responsible for higher-quality solutions. In this work, a good pattern is conceptually defined as a sequence of tasks that appear in a specific order in Π among the best solutions. To assess the quality of a pattern, two metrics are used: the *support* and the *improvement score*.

The support of the pattern I in a population P, noted sup_I^P, is based on the absolute definition used in data mining papers [Fournier-Viger et al. (2017)], i.e., in our case the number of solutions in P containing I. It is computed as:

$$sup_I^P = |\{\Pi \in P | I \in \Pi\}| \tag{6}$$

In the case of the present study, the support of a pattern I in an initial population P will simply be $sup_I^P = |P_I|$; and by extension $sup_{\neg I}^P = |P| - sup_I^P = |P| - |P_I| = |P_{\neg I}|$.

The $score$ is a metric used to assess the role a pattern is expected to have in the quality of a solution. In scheduling problems, the objective function is used to assess the quality of a solution; e.g. the makespan. Therefore, we define the score of a pattern I in a population P, noted $score_I^P$, based on the objective function of all solutions that contain I as:

$$score_I^P = \frac{\sum_{\Pi \in P_I} obj_\Pi}{sup_I^P} \tag{7}$$

with obj_Π the value of the objective function of the solution associated with the vector Π.

The $improvement\ score$ of a pattern I in a population P, noted IS_I^P, is the average improvement quality in solutions provided by the presence of the pattern I. It is defined as follows:

$$IS_I^P = score_I^P - score_{\neg I}^P \tag{8}$$

with $score_{\neg I}^P$ the score metric applied to every solution not containing the pattern I.

In this research project, we aim at extracting good patterns from a set of locally minimal solutions. Extracting directly the patterns with the highest support and/or score, as often performed in the literature [Fournier-Viger et al. (2017)], seems to be an intuitive idea. However, a pattern can be frequent for other reasons. These patterns, which are defined as *noisy*, can occur mainly for two reasons:

- Precedence relations: if there is a precedence relation between two tasks i and j, no valid solution can contain these tasks in the reverse order. Therefore, it will necessarily imply the existence of a pattern $I = \langle t_1, t_2 \rangle$, with $i = t_1$ and $j = t_2$, that has a perfect support $sup_I^P = |P|$.
- Search space exploration: depending on the algorithm used, it is not possible to discard the probability that the local searches have not explored the search space well enough; resulting in similar final solutions even among the best solutions. To assess the responsibility of a pattern in the quality of a solution, it is important to ensure the presence of enough lists with the reverse pattern.

Consequently, as defined in Papon (2016), the extraction of *reasonably frequent patterns* is considered in this paper; i.e., patterns selected in a frequency range. In this perspective, two thresholds are defined.

- The minimum support $minSup$, to which both the support of a pattern I and the support of its reverse pattern $\neg I$ have to be greater or equals, in order to be considered; $minSup \leq sup_I^P \wedge minSup \leq sup_{\neg I}^P$. We define this set of considered patterns in population P as $Freq^P$.
- A minimum improvement score $minIS$. It ensures that the difference in the average value of the objective function for the solutions containing the pattern I, compared to the lists not containing it, has to be higher than $minIS$, $IS_I^P > minIS$. In our case, this threshold is used to tune the concept of "quality" of a pattern, based on the variation of the score of the different solutions in the set P.

The value of $minIS$ should adapt to any instance of any scheduling problem. To do that, a computational method to obtain a consistent value for $minIS$ is proposed.

Extracting the good patterns remains the same as removing the patterns that are on average not good enough. The uninteresting patterns I are expected to have an IS_I close to 0. As the $score_I^P$ is computed for a pattern as the average value of the solutions that contain I, the $score_I^P$ is a random variable following a normal law centered in $E(score_I^P)$ for $sup_I^P \geq minSup$, with $minSup$ a number large enough (central limit theorem). To use the central limit theorem, we fix $minSup = 30$. To approximate the standard deviation value σ of the score of patterns not influencing positively the value of the solutions, we consider the standard deviation of the whole pool $Freq^P$ of patterns I with $sup_I^P \geq minSup \wedge sup_{\neg I}^P \geq minSup$. From this set of score, the approximate value σ^P is computed as in equation 9.

$$\sigma^P = \sqrt{\frac{\sum_{I \in Freq^P} \left(score_I^P - score_{\neg I}^P\right)^2}{|Freq^P|}} \tag{9}$$

Given this approximation of the distribution of uninteresting patterns, it is considered in this work that all the patterns I with a value $IS_I \leq 2 \times \sigma^P$ are probably following a normal law centered in its expected value $\mu = 0$.

Thus, we consider the set GP of interesting patterns I such that $GP = \{I | I \in Freq^P \wedge IS_I^P > minIS\}$. We consider $minIS = 2 \times \sigma^P$, as it is expected, with a good probability, that these patterns are following a normal law centered in $\mu > 0$.

Altogether, the remaining patterns GP can fairly be considered to positively impact the objective function among the solutions of the problem under study.

4.2.3. Usage of the knowledge in local searches

To consider these patterns GP in local searches, the possibility to break a pattern I is prohibited within the neighborhood systems used as presented in Figure 2.

To illustrate this process, let us consider an example using the neighborhood system of the local search for the RCPSP presented in section 3.2.2. This move is an insertion of a task in the order Π that respects the precedence constraint as shown in Figure 3. The move done by the neighborhood system must now also respect the patterns GP: if a pattern $I \in GP$ is present in the solution Π, the neighborhood system must not remove it (Figure 3). In this example, patterns such as DB in GP, do not impact the moves of

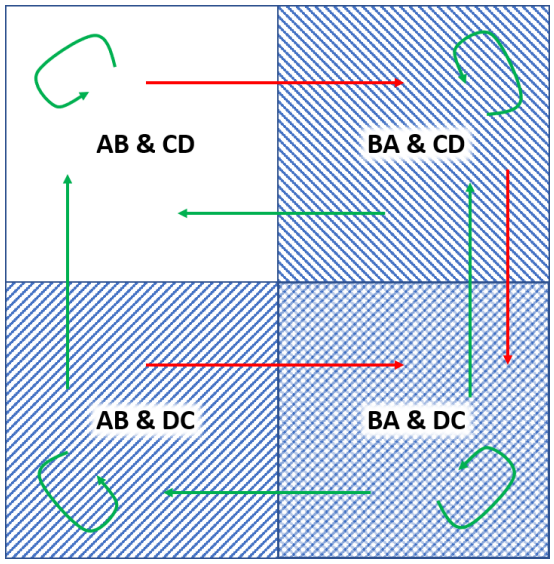

Fig. 2: Moves prohibited in red and allowed in green with $GP = \{AB, CD\}$.

E. Moreover, in this solution, DB is not respected, as B precedes D in the vector. This situation often happens in the process of the SLS, as an initial solution of an SLS has no obligation of containing any of the patterns in GP.

The Figure 4 presents the flow chart of the Stochastic Local Search. This process can potentially remove the accessibility to optimal solutions from some initial solutions. Indeed, if all the optimal solutions contain the pattern $\neg I$ and $I \in GP$, for all initial solutions containing I, the optimal value will never be reached during local searches. However, the process does not impose any pattern in the solution, and thus all the solutions can be reached depending on the initial solution of the SLS. In the case of stochastic local searches within metaheuristics, it would be the responsibility of the perturbation mechanism to ensure the accessibility to an optimal solution. In section 5, the accessibility to the optimal value is discussed if the patterns GP are imposed in every solution.

5. Computational experiments and discussion

In this section, we try to demonstrate the interest of the learning process between several replications of stochastic local searches. In several methods, such as genetic algorithms or evolutionary local searches, local searches are executed through several generations of solutions. In most of these methods, no knowledge is extracted from the first local searches that could be used in the following ones. Suppose that 1000 generations are applied. Instead of relying on the sole metaheuristic scheme, we propose to learn at one moment (we choose the first generation to base the knowledge on independent solutions) and to use the acquired knowledge in the 999 following generations. We consider in this part, the extraction of the set GP of interesting patterns from the locally minimal population P obtained from SLS on random

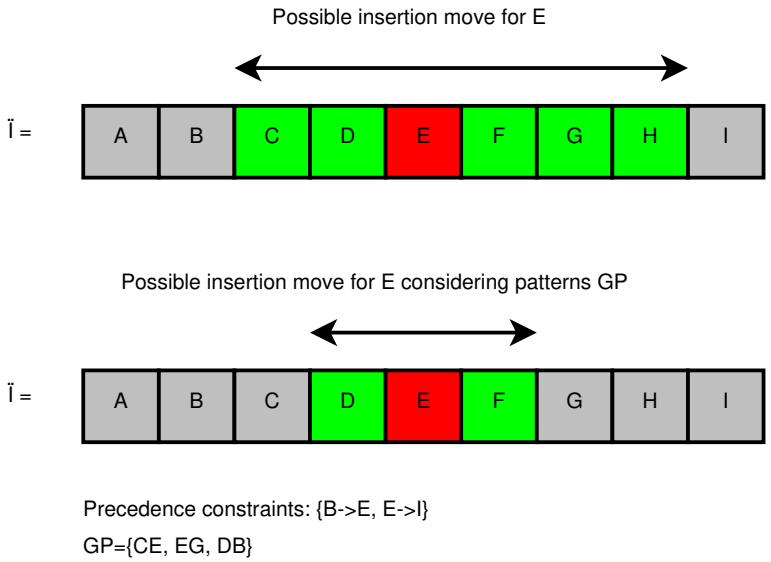

Fig. 3: Possible moves for the insertion neighborhood system for the RCPSP with and without the consideration of patterns GP

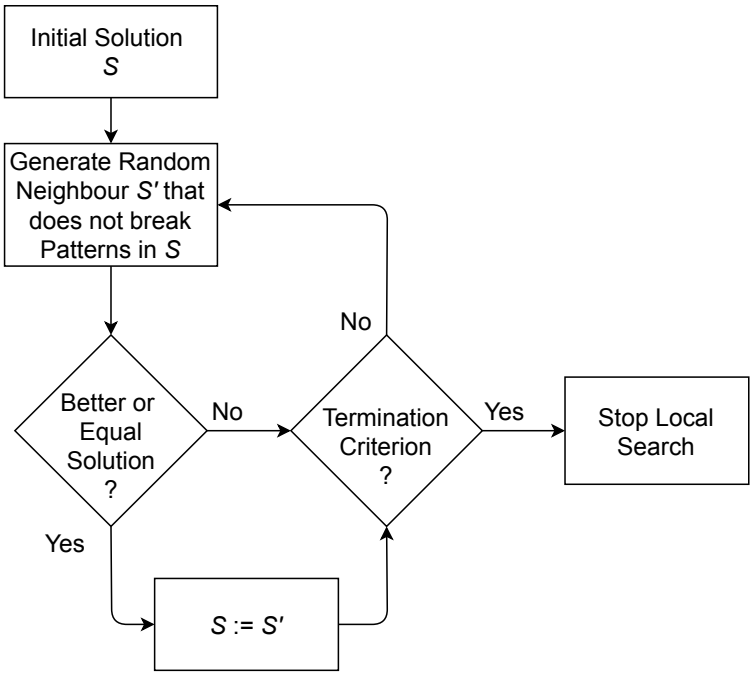

Fig. 4: The stochastic local search algorithm with pattern consideration

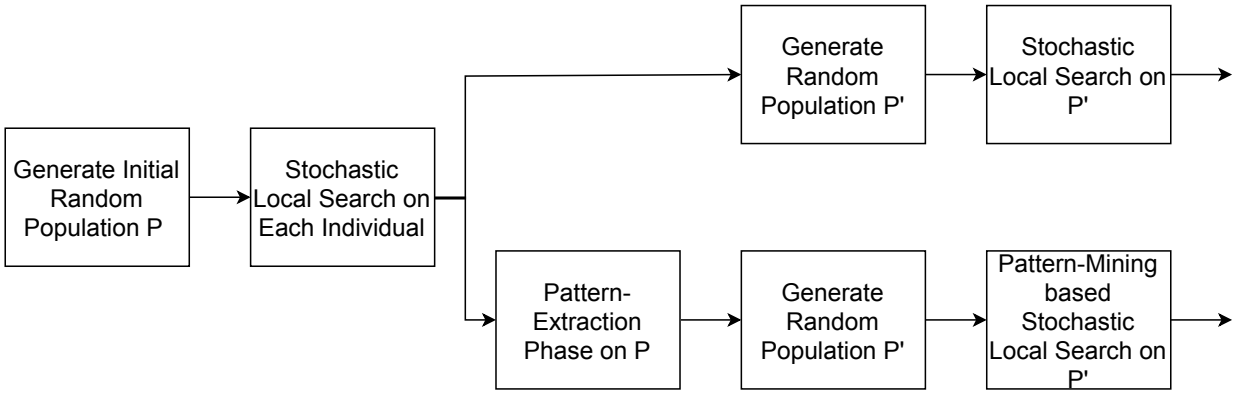

Fig. 5: Process used to compare the results with and without knowledge extraction from a population P

solutions. Indeed, according to first experimental results, it seemed that learning on a random solutions population was not sufficient and led to inconsistency in obtained results, such as deadlock induced by extracted patterns (see Section 4.1).

These are the situations in which the provided module aims to be used. In order to show the potential interest of the module, we try to answer two major questions:

- Do the extracted patterns keep the accessibility to an optimal solution if they are all imposed in solutions? In other words, does the pruning of the search space still lead in best case scenarios to optimal solutions?
- Does the pruning strategy in SLS lead to better local optimum? Are the average values of found solutions better than without the learning process? Are the best found solutions better than without the learning process?

In order to answer the first question, we solve the Job-Shop instances with constraint programming with and without imposing the patterns in GP. This experiment gives a good overview on the probability to loose the access to the optimal value when all the patterns are imposed.

In order to answer the second question and assess the performance of the learning module on SLS we apply the process presented in Figure 5. An initial population P is generated randomly then SLS are applied on solutions and knowledge is extracted from them. Then a new population P' is generated randomly and the pattern mining based Stochastic Local Search is applied to evaluate the impact of the pattern-based pruning strategy compared to the classical SLS. As the main difference between the two SLS for the scheduling problems at hand concerns the applied neighborhoods, the generation of a neighboring solution S' is relying on the permutation of operations for the JSP, and on the insertion of operations for the RCPSP.

5.1. Settings

Several experiments are conducted on classical instances taken from the JSP and RCPSP literature. The instances used consist in 40 test problems taken from Lawrence (1984) ranging from 50 to 300 operations and 10 instances from the PSPLIB Kolisch and Sprecher (1997) ranging from 90 to 120 operations. Results are computed on a Xeon E3-1505M 3GHz with 16Go RAM and running Win10. A time limit of 7200 seconds is considered for the constraint programming method. To have a fair comparison of local searches, two stopping criteria are used:

- A dynamic number of iterations. The SLS stops when a number of iterations without improvement is reached. The number of iterations without improvement is given by equation (10):

$$ln(2) \times N \times (N-1) \tag{10}$$

  and insures a minimum probability of 0.5 to reach a local optimum [Fleury (1993)].
- A total number of 10k iterations.

The Constraint Programming model used for the JSP is the cumulative one, based on the work of Bourreau et al. (2019). In the second subsection of the local searches tests, each replication is done as presented in Figure 5 on a population of $|P|$=300 solutions.

5.2. Validation through Constraint Programming

As several patterns are extracted during the learning process, it seemed important to assess how close the results are from the optimal solution when all patterns GP are imposed in every solution. To this purpose, CP Optimizer for constraint programming (CP) is used. Results are displayed in Table 2. In this table, **Instance_size** refers to the instance tested and its number of operations. **Opt** is the optimal solution obtained with CP without GP, and **CPU(Opt)** the associated computation time in seconds. $avgS$ refers to the average solutions over 10 runs. **%dev** corresponds to the deviation between $avgS$ and Opt. **CPU(s)** corresponds to the aggregated computation time in seconds. **CPU(SLS)** corresponds to the specific local search computation time while **CPU(GP)** corresponds to the required computation time for learning. |**opt**| corresponds to the number of optimal solutions over the different runs when GP is considered. The average number of patterns used during the optimization runs is denoted $|GP|$. Finally, **Magnitude** represents the number of instances for which optimal solutions are always found, while **%OPT** represents the percentage of optimal solutions over the different runs.

Table 2: Comparison of results with and without patterns GP imposed in every solution in the context of Constraint Programming

Instance_size	Without GP		With GP - 10k							With GP - dyn						
	Opt	CPU(s)	avgS	%dev	|opt|	CPU(s)	CPU(SLS)	CPU(GP)	|GP|	avgS	%dev	|opt|	CPU(s)	CPU(SLS)	CPU(GP)	|GP|
la01_50	666	<1	666	0	10	4	4	<1	6.9	666	0	10	2	2	<1	6.4
la02_50	655	<1	655	0	10	4	4	<1	4	655	0	10	3	2	<1	7.2
la03_50	597	<1	597	0	10	4	3	<1	14.1	597	0	10	2	2	<1	14.3
la04_50	590	<1	590	0	10	4	3	<1	7.1	590	0	10	3	2	<1	10.7
la05_50	593	<1	593	0	10	3	3	<1	0	593	0	10	1	1	<1	4.5
la06_75	926	<1	926	0	10	4	4	<1	0	926	0	10	3	3	<1	1.8
la07_75	890	<1	890	0	10	4	4	<1	13	890	0	10	4	4	<1	13.7
la08_75	863	<1	863	0	10	5	4	<1	9.3	863	0	10	4	4	<1	11.4
la09_75	951	<1	951	0	10	5	4	<1	1.9	951	0	10	3	3	<1	3.5
la10_75	958	<1	958	0	10	4	4	<1	0	958	0	10	3	3	<1	0
la11_100	1222	<1	1222	0	10	6	6	<1	0	1222	0	10	6	6	<1	0
la12_100	1039	<1	1039	0	10	6	6	<1	2.6	1039	0	10	7	6	<1	2.4
la13_100	1150	<1	1150	0	10	6	5	<1	0	1150	0	10	6	6	<1	3.2
la14_100	1292	<1	1292	0	10	6	5	<1	0	1292	0	10	6	5	<1	0
la15_100	1207	<1	1207	0	10	6	6	<1	29.9	1207	0	10	10	9	<1	27.6
la16_100	945	1	945.7	0.07	4	10	6	<1	11.6	949	0.42	5	14	11	<1	13.9
la17_100	784	<1	784	0	10	9	7	<1	13.6	784	0	10	13	12	<1	12.9
la18_100	848	1	848.4	0.05	9	10	7	<1	16.6	848	0	10	11	9	<1	14.5
la19_100	842	2	842.2	0.02	8	14	8	<1	14.9	842	0	10	17	9	<1	15
la20_100	902	1	902	0	10	12	8	<1	14.7	902	0	10	13	9	<1	15.4
la21_150	1046	51	1049.1	0.30	5	50	12	<1	27.9	1047.8	0.17	5	91	50	<1	25.7
la22_150	927	7	927.5	0.05	9	19	13	<1	32.4	927	0	10	68	59	<1	29.9
la23_150	1032	<1	1032	0	10	14	13	<1	23.4	1032	0	10	56	55	<1	25.6
la24_150	935	13	936.2	0.13	4	30	11	<1	25.3	936.9	0.20	3	58	33	<1	28.3
la25_150	977	30	977.2	0.02	9	59	10	<1	33.2	977.8	0.08	6	85	32	<1	28.6
la26_200	1218	1	1218	0	10	18	16	1	53.2	1218	0	10	116	113	1	48.9
la27_200	1235	620	1236.1	0.09	9	175	18	1	60.6	1235.4	0.03	9	329	105	1	57.6
la28_200	1216	1	1216	0	10	27	18	1	44.5	1216	0	10	129	119	1	38.7
la29_200	1152	5850	1156.6	0.40	2	759	18	1	49.5	1154.4	0.21	3	533	129	1	50.6
la30_200	1355	1	1355	0	10	20	19	1	46.7	1355	0	10	149	147	1	50.1
la31_300	1784	<1	1784	0	10	32	27	4	102	1784	0	10	221	216	3	26.9
la32_300	1850	<1	1850	0	10	31	27	4	102.4	1850	0	10	229	225	3	0
la33_300	1719	<1	1719	0	10	31	27	3	103.7	1719	0	10	237	232	3	57.7
la34_300	1721	1	1721	0	10	32	27	3	96.2	1721	0	10	272	266	4	48.5
la35_300	1888	1	1888	0	10	33	29	3	113.5	1888	0	10	270	265	4	80.5
la36_225	1268	16	1273.8	0.46	3	40	21	1	40.7	1269.4	0.11	1	214	198	1	35
la37_225	1397	12	1397	0	10	32	22	1	37.1	1397	0	10	195	186	1	33.4
la38_225	1196	180	1203.1	0.59	3	158	22	1	34.7	1202.2	0.52	2	305	206	1	31.5
la39_225	1233	11	1234.5	0.12	9	42	22	1	40.6	1233.6	0.05	8	233	212	1	34.1
la40_225	1222	24	1224	0.16	4	60	22	1	43.6	1222.3	0.02	9	250	216	1	39.2
Avg CPU(s)		171				45							104			
Avg dev in %		0				0.062							0.046			
Magnitude		40				27							30			
%OPT		100				87							87.75			

As can be stressed from the results exposed in Table 2, considering patterns in exact approaches can have different outcomes. The purpose of this first experiment was to evaluate if optimal solutions remain accessible when all the extracted patterns are imposed in solutions. Some cases show that optimal solutions become unreachable with GP imposed. However, the average deviation shows that only a 0.062% difference can be observed when considering 10k iterations, whereas it goes even further down when the number of iterations depends on the size of the problem (0.046%). Considering all replications 72.5% instances are always solved optimally (i.e. situations where avgS=Opt). These results demonstrate that, on average, good quality patterns are detected and that the search space covered is sufficient in 87.75% of the time to reach an optimal solution. On average, the use of all the patterns also drastically increases the convergence speed on instances with a computational time larger than 500 seconds while being close to the optimal solutions when not reaching it.

It can be observed that the number of patterns is not depending on the size of instances, but on their structures. As it can be easily stated from results, some problems are easier than others while having the same size. In some cases it is easier to detect valuable patterns, in others it is more complicated to extract patterns whose presence are statistically significant in solution quality. However it can be mentioned that if all the SLS give the same solution value (i.e. $\mu = 0 \wedge \sigma = 0$), no pattern will be extracted, like for example la10.

In this dataset, instance la27 is a good example of achievable results when considering knowledge in constraint programming, with a speedup factor of 3.5 while keeping accessibility of optimal solution in 9 runs over 10. If the question of accessibility having all patterns GP enforced is addressed in this section, in the next section the objective is to favor good patterns and not make them compulsory.

5.3. Improved local search assessment

In this section, the use of knowledge in local searches is addressed through two datasets. The first one is the same as in section 5.2, and the second one is related to RCPSP. First, Table 3 shows the comparison of the results between the approaches with/without learning on JSP instances. **avgS** consists in the average values obtained with 10 replications of the 300 stochastic local searches. **min** and **MAX** columns provide insight about the best and worst values obtained during the search process. **CPU(s)** refers to the average cumulative computation time, which aggregates population generations, and learning phase when considering the local search relying on GP. Secondly, Table 4 presents results when considering a dynamic number of iterations, based on the size of the problem.

An example of a local search descent is given in Figure 6 with the number of iterations in abscissa. In this figure, the dashed line corresponds to a descent performed with the classical local search whereas the solid line exposes the pattern-based local search's behavior.

As can be observed from Tables 3 and 4, better results can be achieved with a local search integrating knowledge for JSP. It should be mentioned that results are computed using the same experimental conditions (replication seeds are equal when building P'). As the results concern only the behavior of the local searches, outside any metaheuristic, it is not surprising that optimal solutions are not always reached. However, there is a clear difference between the obtained solutions. When considering 10k iterations, the average deviation for the LS with patterns is 5.27% and 6.88% when no knowledge is considered. A Wilcoxon signed-rank test shows that the difference between results is significant (i.e., p-

Table 3: Performance evaluation of 10k local search iterations with and without patterns GP consideration on JSP instances

Instance	Opt	SLS without GP - 10k					SLS With GP - 10k				
		avgS	dev%	CPU(s)	min	MAX	avgS	dev%	CPU(s)	min	MAX
la01	666	674.89	1.33	7	**666**	739	**669.67**	0.55	8	**666**	777
la02	655	678.10	3.53	8	**655**	753	**673.30**	2.79	8	**655**	749
la03	597	634.23	6.24	7	**597**	714	**625.80**	4.82	7	**597**	714
la04	590	609.77	3.35	6	**590**	731	**604.20**	2.41	6	**590**	700
la05	593	**593**	0	6	**593**	593	**593**	0	6	**593**	593
la06	926	**926**	0	8	**926**	926	**926**	0	8	**926**	926
la07	890	892.04	0.23	8	**890**	952	**890.71**	0.08	8	**890**	946
la08	863	863.53	0.06	8	**863**	936	**863.15**	0.02	8	**863**	923
la09	951	951.00	0	8	**951**	951	**951**	0	8	**951**	951
la10	958	**958**	0	8	**958**	958	**958**	0	8	**958**	958
la11	1222	**1222**	0	10	**1222**	1222	**1222**	0	10	**1222**	1222
la12	1039	**1039**	0	10	**1039**	1049	**1039**	0	10	**1039**	1049
la13	1150	**1150**	0	10	**1150**	1151	**1150**	0	10	**1150**	1151
la14	1292	**1292**	0	11	**1292**	1292	**1292**	0	11	**1292**	1292
la15	1207	1211.16	0.34	10	**1207**	1288	**1207.66**	0.05	10	**1207**	1277
la16	945	1011.57	7.04	10	946	1127	**1001.97**	6.03	10	**945**	1113
la17	784	821.18	4.74	10	**784**	1010	**804.83**	2.66	10	**784**	1021
la18	848	909.20	7.22	10	**848**	1065	**891.70**	5.15	10	**848**	1029
la19	842	916.56	8.85	10	**842**	1079	**898.46**	6.71	10	**842**	1018
la20	902	983.06	8.99	10	907	1110	**959.93**	6.42	10	**902**	1089
la21	1046	1166.70	11.54	14	**1065**	1332	**1147.38**	9.69	14	**1065**	1315
la22	927	1059.97	14.34	14	951	1237	**1025.13**	10.59	14	**941**	1165
la23	1032	1095.72	6.17	13	**1032**	1282	**1072.82**	3.96	14	**1032**	1217
la24	935	1046.20	11.89	13	959	1173	**1023.63**	9.48	13	**951**	1197
la25	977	1105.51	13.15	13	1006	1289	**1072.26**	9.75	13	**998**	1243
la26	1218	1341.97	10.18	17	1247	1524	**1309.92**	7.55	17	**1224**	1484
la27	1235	1399.33	13.31	17	1288	1561	**1374.85**	11.32	17	**1274**	1517
la28	1216	1364.64	12.22	17	1255	1524	**1335.24**	9.81	17	**1251**	1518
la29	1152	1354.13	17.55	17	1248	1505	**1323.59**	14.89	17	**1226**	1468
la30	1355	1471.98	8.63	17	**1355**	1650	**1429.78**	5.52	18	**1355**	1581
la31	1784	1810.19	1.468	25	**1784**	1973	**1793.6**	0.538	26	**1784**	1936
la32	1850	1894.10	2.384	25	**1850**	2137	**1871.71**	1.173	26	**1850**	2037
la33	1719	1758.52	2.30	25	**1719**	1903	**1739.66**	1.202	26	**1719**	1899
la34	1721	1816.54	5.55	25	**1723**	1970	**1781.21**	3.498	256	**1721**	1922
la35	1888	1970.78	4.38	26	**1888**	2212	**1921.55**	1.777	26	**1888**	2105
la36	1268	1475.31	16.35	19	1325	1729	**1423.9**	12.29	20	**1319**	1632
la37	1397	1628.00	16.54	19	1479	1940	**1585.43**	13.49	20	**1459**	1836
la38	1196	1421.92	18.89	19	1288	1629	**1405.14**	17.49	20	**1256**	1622
la39	1233	1464.57	18.78	20	**1287**	1711	**1418.35**	15.03	20	**1287**	1647
la40	1222	1438.08	17.68	19	1291	1683	**1395.05**	14.16	20	**1263**	1591
Average on dataset:			6.88	14				**5.27**	14		

value $\leq$ 0,0001). In these experiments, the best found solutions are also better using the learning process. For all instances, better **min** (or equal) solutions are obtained when patterns are considered during the local search process. Also, as can be stressed by the $CPU(s)$ column, the average total computation time

Table 4: Performance evaluation of local searches with and without patterns GP consideration on JSP instances - dynamic iterations case

Instance	Opt	LS without GP - dyn					LS With GP - dyn				
		avgS	dev%	CPU(s)	min	MAX	avgS	dev%	CPU(s)	min	MAX
la01	666	681.90	2.39	2	**666**	777	**676.09**	1.52	2	**666**	797
la02	655	692.95	5.79	3	**655**	811	**683.06**	4.28	2	**655**	785
la03	597	645.91	8.19	2	**597**	754	**636.81**	6.67	2	**597**	757
la04	590	622.78	5.56	2	**590**	782	**612.92**	3.88	2	**590**	781
la05	593	593.03	0.01	1	**593**	622	**593.02**	0.004	1	**593**	622
la06	926	**926**	0	4	**926**	926	**926**	0	4	**926**	926
la07	890	893.81	0.43	6	**890**	979	**891.45**	0.16	6	**890**	997
la08	863	863.90	0.10	5	**863**	959	**863.28**	0.03	5	**863**	937
la09	951	**951.01**	0.001	4	**951**	978	**951.01**	0.001	4	**951**	978
la10	958	**958**	0	4	**958**	958	**958**	0	4	**958**	958
la11	1222	**1222**	0	8	**1222**	1222	**1222**	0	8	**1222**	1222
la12	1039	**1039**	0	8	**1039**	1050	**1039**	0	8	**1039**	1050
la13	1150	**1150**	0	8	**1150**	1151	**1150**	0	8	**1150**	1151
la14	1292	**1292**	0	8	**1292**	1292	**1292**	0	8	**1292**	1292
la15	1207	1208.57	0.13	12	**1207**	1281	**1207.49**	0.04	14	**1207**	1280
la16	945	1006.69	6.53	14	**945**	1127	**997.33**	5.54	14	**945**	1109
la17	784	812.99	3.70	15	**784**	1010	**800.73**	2.13	16	**784**	956
la18	848	901.27	6.28	15	**848**	1065	**885.68**	4.44	15	**848**	1030
la19	842	908.23	7.87	16	**842**	1069	**890.20**	5.72	15	**842**	1008
la20	902	977.08	8.32	15	**902**	1110	**954.75**	5.85	14	**902**	1136
la21	1046	1122.81	7.34	56	1055	1280	**1108.69**	5.99	56	**1046**	1257
la22	927	1004.30	8.34	56	**927**	1164	**975.67**	5.25	57	**927**	1124
la23	1032	1052.19	1.96	50	**1032**	1200	**1040.88**	0.86	53	**1032**	1172
la24	935	1004.03	7.38	55	**941**	1144	**985.04**	5.35	55	**941**	1111
la25	977	1063.97	8.90	52	984	1207	**1035.42**	5.98	52	**978**	1178
la26	1218	1252.86	2.86	135	**1218**	1388	**1233.62**	1.28	140	**1218**	1342
la27	1235	1319.14	6.81	138	1255	1473	**1296.49**	4.98	140	**1244**	1439
la28	1216	1282.02	5.43	139	**1216**	1438	**1257.29**	3.40	140	**1216**	1398
la29	1152	1270.84	10.32	145	1180	1417	**1240.94**	7.72	144	**1164**	1371
la30	1355	1377.11	1.63	127	**1355**	1520	**1359.65**	0.34	138	**1355**	1456
la31	1784	1784.04	0.002	205	**1784**	1813	**1784.04**	0.002	205	**1784**	1813
la32	1850	1850	0	207	**1850**	1850	**1850**	0	207	**1850**	1850
la33	1719	1719.03	0.002	212	**1719**	1736	**1719.01**	0.001	212	**1719**	1736
la34	1721	1721.02	0.001	247	**1721**	1732	**1721.01**	0.001	250	**1721**	1729
la35	1888	1888.25	0.01	230	**1888**	1919	**1888.01**	0.001	236	**1888**	1914
la36	1268	1363.338	7.52	168	1281	1568	**1329.49**	4.85	174	**1277**	1489
la37	1397	1523.405	9.05	175	1410	1711	**1481.09**	6.02	176	**1409**	1634
la38	1196	1320.51	10.41	198	1215	1495	**1299.17**	8.63	194	**1207**	1448
la39	1233	1348.08	9.33	195	1249	1550	**1303.89**	5.75	193	**1243**	1495
la40	1222	1322.769	8.25	195	1233	1528	**1291.57**	5.69	201	**1233**	1429
Average on dataset			4.02	78				**2.81**	79		

does not differ significantly between the two approaches, as the learning phase does not require large computation times, and is hence diluted in the whole process. Extended computational experiments have been conducted with dynamic number of iterations. Results show that there is still a clear difference of

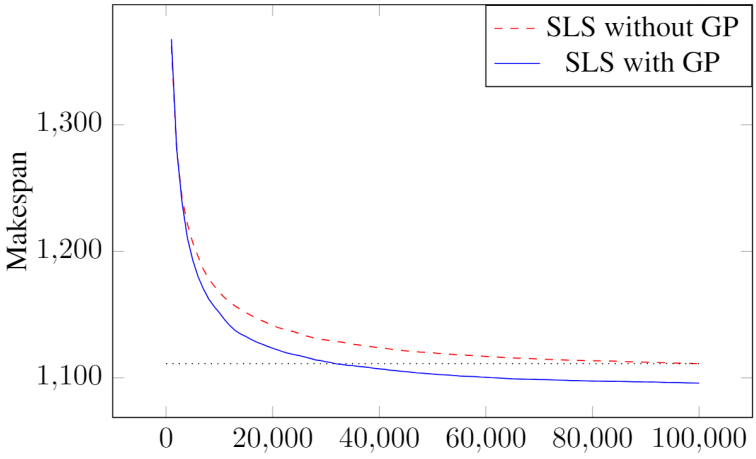

Fig. 6: Example of the average convergence of SLS for the instance La21 over 100k iterations: in dashed line when no pattern is considered, and in solid line when the patterns GP are used.

behaviour between the two types of local searches: the one embedding the learning module produces better results in average (2.81% deviation vs 4.02% when no knowledge is considered).

If some results with CP were showing that some exact solutions can be missed when considering the patterns GP imposed, as discussed in section 4.2.3, the optimal solution can still be found. For example, the fifth replication for instance la16 in the dynamic case is valued 981 with all patterns considered in CP. However, the best solution found is valued 945(optimal) with the local search process with the same set of pattern GP.

Finally, in Tables 5 and 6, results on RCPSP instances are shown with the static number of iterations in Table 5 and with the dynamic number of iterations in Table 6. To the best of our knowledge, solutions are not all demonstrated optimal for this dataset. Hence, Lower bounds (LB) and Upper bounds (UB) are considered in the table.

The computation times are equivalent for SLS with and without GP. As outlined in Tables 5 and 6 better results are obtained when considering patterns in GP. It results in a significant quality improvement; and the best solutions are more often found with the pattern mining module. However, these results should be put into perspective. Indeed, if the difference between results when considering 10k iterations remains significant following the Wilcoxon signed-rank test (p-value < 0.05), it is not the same for the dynamic number of iterations. In this last case, results are not significantly different when comparing LS with and without GP. Further investigations should be conducted to outline the underlying reasons, but one can suppose that more instances should be used for a better comparison.

Table 5: Performance evaluation of local searches with and without patterns GP consideration on RCPSP instances. with 10k iterations

Instance	LB	UB	Local Search Without GP				Local Search With GP				CPU(s)
			AVG_S	dev%	MIN	MAX	AVG_S	dev%	MIN	MAX	
J90_1_1	73	73	78.57	7.62	**74**	85	**76.70**	5.07	**74**	90	1863
J90_11_1	86	86	86.02	0.03	**86**	100	**86.01**	0.01	**86**	129	909
J90_21_1	109	110	122.35	12.25	115	136	**121.83**	11.77	**114**	137	2667
J90_31_1	79	79	79.03	0.04	**79**	85	**79.01**	0.01	**79**	87	3120
J90_41_1	128	142	154.20	20.47	147	167	**153.17**	19.67	**146**	166	3426
J120_1_1	105	105	114.56	9.11	**107**	125	**112.74**	7.37	**107**	128	2277
J120_11_1	156	173	192.56	23.44	**183**	204	**192.34**	23.29	**183**	208	4827
J120_21_1	114	114	124.08	8.84	116	142	**120.26**	5.49	**115**	135	1824
J120_31_1	180	198	220.88	22.71	211	233	**220.11**	22.28	**210**	235	4815
J120_41_1	127	127	135.12	6.39	**127**	144	**131.86**	3.82	**127**	147	1821
Average deviation in %					11.09				**9.88**		
Nb better solutions					6				**10**		

Table 6: Performance evaluation of local searches with and without patterns GP consideration on RCPSP instances with the dynamic number of iterations

Instance	LB	UB	Local Search Without GP				Local Search With GP				CPU(s)
			AVG_S	dev%	MIN	MAX	AVG_S	dev%	MIN	MAX	
J90_1_1	73	73	78.67	7.77	74	90	**76.87**	5.30	**73**	90	1572
J90_11_1	86	86	**86.03**	0.03	**86**	93	**86.03**	0.03	**86**	96	981
J90_21_1	109	110	121.80	11.74	**114**	132	**121.40**	11.38	**114**	137	2787
J90_31_1	79	79	79.06	0.08	**79**	95	**79.03**	0.04	**79**	94	3936
J90_41_1	128	142	153.73	20.10	146	165	**153.08**	19.59	**145**	165	3816
J120_1_1	105	105	113.38	7.98	**107**	121	**111.31**	6.01	**107**	129	2769
J120_11_1	156	173	**189.00**	21.15	**179**	203	189.25	21.31	180	204	12579
J120_21_1	114	114	122.05	7.06	**115**	136	**118.76**	4.18	**115**	135	4248
J120_31_1	180	198	216.13	20.07	**206**	229	**215.94**	19.97	**206**	232	15345
J120_41_1	127	127	134.28	5.73	**127**	143	**130.82**	3.01	**127**	144	3954
Average deviation in %					10.17				**9.08**		
Nb better solutions					8				**9**		

6. Conclusion

In this paper, the use of a machine learning approach is evaluated in the context of local searches for scheduling problems. Two well-known problems from the scheduling literature are addressed: the Job-shop Scheduling Problem and the Resource Constrained Project Scheduling Problem. Solutions in scheduling problems can be represented using an ordered list of operations from which knowledge is extracted under the form of patterns that represent succession relations between tasks. The relevance of the patterns is first evaluated in a Constraint Programming solver (IBM Cplex Optimizer), and then

Local Searches relying on the acquired knowledge are designed. Results on the exact solver show that a speedup factor can be achieved when the set of obtained patterns is enforced in every solution, even though the accessibility of some optimal solutions can be lost. The average performance of Local Searches embedding pattern knowledge demonstrates the validity of the approach for scheduling problems. Moreover, it is important to show that the best solutions are also found with the pattern mining process.

In this research work, no scheduling problem with resources flexibility, for which the duration of operations depends on the chosen resource to process it, or tasks preemption is considered. In such problems, pattern detection in the ordered list of operations would depend on the assignment of operations to specific resources for problems with flexibility. In problems with task preemption, a task can be processed during non-adjacent periods, and thus, the global order used to extract the solution structures would not be relevant. It would be interesting to test the provided method on these types of problem.

If all results tend to demonstrate that using unsupervised learning can improve the efficiency of local searches, several potential improvements can be mentioned.

As it is presented in this paper, the consideration of patterns is binary: suppression of a pattern is prohibited, or it is not. Another way to consider patterns could be to apply a probability to allow a pattern to be broken, depending on its quality for example.

Several other research prospects can be mentioned. First, the integration of pattern mining-based Stochastic Local Searches (SLS) in metaheuristics should be addressed. The population-based metaheuristics using generic SLS seem more appropriate to integrate the proposed module. However, it is possible to adapt the pattern extraction to benefit single-solution based metaheuristics.

For the integration of SLS in metaheuristics, the capacity of a metaheuristic to access solutions with prohibited patterns highly depends on the perturbation process of the metaheuristic. An idea to respect the accessibility to all solutions could be to favor the removal of patterns in GP during the perturbation process.

Addressing large scale instances with constraint programming using the extracted patterns could also be a promising direction, as interesting speedup factors can be observed. If the optimal solution can be missed when considering all patterns, good quality solutions could be obtained, thus defining new upper bounds that can be used in an iterative approach.

Finally, the provided approach can be adapted to fit other optimization problems. With some adjustments to take into account distances between edges in Routing Problems solutions, knowledge could also be extracted in the form of patterns. Addressing integrated scheduling and routing problems could also be a promising research field.

References

Ali, M., Pant, M., Abraham, A., 2013. Unconventional initialization methods for differential evolution. Applied Mathematics and Computation 219, 4474 – 4494.

Aytug, H., Bhattacharyya, S., Koehler, G.J., Snowdon, J.L., 1994. A review of machine learning in scheduling. IEEE Transactions on Engineering Management 41, 165–171.

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. Operations-Research-Spektrum 17, 87–92.

Blazewicz, J., Domschke, W., Pesch, E., 1996. The job shop scheduling problem: Conventional and new solution techniques.

European Journal of Operational Research , 1–33.
Bourreau, E., Gondran, M., Lacomme, P., Vinot, M., 2019. De la programmation linéaire à la programmation par contraintes.
Bürgy, R., 2017. A neighborhood for complex job shop scheduling problems with regular objectives. Journal of Scheduling 20, 391–422.
Carlier, J., 1984. Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité. Ph.D. thesis. Université Paris VI. Paris.
Chiu, C., Yih, Y., 1995. A learning-based methodology for dynamic scheduling in distributed manufacturing systems. International Journal of Production Research 33, 3217–3232.
Corne, D., Dhaenens, C., Jourdan, L., 2012. Synergies between operations research and data mining: The emerging use of multi-objective approaches. European Journal of Operational Research 221, 469–479.
Feng, L., Ong, Y., Jiang, S., Gupta, A., 2017. Autoencoding evolutionary search with learning across heterogeneous problems. IEEE Transactions on Evolutionary Computation 21, 760–772.
Fleury, G., 1993. Méthodes stochastiques et déterministes pour les problèmes NP-difficiles. Ph.D. thesis. Clermont-Ferrand 2.
Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, H.B., 2017. A survey of itemset mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7, 760–772.
Gholami, P., Hafezalkotob, A., 2018. Maintenance scheduling using data mining techniques and time series models. International Journal of Management Science and Engineering Management 13, 100–107.
Guerine, M., Rosseti, I., Plastino, A., 2016. Extending the hybridization of metaheuristics with data mining: Dealing with sequences. Intelligent Data Analysis 20, 1133 – 1156.
Hoos, H.H., Stützle, T., 2004. Stochastic local search: Foundations and applications. Elsevier.
Jain, V., Jain, P., Premalatha, S., Baskar, N., 2012. Implementation of supervised statistical data mining algorithm for single machine scheduling. Journal of Advances in Management Research .
Karthikeyan, S., Asokan, P., Nickolas, S., Page, T., 2012. Solving flexible job-shop scheduling problem using hybrid particle swarm optimisation algorithm and data mining. International journal of manufacturing technology and management 26, 81–103.
Kemmoé-Tchomté, S., Lamy, D., Tchernev, N., 2017. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. Engineering Applications of Artificial Intelligence 62, 80–95.
Kolisch, R., Sprecher, A., 1997. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. European journal of operational research 96, 205–216.
Koonce, D., Tsai, S.C., 2000. Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. Computers & Industrial Engineering 38, 361–374.
Kumar, S., Rao, C., 2009. Application of ant colony, genetic algorithm and data mining-based techniques for scheduling. Robotics and Computer-Integrated Manufacturing 25, 901–908.
Lawrence, S., 1984. Supplement to resource constrained project scheduling: An experimental investigation of heuristic techniques.
Lee, C.Y., Piramuthu, S., Tsai, Y.K., 1997. Job shop scheduling with a genetic algorithm and machine learning. International Journal of production research 35, 1171–1191.
Lin, L., Gen, M., 2018. Hybrid evolutionary optimisation with learning for production scheduling: state-of-the-art survey on algorithms and applications. International Journal of Production Research 56, 193–223.
Mashhadi, H.R., Shanechi, H.M., Lucas, C., 2003. A new genetic algorithm with lamarckian individual learning for generation scheduling. IEEE Transactions on Power Systems 18, 1181–1186.
Meisel, S., Mattfeld, D., 2010. Synergies of operations research and data mining. European Journal of Operational Research 206, 1–10.
Olafsson, S., Li, X., Wu, S., 2008. Operations research and data mining. European Journal of Operational Research 187, 1429–1448.
Papon, P.A., 2016. Extraction Optimisée de Règles d'Association Positives et Négatives Intéressantes. Ph.D. thesis. Université Clermont-Auvergne.
Park, J., Kim, K.Y., 2017. Meta-modeling using generalized regression neural network and particle swarm optimization. Applied Soft Computing 51, 354 – 369.
Priore, P., De La Fuente, D., Gomez, A., Puente, J., 2001. A review of machine learning in dynamic scheduling of flexible manufacturing systems. Ai Edam 15, 251–263.

Priore, P., de la Fuente, D., Puente, J., Parreño, J., 2006. A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems. Engineering Applications of Artificial Intelligence 19, 247–255.

Priore, P., Parreno, J., Pino, R., Gomez, A., Puente, J., 2010. Learning-based scheduling of flexible manufacturing systems using support vector machines. Applied Artificial Intelligence 24, 194–209.

Shahzad, A., Mebarki, N., 2012. Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. Engineering Applications of Artificial Intelligence 25, 1173–1181.

Sun, C., Jin, Y., Zeng, J., Yu, Y., 2015. A two-layer surrogate-assisted particle swarm optimization algorithm. Soft Comput. 19, 1461–1475.

Talbi, E.G., 2016. Combining metaheuristics with mathematical programming, constraint programming and machine learning. Annals of Operations Research 240, 171–215.

Thevenin, S., Zufferey, N., 2019. Learning variable neighborhood search for a scheduling problem with time windows and rejections. Discrete Applied Mathematics 261, 344 – 353.

Wang, C., Rong, G., Weng, W., Feng, Y., 2015. Mining scheduling knowledge for job shop scheduling problem. IFAC-PapersOnLine 48, 800–805.

Wang, X., Tang, L., 2017. A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem. Computers & Operations Research 79, 60–77.

Zare, H.K., Fakhrzad, M.B., 2011. Solving flexible flow-shop problem with a hybrid genetic algorithm and data mining: A fuzzy approach. Expert systems with applications 38, 7609–7615.

Zhang, J., Zhan, Z., Lin, Y., Chen, N., Gong, Y., Zhong, J., Chung, H.S.H., Li, Y., Shi, Y., 2011. Evolutionary computation meets machine learning: A survey. IEEE Computational Intelligence Magazine , 68–75.