Automatic Algorithm Design for Hybrid Flowshop Scheduling Problems

Pedro Alfaro-Fernández[a,*], Rubén Ruiz[a], Federico Pagnozzi[b], Thomas Stützle[b]

[a] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*
[b] *IRIDIA, Université Libre de Bruxelles (ULB), CP 194/6, Av. F. Roosevelt 50, B-1050 Brussels, Belgium*

Abstract

Industrial production scheduling problems are challenges that researchers have been trying to solve for decades. Many practical scheduling problems such as the hybrid flowshop are $\mathcal{NP}$-hard. As a result, researchers resort to metaheuristics to obtain effective and efficient solutions. The traditional design process of metaheuristics is mainly manual, often metaphor-based, biased by previous experience and prone to producing overly tailored methods that only work well on the tested problems and objectives. In this paper, we use an Automatic Algorithm Design (AAD) methodology to eliminate these limitations. AAD is capable of composing algorithms from components with minimal human intervention. We test the proposed AAD for three different optimization objectives in the hybrid flowshop. Comprehensive computational and statistical testing demonstrates that automatically designed algorithms outperform specifically tailored state-of-the-art methods for the tested objectives in most cases.

Keywords: Scheduling, Hybrid flowshop, Automatic algorithm configuration, Automatic algorithm design

*Corresponding author. Tel/Fax: +34 96 387 99 52
Email addresses: pedroaf@iti.upv.es (Pedro Alfaro-Fernández), rruiz@eio.upv.es (Rubén Ruiz), federico.pagnozzi@ulb.ac.be (Federico Pagnozzi), stuetzle@ulb.ac.be (Thomas Stützle)

1. Introduction

Many industries, in particular those related with manufacturing, face scheduling decisions in the daily management of their operations. Broadly speaking, scheduling entails the assignment of production tasks to manufacturing equipment in order to produce goods and services. Usually, equipment and/or machines are limited. Proper scheduling involves sophisticated algorithms that seek to maximize the utilization of these scarce resources in the form of the optimization of one or more objectives. Academic scheduling is a very well developed field with literally thousands of published papers since the seminal work of Johnson (1954).

More formally, a scheduling problem consists of assigning and sequencing n jobs to m machines, The simplest setting is a problem with a single machine. When there are multiple machines,which might be disposed in series to form what is referred to as a flowshop scheduling problem or in parallel, resulting in a parallel machines scheduling problem. Most real production floors have a combination of these two last problems in which there is a set of production stages and at each stage there is more than one machine in parallel. Each product to be manufactured goes from one stage to another. This problem is known as the Hybrid Flowshop Scheduling (HFS) problem, which is well-known to be $\mathcal{NP}$-hard (Gupta, 1988) for the makespan objective. Based on the scheduling literature and complexity hierarchies between objectives and scheduling problems (Pinedo, 2016; Framiñan et al. 2014) we can assert that the problems considered here are also $\mathcal{NP}$-hard.

As a matter of fact, HFS problems are so difficult that the best existing exact approaches are only able to solve problems of a very small size (Ruiz and Vázquez-Rodríguez, 2010). Consequently, researchers often resort to approximate methods: heuristics and/or metaheuristics. While heuristics are problem dependent, metaheuristics propose general mechanisms that are able to deal with almost any optimization problem. When properly engineered, metaheuristic algorithms often perform very well and offer a good balance between speed, solution quality, flexibility and robustness (Hoos and Stützle, 2004; Talbi, 2009).

All these benefits, however, come at a cost. Metaheuristics require careful design, instantiation and parameter calibration or tuning. If these aspects are neglected, most of the aforementioned benefits are lost. Consequently, the literature of metaheuristics focuses a considerable amount of its efforts on the aspects of algorithm design, parameter calibration and related issues. This, in

turn, generates myriad problems that have been clearly acknowledged in the literature. A particular problem is the overuse of so called nature-inspired metaheuristics, duly discussed by Sörensen (2015). As a result, nowadays, a portion of the metaheuristics community resorts to the development of debatable metaheuristics in order to cope with the algorithm design process. On the other hand, when dealing with the aspect of metaheuristic instantiation and parameter calibration, authors often rely on trial-and-error procedures and a sort of artisanship. It has to be noted that most of the excellent results obtained for many different optimization problems by the metaheuristics community are due to the extensive experience and intuition of researchers. However, it is preferable to rely more on sound scientific procedures for the design and calibration of metaheuristic algorithms. Furthermore, many state-of-the-art metaheuristics employ components or operators that are heavily tailored to the specific problem in hand. While this is not bad per se, it goes against the principle of generality present in metaheuristics.

As with most optimization problems, for HFS problems several highly performing metaheuristic algorithms have been proposed to date. Similar to other fields, these existing metaheuristics suffer from the aforementioned shortcomings; nature-inspired and complex methodologies, problem specific operators and trial-and-error algorithm instantiation, etc. Surprisingly, and aAs will be highlighted in the next section, for the same HFS problem, a change in the objective function studied results in wildly different metaheuristic techniques. As a result, the main benefit of metaheuristics, being the generality and ease of instantiation, is lost along the way through a process of manual design and fine tuning. There have been some attempts at automating, to some extent, this instantiation. Some of these techniques have been referred to as hyperheuristics (Cowling et al. 2001) Hyperheuristics often incorporate some rules to help in the instantiation.

An alternative to manual design and calibration of metaheuristics are Automatic Algorithm Configuration (AAC) techniques, one example being the irace method (López-Ibáñez et al., 2016). AAC can speed up and automate the process of instantiating and tuning the different parameters of a given metaheuristic template. However, AAC can be used not only to explore the parameters of different algorithm components but also to explore the usage of different alternative algorithm components, if alternative choices are represented as parameters. If an AAC technique is combined with an algorithmic framework that offers different potential metaheuristic templates and the components of the different algorithms, one may generate as a result

an algorithm consisting of a best combination of algorithmic components while at the same time fine-tuning all numerical parameters. We refer to such a combination of AAC techniques with algorithm frameworks as Automatic Algorithm Design (AAD). Some authors see AAD as a form of advanced hyperheuristics (Burke et al., 2013) as hyperheuristics might range from simple methods to choose heuristics or rules, to any algorithm that works with rules or heuristics and decides which one (or combination of them) to use. In a nutshell, hyperheuristics are very broad and can include anything from machine learning to random or weighted selection of rules.

HFS problems are very common in manufacturing industries. However, and as will be pointed out in the following sections, rReal scheduling deals with different objective functions. At the same time, constantly changing environment and markets result in varying problems within the same company over a relatively short period of time. Therefore, highly problem-specific and tailored metaheuristics are rarely used in practice. Instead, general methods and an automated way of tuning and calibrating metaheuristics is a much more promising approach.

The objective of this paper is to propose and exploit an AAD methodology to automatically obtain highly performing algorithms for HFS problems with varying optimization objectives, that is, without any manual tuning and knowledge that is applicable to only a single problem variant. Of course, the important question here is how these automatically generated algorithms will compare to manually tuned methods. An important result and contribution of this paper will be precisely to show that automatically generated algorithms are, with some exceptions, state-of-the-art for the different objectives considered.

The remainder of this paper is organized as follows: Section 2 formally describes the HFS problem and summarizes the related literature. Section 3 details the current process and issues in traditional algorithm design and calibration. Section 4 describes the proposed Automatic Algorithm Design methodology. Section 5 contains the details and the results of the extensive computational experimentation carried out in this paper. Finally, Section 6 concludes the paper and proposes future lines of research.

2. Problem description and literature review

In the regular flowshop problem one has to sequence a set N of n jobs each one visiting all the m machines from the set of machines M. Machines

are disposed in series and each job is processed first on machine 1, then on machine 2 and so on until machine m. Each job j, $j = 1, 2, \ldots, n$ needs a given amount of time to be processed at each machine i, $i = 1, 2, \ldots, m$. This time is referred to as processing time and denoted as p_{ij}. Different from flowshops, in parallel machine scheduling problems, the different machines are disposed in parallel. Therefore, jobs have to be processed by one out of the m parallel machines. HFS problems were initially studied about 20 years later than the first papers on regular flowshops (Gupta, J. N. D. and Stafford, 2006) and combine flowshops and parallel machines. Hence, in an HFS, instead of m machines in series, there are m stages, where each stage i has $m_i \geq 1$ parallel machines were $\exists i$ such that $m_i > 1$ (Framiñan et al., 2014).

The most commonly studied objective in the HFS literature is makespan minimization (Ruiz and Vázquez-Rodríguez, 2010). If we denote as C_j the completion time of job j at stage m (i.e., when the job is completed and ready for shipment), we have that makespan or $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$. As already mentioned, the HFS with $C_{\max}$ objective is strongly $\mathcal{NP}$-hard, which is the case even for the simplest possible setting where $m = 2$, $m_1 = 1$ and $m_2 = 2$ (Gupta, 1988). Up to 60% of the existing work on the HFS revolves around the $C_{\max}$ objective (Ruiz and Vázquez-Rodríguez, 2010). Yet, this is far from being the most realistic criterion for industrial companies. Another, much less studied problem is the so called total flowtime or $TFT = \sum_{j=1}^{n} C_j$. Under this TFT objective, work-in-progress is adequately dealt with, which means less on-going inventory inside the shop. Note that we do not believe that makespan is not important but rather we stress that other more realistic objectives should not be overlooked. One such example of a more realistic objective is the total earliness and tardiness minimization. Jobs have to be delivered by a certain due date, referred to as d_j. Jobs delivered beyond their due dates are said to be tardy as measured by the tardiness function $T_j = \max\{0, C_j - d_j\}$. Similarly, jobs completed before their due date are early by $E_j = \max\{0, d_j - C_j\}$. Ideally, jobs should be completed as close as possible to their due dates, avoiding tardiness, which translates into customer dissatisfaction, or earliness, which increases inventory holding costs. Adding weights to earliness and tardiness results in total weighted earliness and tardiness minimization or $TWET = \sum_{j=1}^{n} \left(w_j' E_j + w_j T_j\right)$ where w_j' and w_j denote the early and tardy weights of job j respectively. According to Ruiz and Vázquez-Rodríguez (2010), less than 1% of the existing literature on HFS studies this objective and most of the time weights are not considered. Furthermore, due dates are often periods of several hours (for example, serving

a job at a given day, but throughout the day). Recently, Pan et al. (2017) extended this concept to due date windows where a job is neither tardy nor early if completed within a time window $\{d_j^-, d_j^+\}$. This significantly more realistic function is denoted as $TWET^{dw} = \sum_{j=1}^{n} \left(w_j' E_j^{dw} + w_j T_j^{dw}\right)$ where $E_j^{dw} = \max\{0, d_j^- - C_j\}$ and $T_j^{dw} = \max\{0, C_j - d_j^+\}$. Using the three field $\alpha/\beta/\gamma$ notation of Graham et al. (1979) with the extension of Vignier et al. (1999), we study the hybrid flowshop with identical parallel machines denoted as $HFm, ((PM_{(k)}))_{k=1}^{m}//\gamma$, where $\gamma = C_{\max}, TFT, TWET^{dw}$.

The HFS has been surveyed periodically and we can refer the reader to the some of the previous and extensive reviews of Vignier et al. (1999); Ruiz and Vázquez-Rodríguez (2010);Linn and Zhang (1999); Wang (2005); Ribas et al. (2010). It has to be noted though, that according to these reviews, almost a third of the existing research deals with the specific case of two stages only. In what follows, we mostly review papers on the regular HFS without additional constraints and the aforementioned objectives.

Khalouli et al. (2010) proposed an ant colony optimization algorithm (ACO) to tackle the HFS with weighted earliness tardiness. The HFS with multiprocessor tasks (HFSMT) (a version where tasks may need more than one machine to be processed) is studied by Engin et al. (2011) who designed a genetic algorithm, and Singh and Mahapatra (2012) who implemented a particle swarm optimization algorithm (PSO); both papers studied the makespan objective. Another PSO for the regular HFS is that ofA particle swarm optimization algorithm (PSO) was implemented for the HFS by Liao et al. (2012) also for the makespan objective. Most authors studying makespan in the HFS literature use the set of small instances by Carlier and Néron (2000). Liao et al. (2012) added 10 new instances of slightly larger sizes to this benchmark. The same HFS problem was studied by Marichelvam et al. (2013) who proposed a bat algorithm (BA). Wang et al. (2013) designed an estimation of distribution algorithm (EDA). A shuffled frog-leaping algorithm (SFLA) was implemented by Xu et al. (2013). Chung and Liao (2013) introduced an immunoglobulin-based artificial immune system algorithm (IAIS). Bożejko et al. (2013) designed a very complex parallel tabu search (PTS) employing SSE2 (Streaming Single Instruction, Multiple Data Extensions 2) processor instruction sets.

Marichelvam et al. (2014a) proposed a discrete firefly algorithm (DFA) for a multi-objective version of the HFS with makespan and mean flowtime. A migrating birds optimization algorithm (MBO) was designed by Pan and

Dong (2014) for the HFS with total flowtime minimization, and a complete comparison was shown which also included some promising algorithms for similar problems. The authors claimed the MBO to be state-of-the-art for this problem.

For makespan minimization in the HFS, Li et al. (2014) implemented a hybrid variable neighborhood search (HVNS) and Marichelvam et al. (2014b) introduced a cuckoo search (CS). Two different disctrete artificial bee colony (DABC) algorithms were proposed for the HFS with makespan criterion. The former was presented by Pan et al. (2014) and the latter by Cui and Gu (2015); both claimed to be state-of-the-art for this problem. In particular, the DABC of Pan et al. (2014) includes functions and operators that are specifically tailored for the makespan objective. The result is a very efficient and effective algorithm but that is only applicable to this objective.

As mentioned, Pan et al. (2017) presented, to the best of our knowledge, the first and only work for the total earliness and tardiness with due date windows for the HFS. The authors proposed an iterated local search (ILS) and iterated greedy (IG) for the problem and carried out a comparison with various algorithms. The ILS based method, referred to as ILST, resulted in being state-of-the-art for the problem in their reported experiments.

As can be seen, there is a proliferation in the HFS literature of either nature-inspired methods, criticized by Sörensen (2015), but also of heavily tailored methods. We advocate the usage of general metaheuristics, which, as mentioned, can be easily adapted to different problems in practice.

3. Design and tuning of metaheuristics

When solving an optimization problem with metaheuristics, at least two tasks must be carried out: First, the metaheuristic algorithm must be created, usually as an instantiation of a given metaheuristic template. Second, once the method has been created, it has to be calibrated and/or tuned in order to set its operators and parameters. We now detail these two stages in the metaheuristic creation process:

3.1. Algorithm Design

Algorithm Design relies on expertise and knowledge of both metaheuristics and the specific problem to be solved. Usually, the first decision is which algorithm template to apply. For example, academic researchers often decide first that they want to develop a genetic algorithm, or a tabu search or any

other template for the studied problem. This is already a potential shortcoming in traditional algorithm design as the choice of template to instantiate usually depends on previous experience and preferences rather than actual data. Rarely do authors try radically different algorithm templates in academic research when developing and designing metaheuristics.

Furthermore, there are countless known templates of metaheuristic algorithms that can be instantiated for almost any problem. Some well known examples are: simulated annealing, tabu search, genetic algorithms, iterated greedy, ant colony optimization and many more. Actually, and as pointed out by Sörensen (2015), nowadays one could say that there are far too many templates. An interesting satirical reckoning of the ever increasing list of metaheuristics is available at http://conclave.cs.tsukuba.ac.jp/research/bestiary/.

Once the template has been selected, a very wide choice of decisions opens up. One has to decide on the solution representation, mutation operators, neighborhoods, the starting solution, acceptance criteria and a number of other operators and options. Basically, the possible combinations are enormous. Most regrettably, it is fairly common in the academic research to find that the quantity of tests that are performed at this stage are usually limited. Statements such as "in some preliminary experiments" are very common in practice. Most authors employ a small set of instances at this design stage and make decisions based more on trial and error than on actual sound science. It has to be noted that there are some excellent works where authors employ statistically sound methodologies at the design stage, most notably the Design of Experiments (DOE) approach (Montgomery, 2012). However, even in these cases, the number of tested combinations is limited due to the nature of the DOE itself. Let us remark that there are other modern approaches like Algorithm Selection Problems like Bożejko et al. (2018) where the authors rely on the statistical analysis of the performance of different algorithms to decide the best method for a specific instance based on instance characteristics.

Despite the potential aforementioned shortcomings, the current state-of-the-art in metaheuristics is extremely good. The reason is that new research is built upon previous work which has already made inroads into improving solutions Therefore, the accumulated body of research on a particular problem yields very effective results.

3.2. Algorithm Calibration

Most metaheuristic operators rely on a set of often numerical parameters. For example, in a Genetic Algorithm template, the mutation probability has to be set. Too high a mutation probability results in too random a search, while too small a mutation probability results in premature convergence. As a consequence, the performance of a metaheuristic often depends on a suitable choice of values for the different operators. Some metaheuristic templates can easily have 10+ parameters. Another way of looking at parameters is that different values allow metaheuristics to adapt to different problems or instances within a given problem (Talbi, 2009).

Choosing the best set of values for the parameters is a daunting task. Even though there is a large body of literature dealing with metaheuristic calibration (Bartz-Beielstein et al., 2010), a common practice is to set parameters manually using trial-and-error techniques. DOE is a powerful statistical procedure employed by some authors for testing, in a controlled way, a set of parameters that might affect a response variable. DOE frequently relies on the Analysis of Variance (ANOVA) technique to assess the statistical significance of a given parameter. A very powerful trait in DOE+ANOVA is that it is possible to study how different factors interact. Despite these advantages, DOE+ANOVA does not come without significant drawbacks: First, the number of treatments (experimental units) usually grows exponentially with the number of factors and levels studied within each factor. Second, some a priori knowledge is needed in order to reduce the number of experimental units. Third, DOE+ANOVA is far from automatic, requiring several iterations of statistical analysis and plot interpretation, etc. In a nutshell, DOE+ANOVA is far better than manual methods but it is fairly limited in the number of factors that one can study and it is time consuming and prone to error.

Alternatives to either manual or DOE+ANOVA calibration have been proposed in the literature in the form of Automatic Algorithm Configuration (AAC) methodologies. AACs are computational methods that are used to configure metaheuristics or other algorithms, with the objective of obtaining parameter settings that optimize performance. Some software packages capable of doing AAC have been proposed: irace (López-Ibáñez et al., 2016), ParamILS (Hutter et al., 2009), or SMAC (Hutter et al., 2011). Contrary to DOE+ANOVA, AAC is usually simpler, less a priori knowledge is needed and the amount of computation might be reduced. The cost of using AAC is loosing control of the calibration and the capability of studying interactions (as a general rule). Finally, unlike ANOVA, AAC does not study all parame-

ter/value combinations and the results generated are an approximation. This is not to say that ANOVA is "exact" in the sense that ANOVA only yields the best tested combination of parameter values, not the optimal one.

4. Automatic algorithm design for hybrid flowshop

4.1. Automated design of metaheuristic algorithms

The advent of AAC techniques also comes with a rather wide interpretation of parameters. In fact, AAC techniques can typically deal with categorical parameters, which can represent alternative choices for algorithm components or operators, and numerical, i.e., integer or real parameters. Thus, when AAC techniques are combined with appropriately designed, configurable metaheuristic frameworks, we may actually speak of an Automatic Algorithm Design (AAD). In a sense, AAD combines the two phases of algorithm design and calibration, making it automatic in the sense that it requires minimal user intervention. It can be applied without statistical knowledge, as the statistical tests and procedures are often carried out within the AAC tool used. However, AAC is mandatory for a realistic AAD as the number of components and parameters is massive to be dealt with DOE+ANOVA. One requisite of AAD is the isolation of the different algorithm components into individual entities. These entities must be able to work and interact with others without interdependencies among them, i.e., they must be an abstraction. The task of the AAD methodology is then to put together the best combination of components so as to create a new algorithm which performs well for the studied problem.

Several papers have already explored the possibility of using AAD methodologies to obtain highly-performing algorithms. KhudaBukhsh et al. (2016) worked on an automated method to build stochastic local search solvers from algorithmic components for the propositional satisfiability problem (SAT). López-Ibáñez and Stützle (2012) presented a framework for the automatic algorithm configuration of multiobjective ant colony optimization methods. Burke et al. (2012) approached the bin packing problem using grammatical evolution with local search heuristics. In Marmion et al. (2013) AAD was used to develop a hybrid stochastic local search procedure for the permutation flowshop problem with the weighted tardiness objective. Franzin and Stützle (2016) worked on the design of automatically generated simulated annealing algorithms for the quadratic assignment problem. The general consensus in

these papers is that, most of the time, the automatically built procedures are competitive with the manually constructed ones.

Previous work on AAD can be classified according to two main approaches: top-down and bottom-up (López-Ibáñez et al., 2017; Stützle and López-Ibáñez, 2019). In the top-down approach, there is usually a predefined algorithm template where different algorithms are obtained by alternative choices for a priori defined abstract procedures that are needed to instantiate the algorithm template. Therefore, the top-down approach induces a bias towards a type of algorithm as the template is given. Among the above mentioned examples, KhudaBukhsh et al. (2016); López-Ibáñez and Stützle (2012); Franzin and Stützle (2016) follow this approach. In the bottom-up approach, there is a set of rules to enforce the possible combinations without a discernible template. As a result, many more combinations are possible, including those resulting in methods never studied before. The bottom-up approach is more flexible and is less influenced by potential designer bias. An example for the latter is Marmion et al. (2013).

In both approaches, a part of the AAD is referred to as *flexible algorithm framework* in which the functions and parts of the algorithms are abstracted into a set of independent entities. By doing so, AAD methods are able to combine entities in multiple ways, potentially generating different algorithms. AAD automatically conceives specific algorithms as the sum of many entities or components put together in a specific order. However, not any order is feasible as the order matters. Thus we have another requisite, a set of rules to ensure feasibility and correction in the algorithms. In top-down approaches, correctness is typically ensured by the static, already ordered algorithm template. In bottom-up approaches, these rules are enforced often through *grammars* that guarantee that the different entities interact with syntactic and semantic correctness. Finally, since the goal of AAD is not to generate just any algorithm, but a very effective one, we require a system that enables a search for the good algorithms. This final part is carried out by the AAC technique. In the following sections we detail all the parts of the AAD framework that we have developed.

4.2. Automated generation of hybrid metaheuristics

Based on the initial ideas of a bottom-up approach to the design of hybrid metaheuristic algorithms, we have developed a flexible algorithm framework to support the automated design process. In fact, Marmion et al. (2013) use a set of entities that exploit many elements that were available

in the ParadisEO framework (Cahon et al., 2004). However, this led to a code that is rather complex to use. In our work, we employ a C++ flexible algorithm framework specifically designed for AAD, developed at the IRIDIA laboratory of the Université Libre de Bruxelles and refereed to as EMILI (short for Easily Modifiable Iterated Local Search Implementation). The EMILI framework puts a strong focus on the concepts of modularity and abstraction in modern software engineering , using these at a level appropriate to support automated algorithm design. In part, this translates into a re-use of components and supports problem-independent algorithm design. For example, an efficient local search operator usually includes many different components, such as a representation of the solution, a neighborhood, a way of calculating incremental moves, a pivoting rule and a termination criterion, etc. Most authors will add into efficient local search procedures all these components in an integrated way so as to speed up the search. This creates interdependencies between the components as, for example, the incremental move calculation depends on the problem constraints, objective function or neighborhood, etc. Hence, EMILI offers such concepts in generic ways and the algorithms we design, in the large part, use only generic algorithm components. However, if desired, in EMILI problem-specific components may also be integrated, as is done with some constructive heuristics as explained in Section 4.3. Overall this gives the EMILI framework a significant flexibility.

On the metaheuristic side, EMILI is based on a generalized view of hybrid metaheuristics as proposed by Marmion et al. (2013); López-Ibáñez et al. (2017). In essence, the metaheuristic part is based on a generalized template from which many different metaheuristics can be instantiated that manipulate a single incumbent solution. These metaheuristics comprise methods such as iterated or variable neighborhood search, iterated greedy or Tabu Search. The metaheuristics are instantiated starting from an iterated local search (ILS) template (Lourenço et al., 2010), as shown in Algorithm 1 by appropriate choices of alternative algorithm components. For example, by appropriate instantiation of the initial solution, perturbation, local search and acceptance criterion an ILS algorithm is obtained. For specific choices of the perturbation (e.g. a random move in some neighborhood), no application of a further local search (that is, setting SLS in line 6 to `none`) and setting the acceptance criterion to the Metropolis condition, a simulated annealing algorithm may be obtained. In addition, hybrids between the metaheuristics can be instantiated by allowing SLS to be instantiated as the main ILS loop (lines 4 to 8), leading to a possible recursive combination of metaheuristics. More details on this

Algorithm 1 High-level algorithmic outline of an ILS template.

Output The best solution found π^*,
π := Init();
π := SLS(π);
while ! termination criterion **do**
    π' := Perturbation(π);
    π' := SLS(π');
    π := AcceptanceCriterion(π, π');
end while
Return the best solution found in the search process

Algorithm 2 Snapshot of the grammar rules for deriving an iterative improvement algorithm. The derivation starts by applying the first rule for **<iterative_imp>** and replacing the non-terminals (delimited by angular parentheses) on the right side with the respective rules. The rule for deriving the initial solution (indicated by non-terminal symbol < *initial_sol* > is defined in Algorithm 3 below.

<iterative_imp> ::= < *initial_sol* >,< *pivoting_rule* >,< *termination* >,< *neighborhood* >
<intial_sol> ::= *random* | *slack* | *nwslack* | *nrz* | *nrz*2 | *mneh*
<termination> ::= *locmin* | *maxstep* | *soater*
<pivoting_rule> ::= *first improvement* | *best improvement*
<neighborhood> ::= *exchange* | *transpose* | *insert* | *finsert* | *binsert* | *tinsert*

are explained in Marmion et al. (2013); López-Ibáñez et al. (2017); Stützle and López-Ibáñez (2019).

The possible ways of constructing an algorithm are represented by a grammar, which ensures correctness of the algorithm instantiations. In a nutshell, the grammar dictates the possible ways of constructing a correct algorithm from algorithmic components. Consider the example of creating a generic iterative improvement algorithm. For such an iterated improvement algorithm, we need to specify a neighborhood, a pivoting rule that determines which neighbored solution replaces the current one and a termination criterion. This rule and possible alternatives are encoded in the snippet of the grammar given in Algorithm 2.

Instead of deriving possible algorithm compositions directly from the grammar, we transform the grammar into a parametric representation following Mascia et al. (2014). As our grammar includes a recursive rule, to enable this

transformation we cut the recursion at the third level. A main advantage of doing a translation of the grammar representation into a parametric one is to allow the exploitation of standard AAC techniques (Mascia et al., 2014). In particular, we use as an AAC tool irace López-Ibáñez et al. (2016), which is available at `http://iridia.ulb.ac.be/irace/`.

irace generates possible algorithms, to which we refer as "candidates" by sampling values for each of the relevant parameters. At each iteration, irace generates a set of candidates and performs a "race" in which poor candidates are discarded in favor of the best. During one race, irace iteratively executes candidate algorithm configurations on problem instances one by one. If at some point during the race some candidate configurations are identified as performing inferiorly to others, they are dropped from the race. The core principle of each race holds some similarities with the principles of horse race algorithm comparisons or more modern judgemental systems for programming competitions (for instance OPTIL.io by Wasik et al., 2016). A race stops once a maximum number of experiments is reached or the number of surviving candidates drops below a pre-specified bound. Before starting a new race, irace biases the sampling mechanism for generating parameter values towards the best configurations, thus, intensifying the search in the parameter space. Through the combination of irace with EMILI we obtain our proposed automatic algorithm design approach. Using possible parameter values and training instances as the input, the generation and identification of highly-performing algorithms is fully automated.

4.3. EMILI components

One key ingredient in EMILI is the set of algorithmic components it can choose from. In the following, we describe at a high-level the main algorithmic components we have considered for this work. In the set of components, we have focused on generic components that are freely composable with others and we have renounced problem-specific components. Nevertheless, our subsequently presented computational results show that, even with generic choices, very good is already demonstrable.

Metaheuristics. From our framework, we can directly instantiate the metaheuristics Iterated Local Search, Iterated Greedy, Variable Neighborhood Descent and Tabu Search by appropriate choices for the components defined above as well as combinations of these metaheuristics.

Initial solution generation. As possiblities of how to generate initial solutions we implemented the following: A random solution (indicated by *random*),

the slack heuristic (*slack*), the slack heuristic with weights that is used as an initial sequence for an insertion heuristic (*nwslack*) (Dubois-Lacoste et al., 2011), an insertion heuristic that uses as initial seed sequence the RZ heuristic (Rajendran and Ziegler, 1997) (*nrz*), a variant of *nrz* that does not use the local search proposed in the RZ heuristic (*nrz2*), and a variant of the NEH heuristic Nawaz et al. (1983) that uses the initial sequence as defined by increasing order of standard deviations in the processing times of the jobs (*mneh*) (Ding et al., 2015).

Neighborhood. We have implemented five neighborhoods, including the contiguous transpose neighborhood (*transpose*), the pairwise exchange neighborhood (*exchange*), the general insert neighborhood (*insert*), the insert neighborhood restricted to forward (*finsert*) or backward insertion moves (*binsert*) and the insert neighborhood that considers blocks of two jobs for insertion (*tinsert*).

Pivoting rule. The pivoting rules we consider are the first improvement and the best improvement rule.

Termination criteria: The termination criteria chosen as components of the algorithm refer to terminating local search runs; the overall termination of the configured algorithm is taken as a fixed computation time. The termination criteria available as algorithm components are local minium (*locmin*), a maximum number of search steps (*maxstep*) and a maximum number of search steps proportional to the instance size (*soater*).

Perturbations: A first perturbation does random moves in a specified neighborhood, which can be any of those described above; this is implemented by an option $rndmv(N, rm)$, which takes as a parameter the neighborhood N and the number of random moves rm. $igper(rd)$ implements the destruction and construction processes of iterated greedy algorithms, where rd is the number of jobs removed from the current solution (Ruiz and Stützle, 2007). $igls(rd, iterative\_imp)$ is analogous to $igper(rd)$ but in addition applies a local search on the partial solution obtained after jobs removal, following the ideas presented by (Dubois-Lacoste et al., 2017). $igio(rd)$ and $nrzper(rd)$ are destruction–construction perturbations that reorder the jobs before construction according to a non-increasing sum of processing times or the RZ heuristic respectively. Finally $tmiigper(rd, tl)$ introduces a tabu mechanism into the perturbation to avoid re-inserting jobs into previous positions; tl is the length of the tabu memory.

Acceptance criteria. The role of the acceptance criterion is to decide whether to accept a new candidate solution as the new incumbent or not.

The criterion $AC_{IG}(T)$ computes a fixed temperature for the Metropolis condition as used by (Ruiz and Stützle, 2007), parameterized by parameter T. The acceptance criteria $AC_{SA-L}(T_i, T_e, T_r)$, $AC_{SA-pm}(T_i, T_e, T_r, sa_i)$, and $AC_{SA-M}(T_i, T_e, T_r, sa_i, \alpha)$ are variations of the Metropolis condition embedded into an annealing schedule that is defined by parameters relating to the initial temperature T_i, the end temperature T_e, the linear reduction of the temperature T_r and the multiplicative annealing coefficient α; sa_i is a parameter that specifies the number of iterations after which the temperature is updated, which is done in AC_{SA-M} after every iteration. $AC_{RA}(pr)$ accepts a new candidate solution with a fixed probability pr. Finally the criteria AC_{RW} and AC_{Better} always accept a new candidate solution (independent of its quality) or only if it is better than the incumbent one respectively.

Tabu criterion. Finally, we have implemented five different ways of defining the tabu memory in tabu search algorithms, each parameterized by the tabu tenure tl. The available options to define the tabu status of a move, which is defined as a pair of integers (i, j) corresponding to the job j moved from its old position i as follows: forbid the inverse move (*move*), using a hash function of the solution (*hash*), forbidding the solution itself (*solution*), or forbidding insertions in the proximity of positions where a job was removed (*pos_r*) or removed and inserted (*pos_ri*).

As mentioned before, the grammar representation is translated into a parametric representation for the automated configuration process with irace. Depending on the number of recursion levels (from one to three) that are allowed, this results in between 169 and 446 parameters to be set by irace.

5. Computational and statistical experiments

As mentioned in Section 2, we work with three different objectives: $C_{\max}$, TFT and $TWET^{dw}$. These objectives seem related but they are actually very different as regards the solution and objective space topology. For example, $C_{\max}$ and TFT are regular functions of the job completion times C_j, i.e., only a reduction of at least one C_j leads to a decrease in either $C_{\max}$ or TFT. However, $TWET^{dw}$ is not a regular function of C_j and one may need to insert idle times in the sequences to ensure that no jobs finish before their due date windows and, thus, to improve their quality.

As competing algorithms we have selected the three state-of-the-art methods for each objective: *DABC* for $C_{\max}$ by Pan et al. (2014), *MBO* for TFT by Pan and Dong (2014) and *ILST* for $TWET^{dw}$ by Pan et al. (2017). We have

Algorithm 3 Snapshot of the grammar rules that are used to derive a hybrid metaheuristic algorithm. The rules to derive iterative improvement algorithms are given in Algorithm 2. The derivation starts by applying the rule for **<ILS>** and replacing the non-terminals on the right side with the respective rules. Non-terminals are delimited by angular parentheses.

<ILS> ::= $< initial\_sol >, < ILS\_main >$
<ILS_main> ::= $< ls >, < termination >, < perturbation >, < acceptance >$
<intial_sol> ::= *random* | *slack* | *nwslack* | *nrz* | *nrz2* | *mneh*
<tabu> ::= $< neighborhood >, < pivoting\_rule >, < initial\_sol >, < tabu\_tenure >$ $, < termination >$
<VND> ::= $< neighborhoods >, < pivoting\_rule >, < initial\_sol >, < termination >$
<ls> ::= $< iterative\_imp >$ | $< VND >$ | $< tabu >$ | $< ILS >$
<perturbation> ::= $< ig >$ | $< ig\_ls >$ | $< igio >$ | $< rndmv >$ | *noper* | $< nrzper >$ | $< tmiig >$
<ig> ::= *igper*, *rd*
<igio> ::= *igio*, *rd*
<nrzper> ::= *nrzper*, *rd*
<igls> ::= *igls*, *rd*, $< iterative\_imp >$
<tmiig> ::= *tmiigper*, *rd*, *tl*
<rndmv> ::= *rndmv*, $< neighborhood >$, *rm*
<acceptance> ::= AC_{RA} *pr* | AC_{RW} | AC_{Better} | AC_{IG} *T* | $< AC_{SA-L} >$ | $< AC_{SA-PM} >$ | $< AC_{SA-M} >$
$<AC_{SA-L}>$ ::= AC_{SA-L}, T_i, T_e, T_r
$<AC_{SA-PM}>$ ::= AC_{SA-PM}, T_i, T_e, T_r, sa_i
$<AC_{SA-M}>$ ::= AC_{SA-M}, T_i, T_e, T_r, sa_i, α
<neighborhoods> ::= $< neighborhood >, < neighborhoods >$ | $\emptyset$
<tabu_tenure> ::= *size*, *move* | *hash* | *solution* | *pos_r* | *pos_ri*

---

obtained the original source codes and have the details of the calibrations from the original authors. One of the objectives in this paper is to demonstrate that AAD is more robust to changes in the problem definition (like the objective) when compared to other state-of-the-art algorithms. These algorithms usually do not show state-of-the-art performance when tested with other objectives and are therefore not robust. This holds true even if the algorithm is carefully recalibrated for another objective. To this end, we will test *DABC* for its original $C_{\max}$ objective and we refer to it as $DABC^*$. In later sections, we will re-calibrate *DABC* for the *TFT* objective (referred to as $DABC_f$) and for the $TWET^{dw}$ objective (referred to as $DABC_w$). The same logic is applied to algorithms *MBO* and *ILST*. All these methods are tested against the AAD outcomes of the tested framework. With different solution space topologies we

can expect different behavior from AAD runs. Therefore, potentially different algorithms might be generated by the AAD methodology for each objective, but a main difference being that there is no human intervention as the process is fully automatic. Additionally, this different behavior will test the capacity of adaptation of the AAD methodology to the different objectives.

5.1. Instance sets

We have used four different instance sets. Two for makespan and total flow time and two more for total weighted earliness tardiness with due date windows. For each objective there is one calibration and one testing benchmark different from the calibration one. For $C_{\max}$ and TFT, there are some benchmarks proposed in the literature, mainly for $C_{\max}$, which can be used without changes for TFT. Nevertheless, most of these benchmarks have a limited number of instances, which complicates statistical analyses. Furthermore, the maximum instance size in these sets is also limited. For these reasons, we generated a comprehensive set of instances that is at the same time large enough for easy statistical analysis and contains large instances. More specifically, the final testing benchmark contains instances where all combinations of the following factors are tested: number of jobs $n \in \{50, 100, 150, 200\}$, number of stages $i \in \{2, 4, 6, 8, 10\}$ and number of machines per stage $m \in \{2, 4, 6, 8, 10$ and U[1-10]$\}$. Each combination is replicated 10 times, so there are $4 \times 5 \times 6 \times 10 = 1200$ test instances. A total of 120 calibration instances are generated by randomly choosing (with different random seeds) the values of n, i and m. We have included in all instances the best lower bound among the nine state-of-the-art bounds of Hidri and Haouari (2011) for the $C_{\max}$ objective. For the $TWET^{dw}$ objective we have used the instances from the previous work by Pan et al. (2017). The process to generate such instances is quite involved and the interested reader is referred to that work for more details. Suffice to say, that the factors used to build the instances are: $n \in \{50, 100, 150, 200\}$, $m \in \{5, 10\}$, $i \in \{5, 10\}$, tardiness factor $T \in \{0.2, 0.4, 0.6\}$, range of due dates factor $R \in \{0.2, 0.6, 1.0\}$ and width of due date windows $W \in \{10, 20\}$. In this case there are 1440 final test instances and 144 calibration instances. All sets of instances, along with the best solutions, are available at `http://soa.iti.es`.

5.2. Performance measures

To compare the results from the various metaheuristics, we compute the Relative Percentage Deviation RPD, defined in Eq. (1) below, for $C_{\max}$ and

TFT and the Relative Deviation Index RDI, defined in Eq. (2), for $TWET^{dw}$:

$$RPD = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \times 100 \tag{1}$$

$$RDI = \begin{cases} 0 & \text{if } Worst_{sol} = Best_{sol} \\ \dfrac{Method_{sol} - Best_{sol}}{Worst_{sol} - Best_{sol}} \times 100 & \text{if } Worst_{sol} \neq Best_{sol} \end{cases} \tag{2}$$

For each instance, we have available the minimum (maximum) objective function value $Best_{sol}$ ($Worst_{sol}$) and $Method_{sol}$ is the objective function value obtained by a metaheuristic in a specific run for that instance. RPD has a direct interpretation, while RDI is a normalized value between 0 and 1. Note that RPD cannot be used for $TWET^{dw}$ as $Best_{sol}$ might be zero.

5.3. Computation times

During calibration and test, every algorithm uses the same termination criterion based on elapsed CPU time. The running time depends on the instance size and is calculated with the formula $n \cdot m \cdot (\rho + \beta)$. β is set to 30 only for the $TWET^{dw}$ objective and to 0 in all other cases because the $TWET^{dw}$ evaluation requires more time. In the final tests ρ is set to to three different levels $\{30, 60, 90\}$ in order to observe the effect of additional CPU time. In the calibration, we employ the lower value of ρ.

We can also consider the CPU computation time for traditional calibration and irace automatic algorithm generation. ANOVAs required a variable but significant amount of CPU time, directly related with the instances, replicas, factors and levels studied. For example, the ILST calibration for the $TWET^{dw}$ objective involved 5 parameters/factors at 4, 4, 3, 3 and 2 levels/variants, respectively, resulting in $4^2 \times 3^2 \times 2^1 = 288$ treatments with a total of 14580 CPU hours considering the five replicates.
Considering that irace is not an exact algorithm, its results improve over time.We set up the tuning sessions in such a way to use very modest CPU times. So, the most time consuming irace algorithm generation took 10729 CPU hours. Conversely, for regular algorithms, the ANOVAs required a variable but significant amount of CPU time. As a result, an additional benefit of the AAD methodology is to potentially lower the CPU time required for the configuration process. It has to be stressed out that the AAD methodology is actually exploring (425-446 parameters) a much larger treatment space than

ANOVA. This is a clear advantage of the AAD methodology over ANOVA. In fact, experiments with more than a few factors are intractable with ANOVA as the number of treatments grows exponentially.

5.4. Calibration of the tested methods

In order to make a fair comparison between our AAD algorithms and the state-of-the-art methods, we calibrated each algorithm for each objective. This is important mainly for two reasons: First to validate our calibration, we compare the results with the original calibration from the authors. Second, since we employ a different benchmark from that of the original authors, a calibration ensures that there is no deterioration in results. This calibration has been achieved through full factorial ANOVAs.

All algorithms have been compiled in Visual Studio 2013 with optimizations enabled but without language extensions in order to improve performance. We have at our disposal two computing clusters. The SOA-Cluster consists of Windows 7 virtual machines with one virtual processor and 4 GB of RAM memory. Virtual machines run in an OpenStack virtualization platform supported by 12 blades, each one with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GB of RAM, for a total of 576 cores and 3 TBytes of RAM. The second one, IRIDIA-Cluster, contains 16-core AMD Opteron Interlagos 6272 processors running at 2.1 GHz under Cluster Rocks Linux with CentOS 6.3. The details concerning the calibration, instances and results are available as on-line materials. The final employed values for the algorithm parameters, obtained through the calibration, are given in Table 1. The stopping criterion for the ANOVA calibrations is $n \cdot m \cdot (\rho + \beta)$ seconds, where ρ is set to 30 in all cases and β is set to 30 only for the $TWET^{dw}$ objective and 0 in all other cases.The results of the calibration proved to be identical for the ILST method by Pan et al. (2017) and the $TWET^{dw}$ objective. This is expected as it is the same algorithm with the same benchmark. For the other algorithms, there are minor differences mainly due to the fact that we use larger instances.

As will be detailed in Section 4, the outcome of the AAD methodology is three automatically designed algorithms, one for each objective. The configuration procedure consisted of four independent executions of irace on the IRIDIA-Cluster and the configurations output was fed to a final irace execution that was used to generate the best automatically designed algorithm, which is refereed to as *ADA*.

	Parameter	Objectives		
		$C_{\max}$	TFT	$TWET^{dw}$
DABC	α	30	50	50
	P_{size}	20	25	25
	γ	1	-	-
	τ	1	-	-
MBO	P_{size}	21	31	21
	$NeighborType$	*insert*	*insert*	*hybrid*
	β	5	5	3
	γ	2	2	1
	$tour_{size}$	5	5	5
ILST	ω	2	2	2
	ϖ	50	30	30
	$Loop_{\max}$	200	300	300
	θ	4	2	4
	$2LS$	-	-	Yes

Table 1: Parameter values obtained after calibrating each algorithm for each objective

The most time consuming configuration run took 10729 CPU hours. Conversely, for the regular algorithms, the ANOVAs have required a variable but significant amount of CPU time. For example, the ILST calibration for the $TWET^{dw}$ objective involved 5 parameters/factors at 4, 4, 3, 3 and 2 levels, respectively, resulting in $4^2 \times 3^2 \times 2^1 = 288$ treatments with a total of 14580 CPU hours considering the five replicates. As a result, an additional benefit of the AAD methodology is a potentially lower CPU time requirement for the configuration process. It has to be stressed that the AAD methodology is actually exploring a much larger treatment space than ANOVA, which is another advantage of the AAD methdology, as experiments with more than a few factors are intractable with ANOVA as the number of treatments grows exponentially.

5.5. AAD candidates

We now detail the three *ADAs* resulting from the AAD methodology. Algorithms 4-6 detail the *ADAs* for $C_{\max}$, TFT and $TWET^{dw}$ respectively. These are referred to as ADA_m, ADA_f, and ADA_w respectively. We can see that ADA_m and ADA_f have some resemblance as they have a similar general structure (Iterated Greedy) to the Metropolis acceptance criterion. They also share the initial constructive heuristic *nrz2* while ADA_w has *slack*.

Algorithm 4 ADA_m: Iterated Greedy (IG) method for $C_{\max}$.

Output The best solution found π_{best}
π_0 := *nrz2*()
π := $\mathsf{LS}_m(\pi_0$, *first improvement, locmin, transpose*)
$\pi_{best} := \pi$
while termination criterion not satisfied **do**
    π' := *igper* perturbation(π, rd)
    π := iterative_imp$_m(\pi_0$,*first improvement, locmin, transpose*)
    π := AC_{SA-L} acceptance$(\pi, \pi', T_i, T_e, T_r)$
    if $f(\pi) < f(\pi_{best})$ **then**
        $\pi_{best} := \pi$
    end if
end while

However, they have different local search neighborhoods, different destruction components and different parameters. For ADA_m, the number of jobs to destroy (d) is 3, temperature (T) starts at 3.9295 and ends at 0.4806 and has a cooling factor of 0.0793. For ADA_f, d is four, T starts at 3.9774 and ends at 0.2796 and the cooling factor is 0.0331. (By default, the real-valued parameters are set with four decimal positions by the irace technique. Such a setting would not be possible with ANOVA unless complex mixture design of experiments and response surface methodology tools were used.) Different from ADA_m and ADA_f, ADA_w follows an interesting hybrid template. In Algorithm 6, we observe that the general template is again an IG method but the local search procedure is an ILS method (specified in Algorithm 7), essentially, it is an ILS that is run inside an IG. This is a salient feature of AAD, by which combinations of components never tested before might be obtained. Apart from this hybrid template, all the components in both the IG and ILS have different perturbation, destruction, acceptance criterion and local search, etc.

Basically, AAD has resulted in an algorithm that is completely different from those obtained for the other objectives. This indicates that the AAD methodology can adapt the different parameters and operators depending on the objective.

Algorithm 5 ADA_f: Iterated Greedy (IG) method for TFT.

Output The best solution found π_{best}
π_0 := *nrz2*()
π := $\mathsf{LS}_f(\pi$, *first improvement, locmin, exchange*)
$\pi_{best} := \pi$
while termination criterion not satisfied **do**
    π' := *igls* perturbation(π, rd)
    π := iterative_imp$_f(\pi$, *best improvement, locmin, transpose*)
    π := AC_{SA-L} acceptance$(\pi, \pi', T_i, T_e, T_r)$
    if $f(\pi) < f(\pi_{best})$ **then**
        $\pi_{best} := \pi$
    end if
end while

Algorithm 6 ADA_w: Iterated Greedy (IG) method for $TWET^{dw}$

Output The best solution found π_{best}
π_0 := *slack*()
π := $\mathsf{ILS}_w(\pi_0)$
$\pi_{best} := \pi$
while termination criterion not satisfied **do**
    π' := *igio* perturbation(π, rd)
    π'' := $\mathsf{ILS}_w(\pi')$ % see Algorithm 7
    π := $AC_{IG}(\pi'', \pi, T)$
    if $f(\pi) < f(\pi_{best})$ **then**
        $\pi_{best} := \pi$
    end if
end while

Algorithm 7 ILS_w: Iterated Local Search (ILS) method for $TWET^{dw}$

Input Current solution π
Output The best solution found π_{best}
π := $\mathsf{LS}_w(\pi$, *first improvement, locmin, exchange*)
$\pi_{best} := \pi$
for $i = 1$ to $Max\_Iterations$ **do**
    π' := *rndmv*$(\pi$, *s, transpose*)
    π'' := iterative_imp$_w(\pi'$, *first improvement, locmin, exchange*)
    π := $AC_{IG}(\pi'', \pi, T)$
    if $f(\pi) < f(\pi_{best})$ **then**
        $\pi_{best} := \pi$
    end if
end for

		Makespan($C_{\max}$)				Total flow time(TFT)			
ρ	n	$DABC^*$	ADA_m	$ILST_m$	MBO_m	$DABC_f$	ADA_f	$ILST_f$	MBO^*
30	50	**0.45**	0.61	0.69	0.83	1.14	**0.23**	0.52	0.63
	100	**0.37**	0.89	0.79	1.10	2.15	**0.36**	0.90	1.22
	150	**0.38**	1.00	0.80	1.13	2.72	**0.40**	1.06	1.57
	200	**0.40**	1.01	0.78	1.09	3.01	**0.41**	1.10	1.74
Average		**0.40**	0.88	0.76	1.04	2.25	**0.35**	0.90	1.29
60	50	**0.30**	0.44	0.56	0.58	1.00	**0.16**	0.44	0.51
	100	**0.21**	0.63	0.61	0.75	1.79	**0.22**	0.80	1.03
	150	**0.21**	0.73	0.62	0.82	2.25	**0.24**	0.95	1.33
	200	**0.22**	0.73	0.60	0.81	2.53	**0.23**	0.99	1.48
Average		**0.24**	0.63	0.60	0.74	1.89	**0.21**	0.79	1.09
90	50	**0.23**	0.35	0.49	0.45	0.92	**0.12**	0.40	0.46
	100	**0.15**	0.51	0.54	0.60	1.61	**0.15**	0.75	0.93
	150	**0.13**	0.60	0.54	0.67	2.00	**0.15**	0.89	1.22
	200	**0.14**	0.59	0.51	0.67	2.26	**0.14**	0.93	1.35
Average		**0.16**	0.51	0.52	0.60	1.70	**0.14**	0.74	0.99

Table 2: Average RPD for $C_{\max}$ and TFT objectives for every ρ in $n \cdot m(\rho + \beta)$ms. Best results in bold.

5.6. Results

We carried out final experiments on the SOA-cluster with all the calibrated algorithms and AAD outcomes. The benchmarks used now are the test benchmark sets, which have not been used in the calibration and automatic design phase, in other words, the test instances serve as an independent test set. We use a stopping criterion of $n \cdot m(\rho + \beta)$ seconds, where ρ is tested at three different and independent levels $\{30, 60, 90\}$ and β is again set to 30 for the $TWET^{dw}$ objective and 0 for the others.For each objective we have four algorithms. Table 2 shows the average RPD for the $C_{\max}$ and TFT objectives for the three ρ levels. The numbers in bold indicate the best average for each objective. Table 3 shows the results for the three different values of $\rho + \beta$ for the $TWET^{dw}$ objective.

For the $C_{\max}$ objective we observe how the tailored $DABC^*$ gives the best results regardless of the ρ values. It has to be noted that the good performance of $DABC^*$ is due to accelerated $C_{\max}$ objective evaluations and also due to the accelerated local search in the algorithm. More specifically, $DABC^*$ carries out finely tuned local searches in the so called exact neighborhoods. This gives the algorithm a significant advantage at the cost of losing modularity and generality. Basically, the local search, the solution evaluation and other parts of $DABC^*$ cannot be easily separated into independent components and

$\rho+\beta$	n	$stages$	$DABC_w$	ADA_w	$ILST^*$	MBO_w
30+30	50	5	59.68	**12.24**	19.12	36.56
		10	67.55	**12.45**	19.99	33.88
	100	5	65.56	14.56	**13.48**	35.61
		10	67.96	**10.63**	18.06	38.98
	150	5	64.84	20.13	**12.26**	36.75
		10	68.77	**14.55**	17.03	39.70
	200	5	62.77	20.66	**12.48**	40.67
		10	67.41	**18.47**	18.87	40.65
Average			65.57	**15.46**	16.41	37.85
60+30	50	5	50.21	**9.71**	15.54	28.59
		10	58.00	**9.39**	16.52	27.57
	100	5	52.07	**9.36**	10.56	27.64
		10	55.58	**7.39**	14.57	31.21
	150	5	54.09	15.38	**8.75**	26.36
		10	57.65	**9.48**	13.23	32.39
	200	5	53.81	17.68	**9.29**	27.22
		10	57.61	**13.55**	14.76	33.17
Average			54.88	**11.49**	12.90	29.27
90+30	50	5	44.93	**7.69**	13.32	24.66
		10	52.41	**7.81**	14.57	23.81
	100	5	43.80	**6.26**	8.78	22.81
		10	48.33	**5.29**	12.48	27.52
	150	5	46.99	11.90	**6.69**	20.23
		10	50.42	**6.36**	10.80	27.27
	200	5	47.52	15.43	**7.27**	20.79
		10	51.13	**10.12**	12.20	27.05
Average			48.19	**8.86**	10.76	24.27

Table 3: Average RDI for the $TWET^{dw}$ objective for different $\rho+\beta$ values using $n \cdot m(\rho+\beta)$ms. Best results in bold.

it is basically an integrated algorithm only for the $C_{\max}$ objective. Our initial hypothesis in this paper is that a good algorithm for one objective might not necessarily translate into a method that performs well for another objective. $DABC^*$ is a clear empirical demonstration of this hypothesis. Even after recalibrating this method for the TFT and $TWET^{dw}$ objectives, we observe that it is, by far, the worst performing method from the comparison. For the TFT objective, even after allowing three times more CPU time ($\rho = 90$), $DABC_f$ gives an RPD of 1.70, which is significantly larger than that of the second worst method MBO^* with one third CPU time ($\rho = 30$). A similar result holds for the $TWET^{dw}$ objective as $DABC_w$ is again, by far, the worst method in the comparison.

The situation for the TFT and $TWET^{dw}$ objectives is more interesting and quite unexpected. The AAD methodology has produced two new state-of-

the-art methods for these two objectives. More specifically, ADA_f and ADA_w improve upon the previous state-of-the-art methods for each objective, namely MBO^* and $ILST^*$. This is quite remarkable as all the *ADAs* do not include specific accelerations or operators for any of the objectives. Furthermore, for TFT, the performance of ADA_f is noteworthy. Using only one third of the CPU time ($\rho = 30$), ADA_f obtains an average RPD of 0.35, which is less than half of the RPD of the second best method, $ILST_f$ where $\rho = 90$. The outperformance by ADA_w for the $TWET^{dw}$ objective is more modest but again this is due to the fact that the second best method, $ILST^*$ includes a second stage local search specially tailored to this objective, which ADA_w lacks. Even for the very competitive $C_{\max}$ objective, ADA_m is the third best method when $\rho = 30$ but when $\rho = 90$ it is technically tied with the second best method. Considering that it does not include any accelerations or problem-specific knowledge, this is quite an achievement.

While most of the observed differences in the RPD values of Tables 2-3 are large enough so as to be statistically significant, we carry out independent ANOVA tests as a confirmation. For each objective we consider all the results having a single factor (algorithm) and RPD as the response variable (the instance factors are also included as non-controllable factors in the ANOVA experiment). The overall means plots for all three objectives are shown in Figures 1-3. The plotted means include Honest Significant Differences confidence intervals at the 95% confidence level. Overlapping intervals indicate that the differences between the corresponding means are not statistically significant.

As can be observed, even after averaging for all ρ values, the observed differences among all algorithms and objectives are statistically significant.

6. Conclusions and future research

We have presented a methodology to automatically design and configure algorithms in a single process. We targeted a realistic scheduling problem, the hybrid flowshop with three different objectives, namely makespan, total flowtime and total weighted earliness tardiness with due date windows. The proposed methodology is able to combine a large number of algorithmic templates, operators and parameters into potentially new algorithms that result in surprisingly good performance without the need to specifically tailor components that are usually only useful for one objective. In fact, the automatic algorithm design methodology was shown to be able to produce

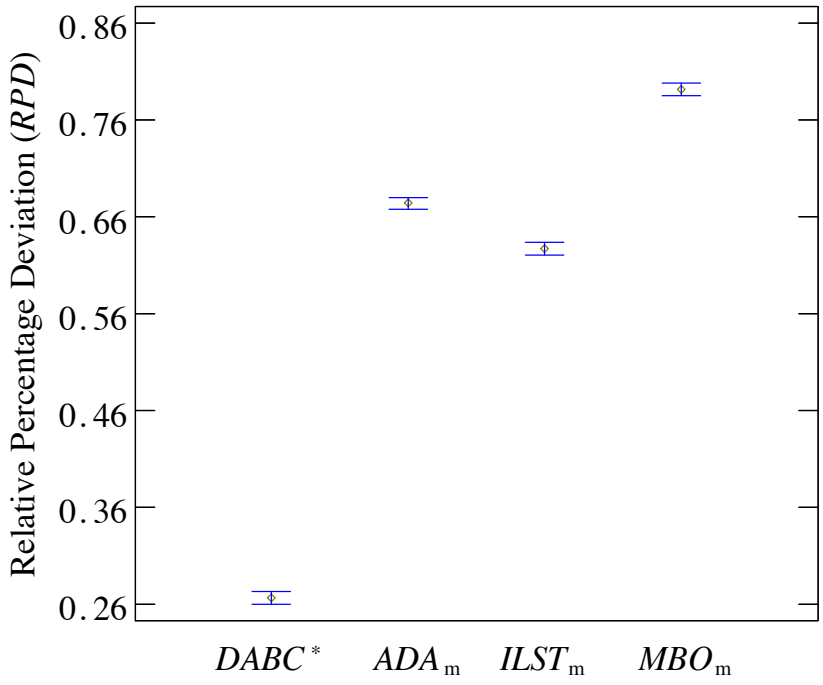


Figure 1: Average *RPD* means plot and confidence intervals for the $C_{\max}$ objective

extremely good results with minimal human intervention: for total flowtime and total weighted earliness tardiness with due date windows the automatically generated algorithms improved upon recent state-of-the-art results, in some cases by a significant margin. This is also remarkable, as our comprehensive computational and statistical testing has also demonstrated that state-of-the-art methods for a given objective do not necessarily translate into good performance for another objective, even if the algorithms are carefully recalibrated.

Given these promising results, we can expect a similarly good performance of the proposed automatic algorithm design methodology for more objectives, additional constraints or different scheduling problems. This can be done by adding additional algorithmic components to the framework and regenerating, using the automatic algorithm design methodology, algorithms for such problems. In fact, additional, more tailored algorithm components would be made available as options in the automated algorithm design process, an extension we are planning for future research. Other directions to take are extensions

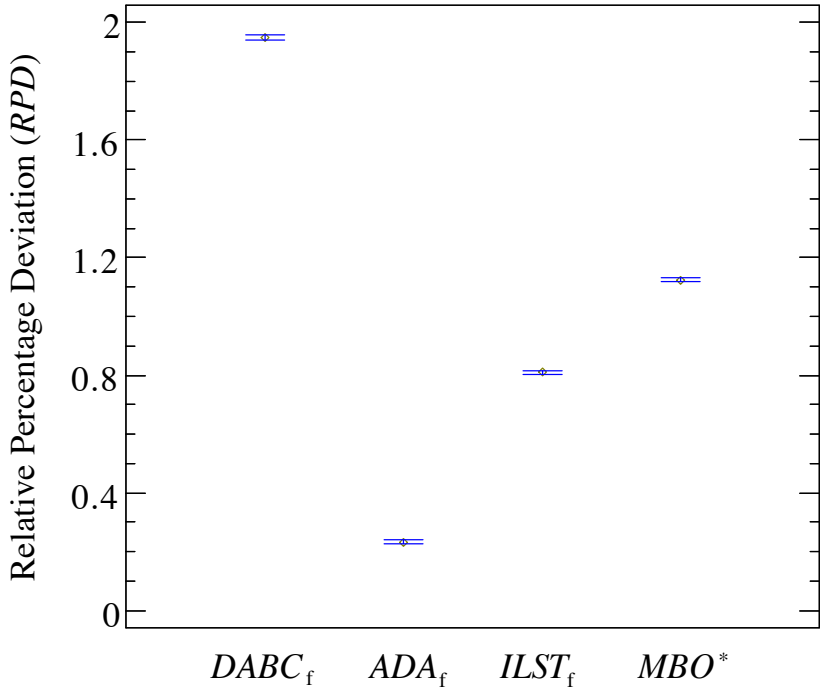


Figure 2: Average *RPD* means plot and confidence intervals for the *TFT* objective

of the framework by population-based algorithm templates and algorithmic components for manipulating populations of solutions. In doing so, interesting hybrid algorithms might be obtained by the proposed methodology.

Acknowledgements

Pedro Alfaro-Fernández and Rubén Ruiz are partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds and under grants BES-2013-064858 and EEBB-I-15-10089. This work was supported by the COMEX project (P7/36) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Director.

References

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Heidelberg.

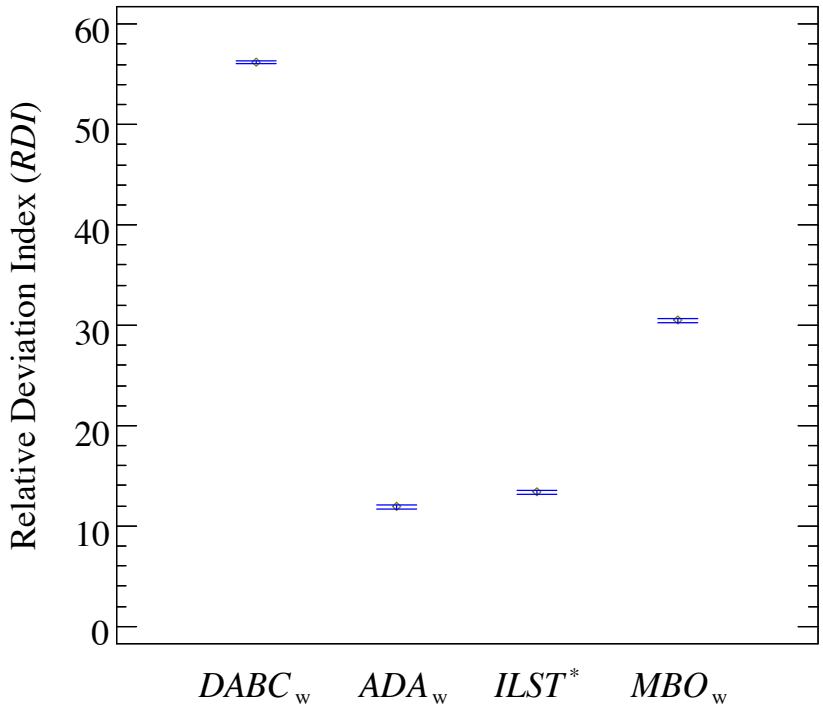


Figure 3: Average *RDI* means plot and confidence intervals for the $TWET^{dw}$ objective

Bożejko, W., Pempera, J., and Smutnicki, C. (2013). Parallel tabu search algorithm for the hybrid flow shop problem. *Computers & Industrial Engineering*, 65(3):466 – 474.

Bożejko, W., Gnatowski, A., Niżyński, T., Affenzeller, M., and Beham, A. (2018). Local optima networks in solving algorithm selection problem for TSP. In *International Conference on Dependability and Complex Systems*, pages 83–93. Springer.

Burke, E. K., Hyde, M. R., and Kendall, G. (2012). Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3):406–417.

Cahon, S., Melab, N., and Talbi, E.-G. (2004). Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380.

Carlier, J. and Néron, E. (2000). An exact method for solving the multi-processor flow-shop. *RAIRO-Operations Research*, 34(1):1–25.

Chung, T.-P. and Liao, C.-J. (2013). An immunoglobulin-based artificial immune system for solving the hybrid flow shop problem. *Applied Soft Computing*, 13(8):3729 – 3736.

Cui, Z. and Gu, X. (2015). An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. *Neurocomputing*, 148:248 – 259.

Ding, J.-Y., Song, S., Gupta, J. N. D., Zhang, R., Chiong, R., and Wu, C. (2015). An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing*, 30:604–613.

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236.

Dubois-Lacoste, J., Pagnozzi, F., and Stützle, T. (2017). An iterated greedy algorithm with optimization of partial solutions for the permutation flowshop problem. *Computers & Operations Research*, 81:160–166.

Framiñan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*. Springer Science & Business Media.

Franzin, A. and Stützle, T. (2016). Exploration of metaheuristics through automatic algorithm configuration techniques and algorithmic frameworks. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 1341–1347, New York, NY, USA. ACM.

Gupta, J. N. D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operational Research Society*, 39(4):359–364.

Gupta, J. N. D. and Stafford, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699 – 711.

Hidri, L. and Haouari, M. (2011). Bounding strategies for the hybrid flow shop scheduling problem. *Applied Mathematics and Computation*, 217(21):8248–8263.

Hoos, H. H. and Stützle, T. (2004). *Stochastic local search: Foundations and applications*. Elsevier.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). *Sequential Model-Based Optimization for General Algorithm Configuration*, pages 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hutter, F., Stützle, T., Leyton-Brown, K., and Hoos, H. H. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

Johnson, S. M. (1954). Optimal Two- and Three-Stage Production Schedules with Setup Times Included. *Naval research logistics quarterly*, 1(1):61–68.

Khalouli, S., Ghedjati, F., and Hamzaoui, A. (2010). A meta-heuristic approach to solve a jit scheduling problem in hybrid flow shop. *Engineering Applications of Artificial Intelligence*, 23(5):765 – 771.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2016). SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence*, 232:20–42.

Li, J.-Q., Pan, Q.-K., and Wang, F.-T. (2014). A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. *Applied Soft Computing*, 24:63 – 77.

Liao, C.-J., Tjandradjaja, E., and Chung, T.-P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing*, 12(6):1755 – 1764.

López-Ibáñez, M. and Stützle, T. (2012). The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M., Kessaci, M.-E., and Stützle, T. (2017). Automatic design of hybrid metaheuristics from algorithmic components. Technical Report TR/IRIDIA/2017-012, IRIDIA, Université Libre de Bruxelles, Belgium.

Lourenço, H. R., Martin, O., and Stützle, T. (2010). Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, chapter 9, pages 363–397. Springer.

Marichelvam, M., Prabaharan, T., and Yang, X.-S. (2014a). A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE Transactions on Evolutionary Computation*, 18(2):301–305.

Marichelvam, M., Prabaharan, T., and Yang, X.-S. (2014b). Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing*, 19:93 – 101.

Marichelvam, M., Prabaharan, T., Yang, X.-S., and Geetha, M. (2013). Solving hybrid flow shop scheduling problems using bat algorithm. *International Journal of Logistics Economics and Globalisation*, 5(1):15–29.

Marmion, M.-E., Mascia, F., López-Ibáñez, M., and Stützle, T. (2013). Automatic design of hybrid stochastic local search algorithms. In Blesa, M. J., Blum, C., Festa, P., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics: 8th International Workshop, HM 2013, Ischia, Italy, May 23-25, 2013. Proceedings*, pages 144–158, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., and Stützle, T. (2014). Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*, 51:190–199.

Montgomery, D. C. (2012). *Design and Analysis of Experiments*. John Wiley & Sons, eigth edition.

Nawaz, M., Enscore, Jr, E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, the International Journal of Management Science*, 11(1):91–95.

Pan, Q.-K. and Dong, Y. (2014). An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. *Information Sciences*, 277:643–655.

Pan, Q.-K., Ruiz, R., and Alfaro-Fernández, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80:50–60.

Pan, Q.-K., Wang, L., Li, J.-Q., and Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *OMEGA, the International Journal of Management Science*, 45:42–56.

Rajendran, C. and Ziegler, H. (1997). An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. *European Journal of Operational Research*, 103(1):129 – 138.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz, R. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18.

Sörensen, K. (2015). Metaheuristics-the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18.

Stützle, T. and López-Ibáñez, M. (2019). Automated design of metaheuristic algorithms. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 541–579. Springer, Cham.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

Vignier, A., Billaut, J. C., and Proust, C. (1999). Les problemes d'ordonnancement de type flow-shop hybride: Etat de l'art. *RAIRO Recherche Operationnelle*, 33(2):117.

Wang, S., Wang, L., Liu, M., and Xu, Y. (2013). An enhanced estimation of distribution algo-

rithm for solving hybrid flow-shop scheduling problem with identical parallel machines. *The International Journal of Advanced Manufacturing Technology*, 68(9):2043–2056.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2016). Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, pages 433–436, New York, NY, USA. ACM.

Xu, Y., Wang, L., Wang, S., and Liu, M. (2013). An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines. *Engineering Optimization*, 45(12):1409–1430.

figure 1

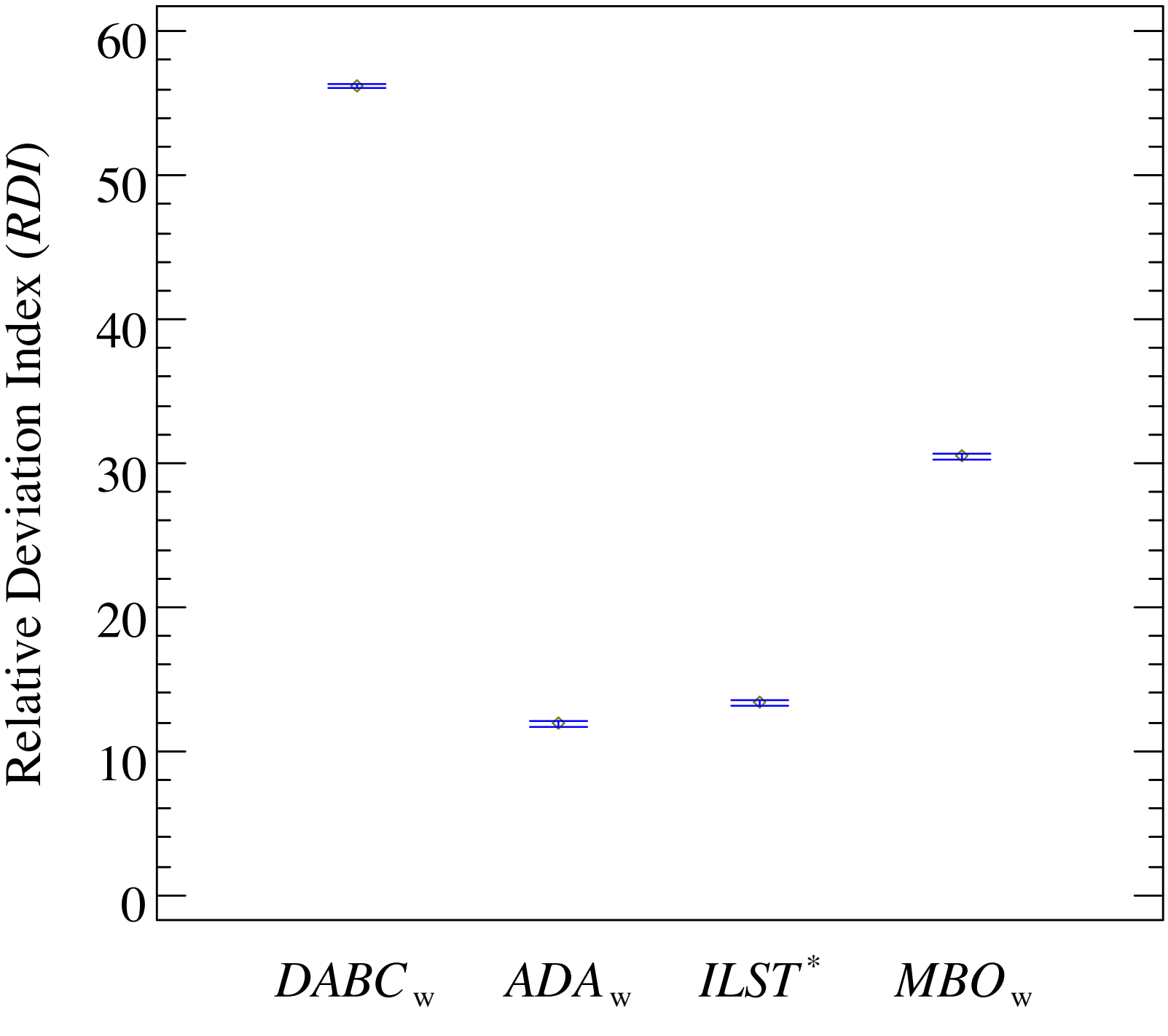

Relative Deviation Index (RDI)
60
50
40
30
20
10
0
DABC w
ADA w
ILST *
MBO w

figure 2

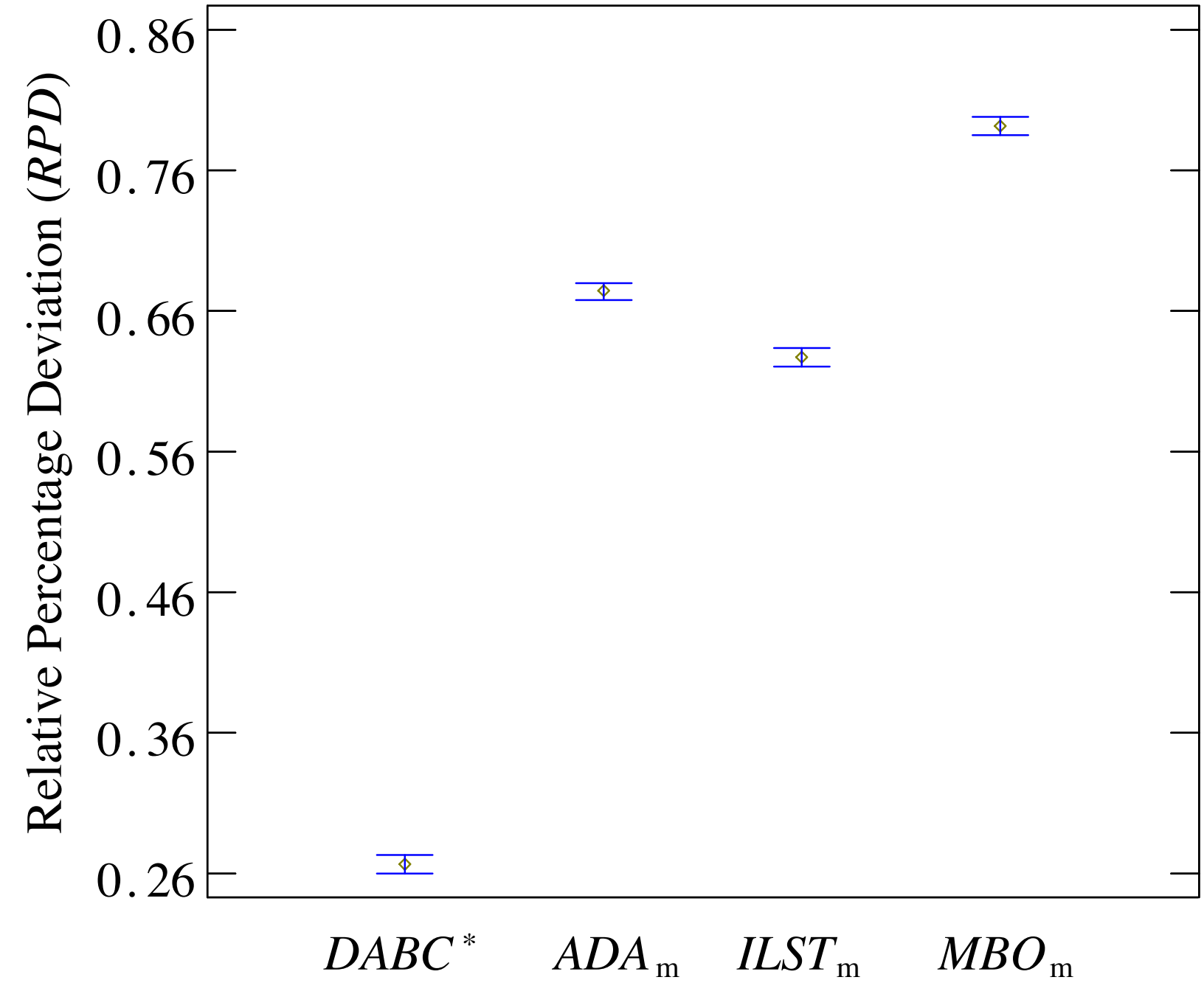
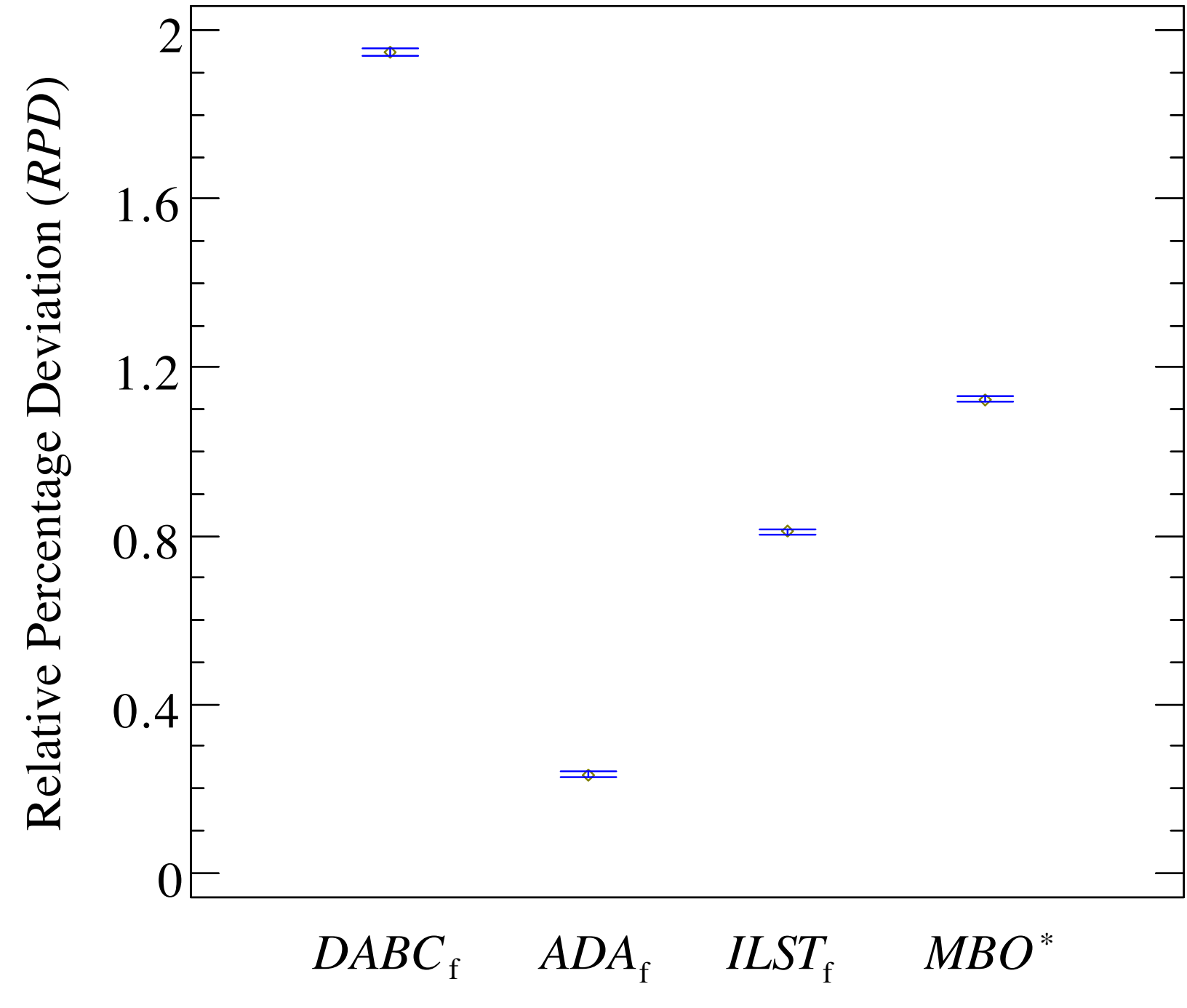

Relative Percentage Deviation (RPD)
0.86
0.76
0.66
0.56
0.46
0.36
0.26
DABC*
ADA m
ILST m
MBO m

figure 3


Relative Percentage Deviation (RPD)
2
1.6
1.2
0.8
0.4
0
DABC f
ADA f
ILST f
MBO *

online materials link

Click here to download Supplementary Material: online materials.txt

LaTeX Source Files tex bib cls pdf and figures

Click here to download LaTeX Source Files: Emili_paper_10.11_ejor_160719.zip