The 26th International Conference on Automated Planning and Scheduling

icaps

2016 – LONDON

Proceedings of the 11th Workshop on

Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS)

Edited by:

Miguel A. Salido, Roman Barták

London, UK, 13-14/06/2016

Organising Committee

Miguel A. Salido (msalido@dsic.upv.es)

Universidad Politécnica de Valencia (Spain)

Roman Barták (bartak@ktiml.mff.cuni.cz)

Charles University (Czech Republic)

Program Committee

Preface

The areas of planning and scheduling in Artificial Intelligence have seen important advances thanks to the application of constraint satisfaction and optimization models and techniques. Especially solutions to real-world problems need to integrate plan synthesis capabilities with resource allocation, which can be efficiently managed by using constraint satisfaction techniques. The workshop will aim at providing a forum for researchers in the field of Artificial Intelligence to discuss novel issues on planning, scheduling, constraint programming/constraint satisfaction problems (CSPs) and many other common areas that exist among them. On the whole, the workshop will mainly focus on managing complex problems where planning, scheduling and constraint satisfaction must be combined and/or interrelated, which entails an enormous potential for practical applications and future research.

In this edition, five papers were accepted. They represent an advance in the integration of constraint satisfaction techniques in planning and scheduling frameworks. These papers are distributed between theoretical papers and application papers.

Miguel A. Salido

Roman Barták

Table of Contents

Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning (Extended Abstract)*

Kyle E. C. Booth, Tony T. Tran, Goldie Nejat, and **J. Christopher Beck**
Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{kbooth, tran, nejat, jcb}@mie.utoronto.ca

Mobile Robot Task Planning

Driven by the increased use of mobile robotics for everyday applications, there has been a flurry of research activity in the pursuit of computationally efficient techniques for autonomous decision making (Gerkey and Matarić 2004). The automated planning and scheduling of tasks is of particular interest to the artificial intelligence (AI) and robotics communities, and considered a core competency of intelligent behavior. As such, the development and integration of solution techniques for such reasoning is fundamental to the successful design of autonomous mobile robots (Ghallab, Nau, and Traverso 2004).

Automated task planning and scheduling has been previously studied in mobile robotics applications such as warehouse management (Kim et al. 2003), hospital assistance, and human care (Cesta et al. 2011). There are a variety of existing solution methods, including those using mathematical programming techniques (Coltin, Veloso, and Ventura 2011), customized interval-algebra algorithms (Mudrova and Hawes 2015), and forward-chaining temporal planners (Louie et al. 2014).

In this work we investigate the application of optimization-based scheduling technologies to such robot task planning problems. Namely, we develop and apply *mixed-integer programming* (MIP) and *constraint programming* (CP) methods to solve two mobile robot task planning problems from the literature. Furthermore, for the second robot task planning problem, we integrate our CP task planning approach on the mobile social robot, Tangy.

In the first problem, a robot plans a set of tasks each with different temporal constraints dictating when a task is available for execution and when task execution must be completed. For this particular problem, the task planner must determine a feasible plan that minimizes the sum of task completion times. In the second mobile robot task planning problem, a socially-interacting robot must generate feasible task plans while adhering to a number of restrictions, including temporal constraints, the timetables of human users, and robot energy levels. We model and solve each of these problems with MIP and CP to find high-quality task plans. For the second problem, we demonstrate the physicaly utility of our methods by integrating our CP approach into a real robot architecture. Eliminating the need for algorithmic development, our model-and-solve techniques exploit ongoing advances within MIP and CP and our experimental results illustrate the promising nature of these general approaches for mobile robot task planning problems.

Optimization Technologies

Combinatorial optimization problems have been historically approached with a wide-range of methods including MIP and CP. MIP is a mathematical programming approach that models problems with continuous or integer variables whose values are restricted by linear constraints and contribute towards a global linear objective function. The approach commonly employs *branch-and-bound* tree search (Land and Doig 1960) and often avoids worst-case exponential search by solving the associated *linear programming* (LP) *relaxation* at each node to attain a bound on the objective and systematically prune subtrees. More sophisticated algorithmic developments have been proposed over the years, resulting in significant machine-independent speedups from the early 1990s to 2012 (Bixby 2012).

Conversely, CP is a rich approach that eschews structural restrictions and is capable of modeling constraints and variables of a variety of forms. Developed primarily within the AI community, CP focuses on the notation of *global constraints* to encapsulate frequently recurring combinatorial substructure. Such global constraints are combined in CP modeling and search effort is reduced through logical *inference* (Jaffar and Maher 1994) where each constraint has an associated algorithm that performs *domain filtering*. Such filtering removes values from variable domains that cannot participate in global solutions, and is performed at each node within the search. CP has also seen significant improvement in recent decades and has established itself as a viable alternative to mathematical programming-based approaches.

*The work in this extended abstract is presented in detail in a 2016 IEEE Journal article (Booth et al. 2016). It accompanies an invited talk given at the *Workshop on Constraint Satisfaction Techniques for Planning and Scheduling* (COPLAS2016) as part of the *26th International Conference on Planning and Scheduling* (ICAPS2016) on June 14, 2016.

This research has been funded by the Natural Sciences and Engineering Council of Canada (NSERC), Dr. Robot Inc., and the Canada Research Chairs (CRC) Program.

Robot Task Planning Problems

We study two mobile robot task planning problems, each requiring the autonomous assignment of start times to a set of tasks while adhering to problem constraints.

Task Planning Problem #1 Given a set of n tasks, $j \in J$, each with a release time, r_j, deadline time, d_j, and processing time, p_j, the robot must find a feasible task plan, or determine that none exist, over a planning horizon, H. Using standard scheduling terminology, this problem can be represented as $1|r_j, d_j, \delta_{jk}| \sum_j C_j$, where 1 represents the single robot, δ_{jk} defines the robot travel time between tasks j and k, and $\sum_j C_j$ is the objective function which minimizes the sum of task completion times. Robot travel times are asymmetric such that $\delta_{jk} \neq \delta_{kj}$ may hold, and follow the triangle inequality, namely $\delta_{jl} + \delta_{lk} \geq \delta_{jk}$. A solution task plan is a set of start times for each task, $\{s_1, s_2, ..., s_n\}$, such that these times adhere to the temporal constraints of each task (i.e. $s_j \in [r_j, d_j - p_j], \forall j \in J$), travel times are satisfied, and the objective is minimized.

We propose both MIP and CP models for this problem. Our disjunctive MIP model is defined by Eqs. (1) through (6), and uses decision variable $x_{jk} :=\{1$ if task j precedes task k, and 0 otherwise$\}$. In this model, Eqn. (1) is the minimization objective function, (2) defines task completion time, Eqs. (3) and (4) ensure a disjunctive relationship between all pairs of tasks, such that they do not conflict temporally, and the remainder of the model identifies variable domains.

$$\begin{aligned}
\min \quad & \textstyle\sum_j C_j & & (1)\\
\text{s.t.} \quad & C_j = s_j + p_j, & \forall j \quad & (2)\\
& C_j + \delta_{jk} \leq s_k + (H + \delta_{jk})(1 - x_{jk}), & \forall j, k \quad & (3)\\
& C_k + \delta_{kj} \leq s_j + (H + \delta_{kj})(x_{jk}), & \forall j, k \quad & (4)\\
& x_{jk} \in \{0, 1\}, & \forall j, k \quad & (5)\\
& s_j \in [r_j, d_j - p_j] & \forall j \quad & (6)
\end{aligned}$$

Our CP model is defined by Eqns. (7) through (10), making use of the $NoOverlap$ global constraint (Laborie 2009) in Eqn. (9) to prevent tasks from conflicting temporally including travel times, where Δ is the matrix of travel times between all pairs of tasks, δ_{jk}. Eqn. (7) defines the objective function, Eqn. (8) completion time, and the remainder identify varible domains.

$$\begin{aligned}
\min \quad & \textstyle\sum_j C_j & & (7)\\
\text{s.t.} \quad & C_j = s_j + p_j, & \forall j \quad & (8)\\
& NoOverlap(\{s_1, .., s_n\}, \{p_1, .., p_n\}, \Delta), & & (9)\\
& s_j \in [r_j, d_j - p_j] & \forall j \quad & (10)
\end{aligned}$$

There have been previously proposed methods for solving this problem within the literature. Specifically, *dynamic user task scheduling* (DUTS) (Coltin, Veloso, and Ventura 2011) introduces a pre-processing step that determines pairs of tasks with overlapping time windows and adds constraints similar to Eqs. (3) and (4) to a mathematical model before assigning start times via a MIP solver. An alternative method uses *task scheduling with interval algebra* (TSIA) (Mudrova and Hawes 2015) to heuristically order all pairs of tasks before using also using MIP to solve the problem. We note that each of these proposed methods are incomplete and not guaranteed to find a feasible solution if such a task plan exists. For larger optimization problems, global optimality may be unachievable within reasonable time, and thus heuristic methods may be preferred. As such, within our experimental analysis, we evaluate the solution-quality vs. run-time tradeoff of the different methods.

Task Planning Problem #2 Given a single-day planning horizon from 8:00AM to 7:00PM, the social robot Tangy must plan and facilitate a set of activities (tasks) involving human users while reasoning about temporal constraints, user timetables, and robot energy levels (Louie et al. 2014). The activities consist of *bingo games* (involving multiple users), *bingo game reminders* (involving a single user), and *robot recharge* tasks. The participants, location, and processing time of each task are known a priori, and the problem requires the robot to autonomously determine task start times and, in the case of optional robot recharge tasks, task presence and duration.

Each user has a timetable dictating when he/she is available, including mandatory breaks for meals from 8:00-9:00AM, 12:00-1:00PM, and 5:00-6:00PM. The set of bingo games and participants are parameters to the problem, and the robot must perform a reminder task with each user prior to his/her game. Robot travel times between any two locations are known, and a feasible task plan must account for these required transitions. Instantaneous battery level for the robot is available and must stay within pre-specified bounds. Each task type has a unique energy consumption rate, and *optional* robot recharge tasks allow for energy replenishment; an upper bound of these is supplied to the model, and they do not need to be utilized.

We solve this problem using both MIP and CP technologies, making use of continuous, integer, and binary decision variables within MIP and *optional interval variables* (Laborie 2009) within CP to properly model task optionality, in addition to a number of global constraints. Due to space limitations, these models, as well as a more comprehensive problem description, are detailed elsewhere (Booth et al. 2016). Prior to this work we proposed an approach for solving this problem using a forward chaining temporal planner (Louie et al. 2014), and we compare the results of our proposed models to this temporal planner.

Implementation & Experimental Analysis

Due to the application-driven focus on quickly finding feasible, high-quality task plans, we define algorithm performance based on run-time and optimality gap (%). Our methods are implemented in C++ on a hexacore machine with a Xeon processor and 12GB of RAM running Linux Ubuntu 14.04. We use the IBM ILOG CPLEX V12.6.2 Optimization Studio, which includes both MIP and CP solvers.

Benchmark problem sets are generated as identified in the journal version of this work (Booth et al. 2016), and the task plan solutions for the second problem are simulated using the Robot Operating System (ROS) (Quigley et al. 2009) on custom-developed visualization software. To validate the physical utility of our methods, the CP approach (best performing) is implemented within a ROS-based architecture on the mobile robot Tangy, using the GMapping technique in OpenSlam (openslam.org) to create an environment map via simultaneous localization and mapping.

Table 1 illustrates the MRE of the various approaches over time for Problem #1. These values are calculated according to the following expression: $MRE_{(CP,P40,0.1)} = \frac{1}{|\bar{P40}|}\sum_{p\in \bar{P40}} \frac{c(CP,p,0.1)-c^*(p)}{c^*(p)} \times 100$, which would yield the average MRE for the CP approach for all five problems with 40 tasks, $P40$, at a run-time duration of 0.1 seconds. In this expression $p \in \bar{P40}$ is the set of 40 task instances where feasible solutions were found at 0.1 seconds using CP. The value $c(CP, p, 0.1)$ is the best solution found by CP at this run-time for problem instance p, and $c^*(p)$ is the optimal solution, if known, or best known bound attained by running the MIP model for 18,000 seconds. If an approach failed to find any feasible plans at a specified run-time, a value of '-' is used. Values with a '†' indicate that MRE was calculated from the subset of instances for which the method found a feasible plan at the associated run-time. '# Inf.' identifies, for a technique, the number of instances for which no feasible plan was found after 100 seconds.

The proposed CP approach is able to find better solutions in shorter run-times than all other methods at nearly all time points, and our proposed MIP model generally outperforms existing MIP-based approaches. Furthermore, our methods do not sacrifice algorithmic completeness like the DUTS and TSIA methods, in part illustrated by the inability for the TSIA method to improve upon its initial heuristic solution and, in some cases, inability to find any feasible solutions.

Table 1: Problem #1: Mean relative error (%) over time

		Run-time (s)				
# Tasks	Technique	0.1	1	10	100	# Inf.
40	**CP**	**0.08**	**0.00**	**0.00**	**0.00**	0
	MIP	7.93	0.13	**0.00**	**0.00**	0
	DUTS	13.10	0.06	0.02	0.02	0
	TSIA	0.98	0.98	0.98	0.98	0
80	**CP**	**0.32**	**0.15**	**0.10**	**0.10**	0
	MIP	9.02	1.38	0.11	0.11	0
	DUTS	10.23	4.49	0.15	0.12	0
	TSIA	0.45†	0.45†	0.45†	0.45†	**2**
120	**CP**	**0.37**	**0.34**	**0.25**	**0.24**	0
	MIP	6.60†	3.67	**0.25**	0.25	0
	DUTS	7.06†	4.48	0.28	0.25	0
	TSIA	0.40†	0.40†	0.40†	0.40†	**4**
160	**CP**	**0.33**	**0.30**	**0.23**	**0.22**	0
	MIP	-	4.07	1.13	0.23	0
	DUTS	4.74†	3.08	0.85	0.23	0
	TSIA	0.33†	0.33†	0.33†	0.33†	**4**
200	**CP**	**0.26**	**0.25**	**0.20**	**0.18**	0
	MIP	-	3.56	1.63	**0.18**	0
	DUTS	4.77	3.83	1.93	**0.18**	0
	TSIA	-	-	-	-	**5**

Experimental results for the simulation of our proposed methods for the second problem are illustrated in Table 2. Again, CP is the dominant performing algorithm, finding feasible solutions much faster than the alternate methods. We note that though CP is by far the best approach for this problem, both the CP and MIP optimization-based technologies outperform the previously proposed forward-chaining temporal planning approach that uses OPTIC (Benton, Coles, and Coles 2012), even though the feasibility-focus of the problem favours the planning method over its optimization-based counterparts.

Table 2: Problem #2: Time to first feasible plan

Scenario		Technique		
Users	Bingo Games	**CP**	MIP	OPTIC
4	1	$<$ **0.01**	0.01	0.54
8	2	$<$ **0.01**	0.36	9.13
12	3	**0.04**	1.30	13.09
16	4	**0.01**	-	-
20	5	**0.08**	-	-

As a proof of concept, we implement our CP-approach in a real-world environment on the social robot, Tangy. We used the first scenario for this physical implementation, consisting of four users, one bingo game activity, and the associated reminder tasks. The results of this physical implementation are detailed within (Booth et al. 2016). This real-world experimentation is significant as it validates the physical utility of our task planning methods in realistic environments.

Conclusions & Future Work

We explored the modeling and solving of two robot task planning problems using optimization-based formalisms mixed-integer programming (MIP) and constraint programming (CP). The first problem involved the automated generation of feasible task plans that adhere to temporal constraints surrounding task release and deadline times. The second problem required reasoning about task precedence relationships, human user timetables, and robot energy consumption and replenishment. We implemented our models within simulated and real environments, comparing them with previous methods and concluding that, for the problems studied, the inference-based search of CP is the superior approach. Additionally, we implemented our CP approach for the second problem on the social robot Tangy to validate the physical utility of our methods.

Overall, our results indicate that these optimization-based techniques are promising for solving mobile robot task planning problems, and a main direction for our future research involves exploring the role of these methods for the development of re-planning and plan repair techniques. We also plan to further investigate robot task planning problems in order to understand the point at which such problems will require more sophisticated methods, including problem-based search manipulations and decompositions.

Acknowledgment

We would like to thank M. Schwenk for the design of the simulation environment and S. Mohamed for the robot navigation and mapping modules utilized in the experiments.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, volume 77, 78.

Bixby, R. E. 2012. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica, Extra Volume: Optimization Stories* 107–121.

Booth, K. E.; Tran, T. T.; Nejat, G.; and Beck, J. C. 2016. Mixed-integer and constraint programming techniques for mobile robot task planning. *Robotics and Automation Letters, IEEE* 1(1):500–507.

Cesta, A.; Cortellessa, G.; Rasconi, R.; Pecora, F.; Scopelliti, M.; and Tiberio, L. 2011. Monitoring elderly people with the robocare domestic environment: Interaction synthesis and user evaluation. *Computational Intelligence* 27(1):60–82.

Coltin, B.; Veloso, M. M.; and Ventura, R. 2011. Dynamic user task scheduling for mobile robots. In *Automated Action Planning for Autonomous Mobile Robots, AAAI Workshops*, volume WS-11-09.

Gerkey, B. P., and Matarić, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9):939–954.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

Jaffar, J., and Maher, M. J. 1994. Constraint logic programming: A survey. *The journal of logic programming* 19:503–581.

Kim, B.-I.; Heragu, S. S.; Graves, R. J.; and Onge, A. S. 2003. A hybrid scheduling and control system architecture for warehouse management. *Robotics and Automation, IEEE Transactions on* 19(6):991–1001.

Laborie, P. 2009. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer. 148–162.

Land, A. H., and Doig, A. G. 1960. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* 497–520.

Louie, W.-Y. G.; Vaquero, T.; Nejat, G.; and Beck, J. C. 2014. An autonomous assistive robot for planning, scheduling and facilitating multi-user activities. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 5292–5298.

Mudrova, L., and Hawes, N. 2015. Task scheduling for mobile robots using interval algebra. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 383–388.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5.

A CASP-Based Approach to PDDL+ Planning

Marcello Balduccini
Drexel University
marcello.balduccini@gmail.com

Daniele Magazzeni
King's College London
daniele.magazzeni@kcl.ac.uk

Marco Maratea
University of Genoa
marco@dibris.unige.it

Abstract

PDDL+ is an extension of PDDL that makes it possible to model planning domains with mixed discrete-continuous dynamics. In this paper we present a new approach to PDDL+ planning based on the paradigm of Constraint Answer Set Programming (CASP), an extension of Answer Set Programming that supports efficient reasoning on numerical constraints. We provide an encoding of PDDL+ models into CASP problems. The encoding can handle non-linear hybrid domains, and represents a solid basis for applying logic programming to PDDL+ planning. As a case study, we consider an implementation of our approach based on CASP solver EZCSP and present very promising results on a set of PDDL+ benchmark problems.

1 Introduction

Planning in hybrid domains is a challenging problem that has found increasing attention in the planning community, mainly motivated by the need to model real-world domains. Indeed, in addition to classical planning, hybrid domains allow for modeling continuous behavior with continuous variables that evolve over time. PDDL+ (Fox and Long 2006) is the extension of PDDL that allows for modelling domains with mixed discrete-continuous dynamics, through continuous processes and exogenous events.

Various techniques and tools have been proposed to deal with hybrid domains (Penberthy and Weld 1994; McDermott 2003; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005). More recent works include (Bryce et al. 2015), which presents an approach based on Satisfiability Modulo Theory (SMT) and restricted to a subset of the PDDL+ features, and (Bogomolov et al. 2014; Bogomolov et al. 2015) that combines hybrid system model checking and planning, but is only limited to proving plan non-existence.

To date, the only approach able to handle the full PDDL+ is the *discretise and validate* approach implemented in UPMurphi (Della Penna et al. 2009). There, the continuous model is discretised and forward search is used to find a solution, which is then validated against the continuous model using VAL (Fox, Howey, and Long 2004). If the solution is not valid, the discretisation is refined and the process iterates. The main drawback of UPMurphi, though, is the lack of heuristics that strongly limits its scalability, and hence its applicability to real case studies.

This motivates the need for finding new ways to handle PDDL+. To this aim, in this paper we present a new approach to PDDL+ planning based on Constraint Answer Set Programming (CASP) (Baselice, Bonatti, and Gelfond 2005), an extension of Answer Set Programming (ASP) (Gelfond and Lifschitz 1991) supporting efficient reasoning on numerical constraints. We provide an encoding of PDDL+ models into CASP problems, which can handle linear and non-linear domains, and can deal with PDDL+ processes and events. This contribution represents a solid basis for applying logic programming to PDDL+ planning, and opens up the use of CASP solvers for planning in hybrid domains.

We describe how the different components of a PDDL+ domain can be encoded into CASP. In our encoding, continuous invariants are checked at discretised timepoints, and following the discretise and validate approach (Della Penna et al. 2009), VAL is used to check whether the found solutions are valid or whether more timepoints need to be considered. As a case study, we use the CASP solver EZCSP (Balduccini 2009). Experiments performed on PDDL+ benchmarks show that our approach outperforms the state-of-the-art PDDL+ planners dReal and UPMurphi.

The paper is structured as follows. We begin with preliminaries on PDDL+ planning and CASP. In Section 3, we present our encoding, followed by a discussion of the results of our experiments. Finally, in Section 6, we draw conclusions and discuss future directions of work.

2 Background

In this section, we provide background on the main topics covered by the paper. We first introduce PDDL+ planning, and then ASP and CASP.

Hybrid systems can be described as hybrid automata (Henzinger 1996), that are finite state automata extended with continuous variables that evolve over time. More formally, we have the following:

Definition 1 (Hybrid Automaton) *A* hybrid automaton *is a tuple $\mathscr{H} = (Loc, Var, Init, Flow, Trans, I)$, where*

- *Loc is a finite set of locations, $Var = \{x_1, \ldots, x_n\}$ is a set of real-valued variables, $Init(\ell) \subseteq \mathbb{R}^n$ is the set of initial values for $x_1, \ldots, x_n$ for all locations ℓ.*
- *For each location ℓ, $Flow(\ell)$ is a relation over the variables in Var and their derivatives of the form*
$$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathscr{U},$$
where $x(t) \in \mathbb{R}^n$, A is a real-valued nxn matrix and $\mathscr{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set.
- *Trans is a set of discrete transitions. A discrete transition $t \in Trans$ is defined as a tuple (ℓ, g, ξ, ℓ') where ℓ and ℓ' are the source and the target locations, respectively, g is the guard of t (given as a linear constraint), and ξ is the update of t (given by an affine mapping).*
- *$I(\ell) \subseteq \mathbb{R}^n$ is an invariant for all locations ℓ.*

An illustrative example is given by the hybrid automaton for a thermostat depicted in Figure 1. Here, the temperature is represented by the continuous variable x. In the discrete location corresponding to the heater being off, the temperature falls according to the flow condition $\dot{x} = -0.1x$, while when the heater is on, the temperature increases according to the flow condition $\dot{x} = 5 - 0.1x$. The discrete transitions state that the heater *may* be switched on when the temperature falls below 19 degrees, and switched off when the temperature is greater than 21 degrees. Finally, the invariants state that the heater can be on (off) *only* if the temperature is not greater than 22 degrees (not less than 18 degrees).

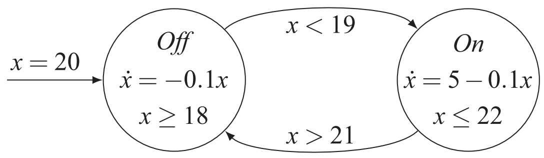


Figure 1: Thermostat hybrid automaton

Planning is an AI technology that seeks to select and organise activities in order to achieve specific goals (Nau, Ghallab, and Traverso 2004). A planner uses a domain model, describing the actions through their pre- and post-conditions, and an initial state together with a goal condition. It then searches for a trajectory through the induced state space, starting at the initial state and ending in a state satisfying the goal condition. In richer models, such as hybrid systems, the induced state space can be given a formal semantics as a timed hybrid automaton, which means that a plan can synchronise activities between controlled devices and external events.

2.1 PDDL+ Planning

Definition 2 (Planning Instance) *A planning instance is a pair $I = (Dom, Prob)$, where $Dom = (Fs, Rs, As, Es, Ps, arity)$ is a tuple consisting of a finite set of* function symbols *Fs, a finite set of* relation symbols *Rs, a finite set of (durative)* actions *As, a finite set of* events *Es, a finite set of* processes *Ps, and a function arity mapping all symbols in $Fs \cup Rs$ to their respective arities.*

The triple $Prob = (Os, Init, G)$ consists of a finite set of domain objects *Os, the* initial *state Init, and the* goal *specification G.*

Following (Bogomolov et al. 2014), for a given planning instance I, a *state* of I consists of a discrete component, described as a set of propositions P called *Boolean fluents*, and a numerical component, described as a set of real variables $\mathbf{v}$ called *numerical fluents*. Instantaneous actions are described through preconditions (which are conjunctions of propositions in P and/or numerical constraints over $\mathbf{v}$, and define when an action can be applied) and effects (which define how the action modifies the current state). *Instantaneous* actions and events are restricted to the expression of discrete change. Events have preconditions as for actions, but they are used to model exogenous change in the world, therefore they are triggered as soon as the preconditions are true. A process is responsible for the continuous change of variables, and is active as long as its preconditions are true. *Durative* actions have three sets of preconditions, representing the conditions that must hold when it starts, the invariant that must hold throughout its execution and the conditions that must hold at the end of the action. Similarly, a durative action has three sets of effects: effects that are applied when the action starts, effects that are applied when the action ends and a set of continuous numeric effects which are applied continuously while the action is executing.

Definition 3 (Plan) *A plan for a planning instance $I = ((Fs, Rs, As, Es, Ps, arity), (Os, Init, G))$ is a finite set of triples $(t, a, d) \in \mathbb{R}^* \times As \times \mathbb{R}^*$, where t is a timepoint, a is an action and d is the action duration.*

Note that processes and events do not appear in a plan, as they are not under the direct control of the planner.

2.2 Answer Set Programming

Let Σ be a signature containing constant, function and predicate symbols. Terms and atoms are formed as in first-order logic. A literal is an atom a or its classical negation $\neg a$. A *rule* is a statement of the form:

$$h \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n \quad (1)$$

where h and l_i's are literals and *not* is the so-called *default negation*. The intuitive meaning of the rule is that a reasoner who believes $\{l_1, \dots, l_m\}$ and has no reason to believe $\{l_{m+1}, \dots, l_n\}$, has to believe h. The formal semantics, defined in terms of models of a set of rules, is given later. We call h the *head* of the rule, and $\{l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n\}$ the *body* of the rule. Given a rule r, we denote its head and body by $head(r)$ and $body(r)$, respectively. A rule with an empty body is called a *fact*, and indicates that the head is always true. In that case, the connective $\leftarrow$ is often dropped.

A *program* is a pair $\langle \Sigma, \Pi \rangle$, where Σ is a signature and Π is a set of rules over Σ. Often we denote programs by just the second element of the pair, and let the signature be defined implicitly.

A set A of literals is *consistent* if no two complementary literals, a and $\neg a$, belong to A. A literal l is *satisfied* by a consistent set of literals A (denoted by $A \models l$) if $l \in A$. If l is not satisfied by A, we write $A \not\models l$. A set $\{l_1, \dots, l_k\}$ of literals is satisfied by a set of literals A ($A \models \{l_1, \dots, l_k\}$) if each l_i is satisfied by A.

Programs not containing default negation are called *definite*. A consistent set of literals A is *closed* under a definite program Π if, for every rule of the form (1) such that the body of the rule is satisfied by A, the head belongs to A. This allows us to state the semantics of definite programs.

Definition 4 *A consistent set of literals A is an* answer set *of definite program Π if A is closed under all the rules of Π and A is set-theoretically minimal among the sets closed under all the rules of Π.*

To define answer sets of arbitrary programs, we introduce the *reduct* of a program Π with respect to a set of literals A, denoted by Π^A. The reduct is obtained from Π by: **(1)** deleting every rule r such that $l \in A$ for some expression of the form *not l* from the body of r, and **(2)** removing all expressions of the form *not l* from the bodies of the remaining rules. The semantics of arbitrary ASP programs can thus be defined as follows.

Definition 5 *A consistent set of literals A is an* answer set *of program Π if it is an answer set of Π^A.*

To simplify the programming task, variables (identifiers with an uppercase initial) are allowed in ASP programs. A rule containing variables (a *non-ground* rule) is viewed as a shorthand for the set of its *ground instances*, obtained by replacing the variables by all possible ground terms. Similarly, a non-ground program is viewed as a shorthand for the program consisting of the ground instances of its rules.

There are also shorthands, which we introduce informally to save space. A rule whose head is empty is called *denial*, and states that its body must not be satisfied. A *choice rule* has a head of the form

$$\lambda\{m(\vec{X}) : \Gamma(\vec{X})\}\mu$$

where $\vec{X}$ is a list of variables, λ, μ are non-negative integers, and $\Gamma(\vec{X})$ is a set of literals that may include variables from $\vec{X}$. A choice rule intuitively states that, in every answer set, the number of literals of the form $m(\vec{X})$ such that $\Gamma(\vec{X})$ is satisfied must be between λ and μ. If not specified, λ, μ default, respectively, to 0, ∞. For example, given a relation q defined by $\{q(a), q(b)\}$, the rule:

$$1\{p(X) : q(X)\}2.$$

intuitively identifies three possible sets of conclusions: $\{p(a)\}$, $\{p(b)\}$, and $\{p(a), p(b)\}$.

2.3 Constraint ASP

CASP integrates ASP and Constraint Programming (CP) in order to deal with continuous dynamics. In this section we provide an overview of CP and of its integration in CASP.

The central concept of CP is the *Constraint Satisfaction Problem (CSP)* (Rossi, van Beek, and Walsh 2006), which is formally defined as a triple $\langle X, D, C \rangle$, where $X = \{x_1, \dots, x_n\}$ is a set of variables, $D = \{D_1, \dots, D_n\}$ is a set of domains, such that D_i is the domain of variable x_i, and C is a set of constraints. A *solution* to a CSP $\langle X, D, C \rangle$ is a complete assignment (i.e. where a value from the respective domain is assigned to each variable) satisfying every constraint from C.

There is currently no widely accepted, standardized definition of CASP. Multiple definitions have been given in the literature (Ostrowski and Schaub 2012a; Mellarkod, Gelfond, and Zhang 2008a; Baselice, Bonatti, and Gelfond 2005; Balduccini 2009). Although largely overlapping, these definitions are all somewhat distinct from each other.

To ensure generality of our results, we introduce a simplified definition of CASP, defined next, which captures the common traits of the above approaches. The main results of this paper will be given using our simplified definition of CASP. Later, in Section 4, we introduce a specific CASP language to discuss the use case and the experimental results.

Syntax. In order to accommodate CP constructs, the language of CASP extends ASP by allowing *numerical constraints* of the form $x \bowtie y$, where $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$, and x and y are *numerical variables*[1] or standard arithmetic terms possibly containing numerical variables, numerical constants, and ASP variables. Numerical constraints are only allowed in the head of rules.

Semantics. Given a numerical constraint c, let $\tau(c)$ be a function that maps c to a syntactically legal ASP atom and τ^{-1} be its inverse. We say that an ASP atom *a denotes* a constraint c if $a = \tau(c)$. Function τ is extended in a natural way to CASP rules and programs. Note that, for every CASP program Π, $\tau(\Pi)$ is an ASP program.

[1]Numerical variables are distinct from ASP variables.

Finally, given a set A of ASP literals, let $\gamma(A)$ be the set of ASP atoms from A that denote numerical constraints. The semantics of a CASP program can thus be given by defining the notion of CASP solution, as follows.

Definition 6 *A pair $\langle A, \alpha \rangle$ is a* CASP solution *of a CASP program Π if-and-only-if A is an answer set of $\tau(\Pi)$ and α is a solution to $\tau^{-1}(\gamma(A))$.*

3 Encoding PDDL+ Models into CASP Problems

In this section we describe our encoding of PDDL+ problems in CASP. Our approach is based on research on reasoning about actions and change, and action languages (Gelfond and Lifschitz 1993; Reiter 2001; Chintabathina, Gelfond, and Watson 2005). It builds upon the existing SAT-based (Kautz and Selman 1992) and ASP-based planning approaches (Lifschitz 1999), and extends them to hybrid domains.

In reasoning about actions and change, the evolution of a domain over time is often represented by a *transition diagram* (or *transition system*) that represents states and transitions between states through actions. Traditionally, in transition diagrams, actions are instantaneous, and states have no duration and are described by sets of Boolean fluents. Sequences of states characterizing the evolutions of the domain are represented as a sequence of *discrete time steps*, identified by integer numbers, so that step 0 corresponds to the initial state in the sequence. We extend this view to hybrid domains according to the following principles:

- Similarly to PDDL+, a state is characterized by Boolean fluents and numerical fluents.
- The flow of actual time is captured by the notion of *global time* (Chintabathina, Gelfond, and Watson 2005). States have a duration, given by the global time at which a state begins and ends. Intuitively, this conveys the intuition that time flows "within" the state.
- The truth value of Boolean fluents only changes upon state transitions. That is, it is unaffected by the flow of time "within" a state. On the other hand, the value of a numerical fluent may change within a state.
- The global time at which an action occurs is identified with the end time of the state in which the action occurs.
- Invariants are checked at the beginning and at the end of every state in which durative actions and processes are in execution. Thus, in order to guarantee soundness we exploit a discretize and validate approach.

Next, we describe the CASP formalization of PDDL+ models. We begin by discussing the correspondence between global time and states, and the representation of the values of fluents and of occurrences of actions.

The global time at which the state at step i begins is represented by numerical variable $start(i)$. Similarly, the end time is represented by $end(i)$. The truth value of Boolean fluent f at discrete time step i is represented by literal $holds(f,i)$ if f is true and by $\neg holds(f,i)$ otherwise. For every *numerical fluent n*, we introduce two numerical variables, representing its value at the beginning and at the end of time step i. The variables are $v\_initial(n,i)$ and $v\_final(n,i)$, respectively. The occurrence of an action a at time step i is represented by an atom $occurs(a,i)$.

Additive fluents, whose value is affected by *increase* and *decrease* statements of PDDL+, are represented by introducing numerical variables of the form $v(contrib(n,s),i)$, where n is a numerical fluent, s is a constant denoting a source (e.g., the action that causes the increase or decrease), and i is a time step. The expression denotes the amount of the contribution to fluent n from source s at step i. Intuitively, the value of n at the end of step i (encoded by numerical variable $v\_final(n,i)$) is calculated from the values of the individual contributions. Next, we discuss the encoding of the domain portion of a PDDL+ problem.

3.1 Domain Encoding

In the following discussion, ASP variables I, $I1$, $I2$ denotes time steps.

Actions. The encoding of the preconditions of actions varies depending on their type. Preconditions on Boolean fluents are encoded by means of denials. For example, a denial:

$$\leftarrow holds(unavail(tk1),I), occurs(refuel\_with(tk1),I).$$

states that refuel tank $tk1$ must be available for the corresponding refuel action to occur. Preconditions on numerical fluents are encoded by means of numerical constraints on the corresponding numerical variables. For example, a rule

$$v\_final(height(ball),I) > 0 \leftarrow occurs(drop(ball),I).$$

states that, if $drop(ball)$ is selected to occur, then the height of the ball is required to be greater than 0 in the preceding state.

The effects of instantaneous actions on Boolean fluents are captured by rules of the form:

$$holds(f,I+1) \leftarrow occurs(a,I).$$

where f is a fluent and a is an action. The rule states that f is true at the next time step $I+1$ if the action occurs at (the end of) step I. The effects on numerical fluents are represented similarly, but the head of the rule is replaced by a numerical constraint. For example, the rule:

$$v\_initial(height(ball),I+1) = 10 \leftarrow occurs(lift(ball),I).$$

states the action of lifting the ball causes its height to be

10 at the beginning of the state following the occurrence of the action. If the action increases or decreases the value of a numerical fluent, rather than setting it, then a corresponding variable of the form $v(contrib(n,s),i)$ is used in the numerical constraint. The link between contributions and numerical fluent values is established by axioms described later in this section.

Durative actions. A durative action d is encoded as two instantaneous actions, $start(d)$ and $end(d)$. The start (end) preconditions of d are mapped to preconditions of $start(d)$ ($end(d)$). The overall conditions are encoded with denials and constraints, as described above in the context of preconditions. Start (end) effects are mapped to effects of $start(d)$ and $end(d)$ actions. Additionally, $start(d)$ makes fluent $inprogr(d)$ true. The continuous effects of d are made to hold in any state in which $inprogr(d)$ holds. For example, if a $refuel$ action causes the level of fuel in a tank to increase linearly with the flow of time, its effect may be encoded by:

$$v(contrib(flevel,refuel),I) = end(I) - start(I) \leftarrow holds(inprogr(d),I).$$

The above rule intuitively states that, at the end of any state in which d is in progress, the fuel level increases proportionally to the duration of the state. The value of the fluent is updated from its set of contributions S by the general constraint, shown next, which applies to every fluent F:

$$v\_final(F,I) = v\_initial(F,I) + \sum_{s \in S} v(contrib(F,s),I).$$

The fact that the value of numerical fluents stays the same by default throughout the time interval associated with a state is modeled by a rule:

$$v\_final(F,I) = v\_initial(F,I) \leftarrow \text{not } ab(F,I).$$

which applies to every numerical fluent F. Intuitively, this rule must not be applicable when the value of F is being changed by an action, process, or event. This is enforced by adding a rule that makes $ab(F,I)$ true. For example, for a durative action d that affects a numerical fluent f, the encoding includes a rule:

$$ab(f,I) \leftarrow holds(inprogr(d),I).$$

In a similar way, the contribution to a numerical fluent by every source is assumed to be 0 by default. This is guaranteed by the rule:

$$v(contrib(F,S),I) = 0) \leftarrow \text{not } ab(F,I).$$

To keep track of the duration of a durative action when the action spans multiple time steps, a rule records the global time at which d begun:

$$stime(d) = end(I) \leftarrow occurs(start(d),I).$$

Action $end(d)$ is modeled so that it is automatically triggered after $start(d)$. Finding the time at which the end action occurs, both in terms of time step and global time, is part of the constraint problem to be solved. The following rule:

$$1\{occurs(end(d),I2) : I2 > I1\}1 \leftarrow occurs(start(d),I1).$$

ensures that $end(d)$ will be triggered at some timepoint following $start(d)$. Finally, requirements on the duration of durative actions are encoded using numerical constraints: if the PDDL+ problem specifies that the duration of d is δ, the requirement is encoded by a rule:

$$end(I) - stime(d) = \delta \leftarrow occurs(end(d),I).$$

Intuitively, any CASP solution of the corresponding program will include a specification of when $end(d)$ must occur, both in terms of time step and global time.

Processes and Events. The encoding of processes and events follows the approach outlined earlier, respectively, for durative and instantaneous actions. However, their triggering is defined by PDDL+'s *must* semantics, which prescribes that they are triggered as soon as their preconditions are true. In CASP, this is captured by a choice rule combined with numerical constraints. Intuitively, when the Boolean conditions of the process are satisfied, the choice rule states the process will start unless it is inhibited by unsatisfied numerical conditions. Constraints enforced on the numerical conditions capture the latter case. Consider a process corresponding to a falling object, with preconditions $\neg held$ and $height > 0$. The choice rule:

$$1\{occurs(start(falling),I), is\_false(height > 0,I)\}1 \leftarrow holds(\neg held,I).$$

entails two possible, equally likely, outcomes: the object will either start falling, or be prevented from doing so by the fact that condition $height > 0$ is false. The second outcome is possible only if the height is indeed not greater than 0, which is enforced by the constraint:

$$v\_final(height,I) \leq 0 \leftarrow is\_false(height > 0,I).$$

Given an arbitrary process, the corresponding choice rule lists an atom $is\_false(\cdot,I)$ for every numerical condition, and the encoding includes a constraint on the value of $v\_final(n,I)$ corresponding to the complement of that condition. The treatment of events is similar.
The encoding is completed by the following statements:

$$start(I+1) = end(I).$$

$$v\_initial(F,I+1) = v\_final(F,I).$$

$$holds(F,I+1) \leftarrow holds(F,I), \text{not } holds(\neg F,I+1).$$
$$holds(\neg F,I+1) \leftarrow holds(\neg F,I), \text{not } holds(F,I+1).$$

The first rule ensures that there are no gaps between the time intervals associated with consecutive states. The others handle fluent propagation from a state to the next.

3.2 Problem Encoding

The problem portion of the PDDL+ problem is encoded as follows.

Initial state. The encoding of the initial state consists of a set of rules specifying the values of fluents in $P \cup v$ at step 0.

Goals. The encoding of a goal consists of a set of denials on Boolean fluents and of constraints on numerical fluents, obtained similarly to the encoding of preconditions of actions, discussed earlier.

Given a PDDL+ planning instance I, by $\Pi(I)$ we denote the CASP encoding of I. Next, we turn our attention to the planning task.

3.3 Planning Task

Our approach to planning leverages techniques from ASP-based planning (Lifschitz 2002; Balduccini, Gelfond, and Nogueira 2006). The planning task is specified by the planning module, M, which consists of the single rule:

$$\{occurs(A,I), occurs(start(D),I)\}.$$

where A,D are variables ranging over instantaneous actions and durative actions, respectively. The rule intuitively states that any action may occur (or start) at any time step.

It can be shown that the plans for a given maximum time step for a PDDL+ planning instance I are in one-to-one correspondence with the CASP solutions of $\Pi(I) \cup M$. The plan encoded by a CASP solution A can be easily obtained from the atoms of the form $occurs(a,i)$ and from the value assignments to numerical variables $start(i)$ and $end(i)$.

It is also worth noting the level of modularity of our approach. In particular, it is straightforward to perform other reasoning tasks besides planning (e.g., a hybrid of planning and diagnostics is often useful for applications) by replacing the planning module by a different one, as demonstrated for example in (Balduccini and Gelfond 2003b).

4 Case Study

For our case study, we have focused on a specific instance of CASP, called EZCSP (Balduccini 2009; Balduccini and Lierler 2013). In EZCSP, numerical constraints are encoded as arguments of the special relation *required*, e.g. $required(start(I+1) = end(I))$. Encodings of the *generator* (Bogomolov et al. 2014) and *car* domains (Bryce et al. 2015) were created as described above, and the architecture of the EZCSP solver was expanded to ensure soundness of the algorithm (see below). The complete encodings are omitted due to space considerations. Rather, to illustrate our approach, we present a fragment of the encoding of process *generate* from the *generator* domain, whose PDDL+ representation is shown in Figure 2. The fragment captures the invariants and the change of fuel level. The process has two continuous effects: it decreases the fuel level (the expression `(* #t 1)` states that the change is continuous and linear with respect to time) and increases the value of variable `generator_time`, which keeps track of how long the generator ran. The choice of *generate* was motivated by the fact that the representation of processes is arguably one of the most challenging aspects of encoding PDDL+ in CASP. The invariant on the maximum fuel level is encoded by two EZCSP rules (atom $tankcap(\cdot)$ determines the capacity of the tank):

$$required(v\_initial(fuel\_level,I) \leq TC) \leftarrow tankcap(TC).$$

$$required(v\_final(fuel\_level,I) \leq TC) \leftarrow tankcap(TC).$$

The (negative) contribution to the generator's fuel level is modeled by:

$$required(v(contrib(fuel\_level,generate),I) = -1 * (end(I) - start(I))) \leftarrow holds(inprogr(generate),I).$$

From an algorithmic perspective, the EZCSP solver

```
(:process generate
 :parameters (?g - generator)
 :condition
  (and
   (over all
    (>= (fuelLevel ?g) 0)
   )
   (over all
    (<= (fuelLevel ?g) (capacity ?g))
   )
  )
 :effect
  (and
   (decrease (fuelLevel ?g) (* #t 1))
   (increase (generator_time ?g)
             (* #t 1))
   )
 )
```

Figure 2: PDDL+ process from the Generator domain

computes CASP solutions of a program Π by iteratively (1) using an ASP solver to find an answer set A of Π, and (2) using a constraint solver to find the solutions of the CSP encoded by A. To account for the discretize and validate approach mentioned earlier, we have extended the EZCSP solver with a validation step. In the extended architecture, shown in Figure 3, if step (2) is successful, the tool VAL is called to validate the plan before returning it. If VAL finds the plan not to be valid, it returns which invariant was violated and at which timepoint. If that happens, the *expansion* process occurs, where the encoding is expanded with (1) new numerical variables that represent the value of the involved numerical fluents at that timepoint, and (2) numerical constraints enforcing the invariant on them. The CASP solutions for the new encoding are computed again[2], and the process is iterated until no invariants are violated.

To illustrate the expansion process, let us consider a durative action d causing fluent f to increase by $\iota(\Delta)$, where Δ is elapsed time. Suppose invariant $f < c$ is violated at a timepoint t that falls within the time interval associated with time step i. The encoding is then expanded by:

$$
\begin{aligned}
&required(v'(F,i) = \\
&\qquad v\_initial(F,i) + v'(contrib(F,s),i)).
\end{aligned}
$$

$$
\begin{aligned}
&required(\\
&\qquad v'(contrib(F,s),i)) = \iota(t - start(i)) \\
&\quad) \leftarrow holds(inprog(d),i).
\end{aligned}
$$

$$required(v'(F,i) < c).$$

5 Experimental Results

We performed an empirical evaluation of the performance achieved with our approach. The comparison was with the state-of-the-art PDDL+ planners dReal (Bryce et al. 2015) and UPMurphi. Although SpaceEx (Bogomolov et al. 2014) is indeed a related approach, it was not included in the preliminary comparison because it is focused on proving only plan non-existence. The experimental setup used a virtual machine running in VMWare Workstation 12 on a computer with an i7-4790K CPU at 4.00GHz. The virtual machine was assigned a single core and 4GB RAM. The operating system was Fedora 22 64 bit. The version of EZCSP used was 1.7.4[3], with gringo 3.0.5[4] and clasp 3.1.3[5] as grounding tool and ASP solver, and B-Prolog 7.5[6] and GAMS 24.5.7[7] as constraint solvers. The former was used for all linear problems and the latter for the non-linear ones. The other systems used were dReal 2.15.11[8], configured as suggested by its authors, and UPMurphi 3.0.2[9].

The experiments were conducted on the linear and non-linear versions of the *generator* and *car* domains.

The comparison with dReal was based on finding a single plan with a given maximum time step, as discussed in (Bryce et al. 2015). The results are summarized in Table 1. The comparison with UPMurphi was based on the cumulative times for finding a single plan by progressively increasing the maximum time step. The results are reported in Table 2. In the tables, entries marked "-" indicate a timeout (threshold 600 sec). Entries marked "*" indicate missing entries due to licensing limitations (see below). It should be noted that none of the instances triggered the expansion process described in the previous section, given that all plans were found to be valid by VAL. Next, we discuss the experimental results obtained for each domain.

Generator. Our encoding uses Torricelli's law ($v = \sqrt{2gh}$) to model the transfer of liquid. This is a more complex model than the one used in the dReal encoding, but is more physically accurate. The instances were generated by increasing the number of refuel tanks from 1 to 8. The CASP encoding was as discussed above, and included a single, encoding-level heuristic stating that action $start(generate)$ must occur during the first state transition and at timepoint 0. (dReal includes multiple heuristics that are hard-coded in the solver.)

The execution times for EZCSP for a fixed maximum time step (Table 1) ranged between 0.28 sec and 261.89 sec for the linear variant, and between 0.72 sec and 256.59 sec for the non-linear one. The non-linear variant was only tested up to instance 7 because of limitations of the free version of GAMS. In both the linear and non-linear case, the EZCSP encoding was substantially faster than dReal. Especially remarkable is the fact that, in both cases, dReal timed out on all instances except for the first one.

The cumulative times for EZCSP (Table 2) ranged between 0.89 sec and 292.22 sec for the linear case, with no timeouts. In the non-linear case, the times were between 1.44 sec and 267.11 sec, with a timeout in instance 8. UPMurphi did not scale as well. In the linear case, only instances 1-3 were solved, and resulted in times ranging between 2.02 sec and 91.80 sec. The speedup yielded by EZCSP reached about one order of magnitude before UPMurphi began to time out. In the non-linear case, UPMurphi timed out in all instances.

Car. The version of the car domain we used is the same that was adopted in (Bryce et al. 2015). In this domain, a vehicle needs to travel a certain goal distance from its start position. The vehicle is initially at rest. Two actions allow the vehicle to accelerate and to decelerate. The goal is achieved when the vehicle reaches the desired distance and its speed is 0. In the linear variant, accelerating increases the velocity by 1 and decelerating decreases it by 1. In the non-linear variant, accelerating increases the acceleration by 1, and similarly for decelerating. The velocity is influenced by the acceleration according to the usual laws of physics. The calculation also takes into account a drag factor equal to $0.1 \cdot v^2$. The instances were obtained by progressively increasing the range of allowed accelerations (velocities in the linear version) from $[-1,1]$ to $[-8,8]$. The CASP encoding leveraged no heuristics and, as discussed earlier, the underlying solvers are completely general-purpose.

[2]Only the solutions of the CSP need to be recomputed.

[3]`http://mbal.tk/ezcsp/`

[4]`http://sourceforge.net/projects/potassco/files/gringo/`

[5]`https://sourceforge.net/projects/potassco/files/clasp/`

[6]`http://www.picat-lang.org/bprolog/`

[7]`http://www.gams.com/`

[8]`http://dreal.github.io/`

[9]`https://github.com/gdellapenna/UPMurphi/`

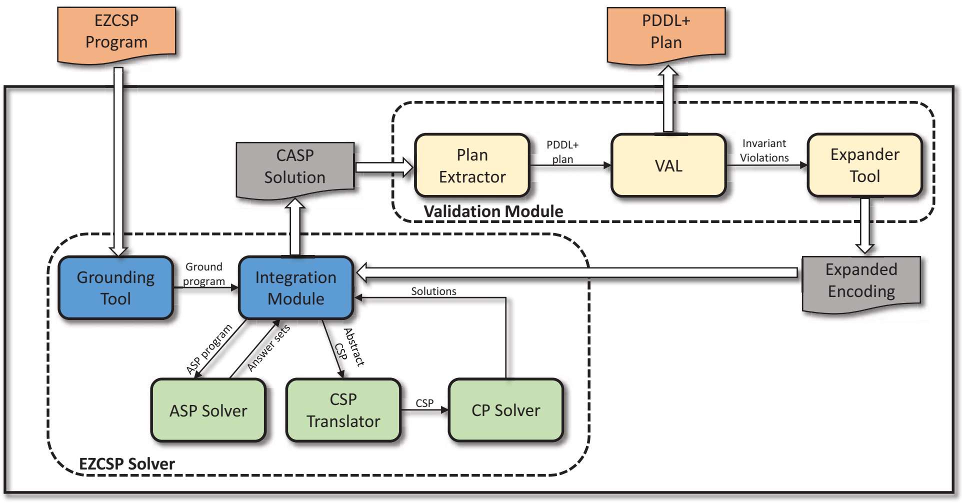


Figure 3: Extended Solver Architecture

Domain	**Solver**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Gen linear	EZCSP	0.28	1.03	4.21	7.25	27.08	43.42	54.83	261.89
	dReal	3.73	-	-	-	-	-	-	-
Gen non-linear	EZCSP	0.72	1.62	0.68	1.05	87.95	256.59	238.93	*
	dReal	8.18	-	-	-	-	-	-	-
Car linear	EZCSP	0.32	0.31	0.32	0.32	0.32	0.30	0.31	0.31
	dReal	1.11	1.11	1.15	1.14	1.19	1.13	1.14	1.19
Car non-linear	EZCSP	0.71	0.68	0.29	0.39	0.25	0.25	0.26	0.84
	dReal	58.21	162.60	-	-	-	-	-	-

Table 1: Fixed time step. Results in seconds. Problem instances refer to number of tanks (*generator*) and max acceleration (*car*).

Domain	**Solver**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Gen linear	EZCSP	0.89	1.92	5.46	9.93	30.79	50.25	67.97	292.22
	UPMurphi	2.02	12.75	91.80	-	-	-	-	-
Gen non-linear	EZCSP	1.44	2.44	13.10	53.70	88.58	267.11	250.03	-
	UPMurphi	-	-	-	-	-	-	-	-
Car linear	EZCSP	1.01	0.98	1.04	0.99	0.91	0.85	0.88	0.83
	UPMurphi	0.40	0.38	0.38	0.38	0.41	0.39	0.40	0.41
Car non-linear	EZCSP	2.32	1.49	1.14	1.85	1.14	1.18	1.06	2.13
	UPMurphi	184.88	-	-	-	-	-	-	-

Table 2: Cumulative times. Results in seconds. Problem instances refer to number of tanks (*generator*) and max acceleration (*car*).

As shown in Table 1, the execution times for EZCSP were around 0.30 sec for the linear case, and between 0.25 sec and 0.84 sec for the non-linear one. These times are about 3 times faster than dReal in the linear case and orders of magnitude better in the non-linear case, where dReal times out in instances 3-8. The scalability of EZCSP appears to be excellent, with no significant growth.

The comparison with UPMurphi on cumulative times shows some interesting behavior. In the linear case, EZCSP is, in fact, about 2.5 times slower than UPMurphi. The former has times ranging between 0.83 sec and 1.04 sec, while UPMurphi's times are between 0.38 sec and 0.41 sec. On the other hand, EZCSP outperforms UPMurphi in the non-linear case, with all instances solved in times between 1.06 sec and 2.32 sec, while UPMurphi only solves the first instance with a time of 184.88 sec, i.e., nearly 2 orders of magnitude slower than EZCSP.

We believe the empirical results demonstrate the promise of our approach. From the perspective of the underlying solving algorithms, it is worth stressing that the better results of EZCSP over dReal are especially remarkable given that the latter employs planning-specific heuristics, while the EZCSP solver and its components are not specialized for a given reasoning task.

6 Conclusions

In this paper we have presented a new approach to PDDL+ planning based on CASP languages that provides a solid basis for applying logic programming to PDDL+ planning. Experiments on well-known domains, some involving non-linear continuous change, have shown that our approach outperforms comparable state-of-the-art PDDL+ planners.

Although other CASP solvers exist, EZCSP is, to the best of our knowledge, the only one supporting both non-linear constraints, required for modeling non-linear continuous change, and real numbers.

ACSOLVER (Mellarkod, Gelfond, and Zhang 2008b) implements an eager approach to CASP solving, where (in contrast to the lazy approach of EZCSP) ASP and CSP solving are tightly coupled and interleaved. It does not support non-linear or global constraints, but allows for real numbers.

CLINGON (Ostrowski and Schaub 2012b) is another tightly coupled CASP solver. The available implementation, however, is not broadly applicable to the kinds of problems considered in this paper. In fact, CLINGON does not support non-linear constraints and real numbers. On the other hand, differently from EZCSP, it allows for numerical constraints both in the head of rules and in their bodies.

A high level view of the languages and solving techniques employed by these solvers can be found in (Lierler 2014).

Finally, it is also worth noting that basing our approach on CASP makes it amenable to be expanded to handle uncertainty about the initial situation or the effects of actions (e.g., (Morales, Tu, and Son 2007)). Another interesting possibility is the use of PDDL+ domain descriptions, translated to CASP, for both planning and diagnosis, along the lines of the approach applied in (Balduccini and Gelfond 2003a) to ASP domain descriptions.

References

[Balduccini and Gelfond 2003a] Balduccini, M., and Gelfond, M. 2003a. Diagnostic reasoning with A-Prolog. *Journal of Theory and Practice of Logic Programming (TPLP)* 3(4–5):425–461.

[Balduccini and Gelfond 2003b] Balduccini, M., and Gelfond, M. 2003b. Logic Programs with Consistency-Restoring Rules. In Doherty, P.; McCarthy, J.; and Williams, M.-A., eds., *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, 9–18.

[Balduccini and Lierler 2013] Balduccini, M., and Lierler, Y. 2013. Integration Schemas for Constraint Answer Set Programming: a Case Study. *Theory and Practice of Logic Programming (TPLP), On-line Supplement*.

[Balduccini, Gelfond, and Nogueira 2006] Balduccini, M.; Gelfond, M.; and Nogueira, M. 2006. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence* 47(1–2):183–219.

[Balduccini 2009] Balduccini, M. 2009. Representing Constraint Satisfaction Problems in Answer Set Programming. In *ICLP09 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP09)*.

[Baselice, Bonatti, and Gelfond 2005] Baselice, S.; Bonatti, P. A.; and Gelfond, M. 2005. Towards an Integration of Answer Set and Constraint Solving. In *Proceedings of ICLP 2005*.

[Bogomolov et al. 2014] Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2228–2234.

[Bogomolov et al. 2015] Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015), Jerusalem, Israel, June 7-11, 2015.*, 42–46.

[Bryce et al. 2015] Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. Smt-based nonlinear PDDL+ planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 3247–3253.

[Chintabathina, Gelfond, and Watson 2005] Chintabathina, S.; Gelfond, M.; and Watson, R. 2005. Modeling Hybrid Domains Using Process Description Language. In *Proceedings of ASP '05 – Answer Set Programming: Advances in Theory and Implementation*, 303–317.

[Coles et al. 2012] Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

[Della Penna et al. 2009] Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A tool for universal planning on PDDL+ problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI.

[Fox and Long 2006] Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27:235–297.

[Fox, Howey, and Long 2004] Fox, M.; Howey, R.; and Long, D. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 294–301.

[Gelfond and Lifschitz 1991] Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

[Gelfond and Lifschitz 1993] Gelfond, M., and Lifschitz, V. 1993. Representing Action and Change by Logic Programs. *Journal of Logic Programming* 17(2–4):301–321.

[Henzinger 1996] Henzinger, T. A. 1996. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, 278–292.

[Kautz and Selman 1992] Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

[Li and Williams 2008] Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 206–213. AAAI.

[Lierler 2014] Lierler, Y. 2014. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence* 207:1–22.

[Lifschitz 1999] Lifschitz, V. 1999. *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, Berlin. chapter Action Languages, Answer Sets, and Planning, 357–373.

[Lifschitz 2002] Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138:39–54.

[McDermott 2003] McDermott, D. V. 2003. Reasoning about autonomous processes in an estimated-regression planner. In Giunchiglia, E.; Muscettola, N.; and Nau, D. S., eds., *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, 143–152. AAAI.

[Mellarkod, Gelfond, and Zhang 2008a] Mellarkod, V. S.; Gelfond, M.; and Zhang, Y. 2008a. Integrating Answer Set Programming and Constraint Logic Programming. *Annals of Mathematics and Artificial Intelligence*.

[Mellarkod, Gelfond, and Zhang 2008b] Mellarkod, V. S.; Gelfond, M.; and Zhang, Y. 2008b. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence* 53(1-4):251–287.

[Morales, Tu, and Son 2007] Morales, R.; Tu, P. H.; and Son, T. C. 2007. An Extension to Conformant Planning Using Logic Programming. In Veloso, M. M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 1991–1996.

[Nau, Ghallab, and Traverso 2004] Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Ostrowski and Schaub 2012a] Ostrowski, M., and Schaub, T. 2012a. ASP Modulo CSP: The Clingcon System. *Journal of Theory and Practice of Logic Programming (TPLP)* 12(4–5):485–503.

[Ostrowski and Schaub 2012b] Ostrowski, M., and Schaub, T. 2012b. ASP modulo CSP: the clingcon system. *TPLP* 12(4-5):485–503.

[Penberthy and Weld 1994] Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In Hayes-Roth, B., and Korf, R. E., eds., *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 1010–1015. AAAI Press / The MIT Press.

[Reiter 2001] Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dy- namical Systems*. MIT Press.

[Rossi, van Beek, and Walsh 2006] Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier.

[Shin and Davis 2005] Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166(1-2):194–253.

A multi-objective memetic algorithm for solving job shops with a non-regular energy cost

Miguel A. González
University of Oviedo
Spain
mig@uniovi.es

Angelo Oddi
ISTC-CNR
Italy
angelo.oddi@istc.cnr.it

Riccardo Rasconi
ISTC-CNR
Italy
riccardo.rasconi@istc.cnr.it

Abstract

In this work, we tackle a bi-objective version of the job shop scheduling problem where the objectives to be minimized are the total weighted tardiness and the energy cost. Taking into account energy costs in the schedules is very interesting in real production environments, as the increasing energy prices and requirements to reduce carbon footprint are an important issue nowadays for sustainable manufacturing. We propose a multi-objective memetic algorithm that hybridizes an enhanced version of the NSGA-II dominance-based evolutionary algorithm with a multi-objective local search based on hill-climbing. Given the non-regularity of the energy consumption objective function, we design a low-polynomial energy post-optimization procedure which attempts to reduce the energy cost of a solution without increasing its total weighted tardiness. We report results from an experimental study where we analyse our method and demonstrate that the proposed post-optimization procedure adds a significant improvement in its performance, obtaining results that outperform those of the state-of-the-art.

1 Introduction

The Job Shop Scheduling Problem (JSP) has been a research topic over the last decades due to the fact that it is a simple model of many real production processes.

The importance of due date related performance criteria has been widely recognized in many real production environments. A survey reported in (Panwalkar, Dudek, and Smith 1973) has found that meeting due dates is identified as the most important scheduling objective in competitive markets. The total weighted tardiness (TWT) is a due date related objective that can assign different priorities to different operations, and because of its usefulness, its minimization is the subject of a large amount of literature in scheduling.

In particular, the JSP with TWT minimization was first considered in (Singer and Pinedo 1998), (Singer and Pinedo 1999) and (Kreipl 2000). The first paper proposes an exact branch and bound algorithm, the second proposes a shifting bottleneck algorithm and the third one proposes a large step random walk heuristic. Other approaches include the genetic local search given in (Essafi, Mati, and Dauzère-Pérès 2008), the local search proposed in (Mati, Dauzere-Peres, and Lahlou 2011), the hybrid shifting bottleneck-tabu search heuristic found in (Bülbül 2011), or the genetic algorithm combined with tabu search of (González et al. 2012). Also, in (Kuhpfahl and Bierwirth 2016) some sophisticated and time-consuming neighborhood structures are proposed.

The increasing price of energy, as well as the emission reduction needs, are forcing manufacturing enterprises to put more and more efforts towards the reduction of consumption and the study of energy-saving opportunities and best practices. In particular, the job shop with energy considerations is receiving an increasing attention. Existing approaches include the genetic algorithm proposed in (Liu et al. 2014) which tries to minimize both the weighted tardiness and the energy consumption in a job shop, or the genetic-simulated annealing method of (Dai et al. 2013), that solves a flexible flow shop scheduling problem with energy considerations.

Clearly, when the improvement in energy consumption must not be obtained at the cost of losing performance quality in the solutions, we face a bi-objective scheduling problem. There is a growing interest in multi-objective optimization for scheduling and, given its complexity, in the use of metaheuristic techniques to solve these problems (Dabia et al. 2013).

In the single-objective case, it is common to hybridize evolutionary algorithms with local search to produce memetic algorithms, which benefit from the synergy between their components to provide a better search capacity. It is possible to find various multi-objective memetic algorithms in the literature, some of them applied to manufacturing problems (Ishibuchi et al. 2009). However, according to (Liefooghe et al. 2012), the number of multi-objective local search algorithms proposed so far is still reduced. In fact, the main difficulty in designing multi-objective memetic algorithms is the implementation of the local search, which essentially is a single-objective optimization technique.

In this paper we propose a memetic algorithm to minimize both the TWT and the energy consumption in a job shop. Our proposal hybridizes several techniques:

- A version of the NSGA-II dominance-based evolutionary algorithm with a mechanism to penalize repeated individuals in the population.
- A multi-objective local search based on hill-climbing.

- A low-polynomial energy post-optimization procedure which attempts to reduce the energy cost of a solution.
- An optimal linear programming approach to further reduce the energy cost of a solution.

The effectiveness of our method is analysed in the experimental study, and its results are compared with those of the state-of-the-art in this problem, which is the NSGA-II proposed in (Liu et al. 2014). We have to remark that, even though our proposal is also a NSGA-II algorithm, its crossover operator and replacement strategy are different, and moreover it is hybridized with additional components that lead to a much better performance, as we will see in the experimental study.

The remainder of the paper is organized as follows. In Section 2 we formulate the problem. In Section 3 we define the proposed multi-objective memetic algorithm. In Section 4 we report the results of the experimental study, and finally Section 5 summarizes the main conclusions of this paper.

2 Problem formulation

In the job shop scheduling problem, a set of N jobs, $J = \{J_1, \dots, J_N\}$, are to be processed on a set of M machines or resources, $R = \{R_1, \dots, R_M\}$. Each job J_i consists of a sequence of n_i operations $(\theta_{i1}, \dots, \theta_{in_i})$, where each operation requires the uninterrupted and exclusive use of a given resource during all its processing time. The objective is to minimize some function of the completion times of the jobs, subject to the following constraints: (i) the sequence of machines for each job is prescribed, and (ii) each machine can process at most one operation at a time. Jobs may also have a due date, that is, a time before which all operations of the job should be completed, and a weight, which represents its relevance. In order to simplify expressions, instead of using θ_{ij}, in the following we are denoting operations by a single letter whenever possible. We denote by:

- Ω the set of operations
- d_i the due-date of job J_i
- w_i the weight of job J_i
- P_k^{idle} the idle power level of machine R_k
- p_u the processing time of operation u
- s_u the starting time of operation u that needs to be determined

The JSP has two binary constraints: precedence and capacity. Precedence constraints, defined by the sequential routings of the operations within a job, translate into linear inequalities of the type: $s_u + p_u \leq s_v$, where v is the next operation to u in the job sequence. Capacity constraints that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the form: $s_u + p_u \leq s_v \vee s_v + p_v \leq s_u$, where u and v are operations requiring the same machine.

The objective here is to obtain a feasible schedule, i.e. a starting time for each one of the operations such that all constraints are satisfied.

We are minimizing two objective functions: the total weighted tardiness (TWT) and the energy consumption.

The TWT is the weighted cost of the jobs exceeding its due-dates, and is defined as follows

$$\sum_{i=1,\dots,N} w_i T_i \tag{1}$$

T_i is the tardiness of the job i, given by

$$T_i = \max\{C_i - d_i, 0\} \tag{2}$$

where C_i is the completion time of job i.

The energy consumption model is taken from (Liu et al. 2014), where it is proven that the objective to reduce the total electricity consumption of a job shop can be converted to reduce the total non-processing energy (NPE), i.e. the amount of time a machine is on and not executing a job. Notice that each machine must process a fixed set of operations, and all these operations have fixed durations, and therefore for any schedule the processing energy must be equal.

Hence, the objective function can be set as the sum of all the NPE consumed by all the machines in a job shop to carry out a given job schedule. Then, the total NPE is defined as

$$\sum_{k=1,\dots,M} TEM_k \tag{3}$$

where TEM_k is the NPE of machine R_k, given by

$$TEM_k = P_k^{idle} \times (s_{\omega_k} + p_{\omega_k} - s_{\alpha_k} - \sum_{u \in M_k} (p_u)) \tag{4}$$

where M_k is the set of operations that must be executed in the resource R_k, and α_k and ω_k are the first and the last operation respectively on machine R_k in the considered schedule.

The TWT is a regular performance measure (Baker 1974), which means that its value can be increased only by increasing the completion time of a job. To minimize a regular measure, it is sufficient to only consider "left-shift schedules"; i.e. schedules built from a partial ordering of the operations, so that each operation starts as soon as possible after all the preceding operations in the partial ordering. On the other hand, the NPE is a non-regular performance measure. Notice that, given a schedule, the NPE of a machine R_k is reduced if we are able to delay the starting time of its first operation (s_{α_k}) without increasing the starting time of its last operation (s_{ω_k}).

2.1 The disjunctive graph model representation

The disjunctive graph is a common representation in scheduling, its exact definition depending on the particular problem. For our problem we use a similar representation as other papers in the literature (see (Kreipl 2000; Mati, Dauzere-Peres, and Lahlou 2011; Essafi, Mati, and Dauzère-Pérès 2008; González et al. 2012; Kuhpfahl and Bierwirth 2016)). In particular, we propose a directed graph $G = (V, A \cup E)$. Each node in set V represents an operation of the problem, with the exception of the dummy nodes *start* and end_i $1 \leq i \leq N$, which represent fictitious operations that do not require any machine. Arcs in A are called *conjunctive arcs* and represent the precedence constraints between the operations of each job. Additionally,

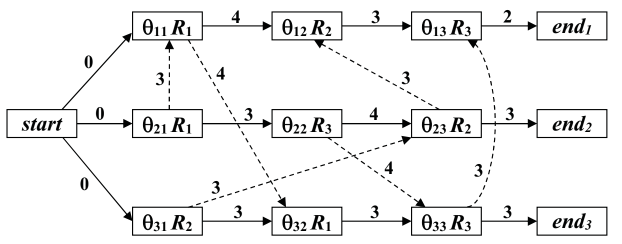


Figure 1: A feasible schedule to a problem with 3 jobs and 3 machines

there are arcs from node *start* to the first operation of each job, and also arcs from the last operation of each job i to its corresponding node end_i. Arcs in E are called *disjunctive arcs* and represent capacity constraints. Set E is partitioned into subsets E_i, with $E = \cup_{j=1,\dots,M} E_j$, where E_j corresponds to resource R_j and includes two directed arcs (v, w) and (w, v) for each pair v, w of operations requiring that resource. Each arc (v, w) in A and in E is weighted with the processing time of the operation at the source node, p_v (note that $p_{start} = 0$ and $p_{end_i} = 0$).

A feasible schedule S is represented by an acyclic subgraph of G: $G_S = (V, A \cup H)$, where $H = \cup_{j=1\dots M} H_j$, H_j being a minimal subset of arcs of E_j defining a processing order for all operations requiring R_j. Finding a solution can thus be reduced to discovering compatible orderings H_j, or partial schedules, that translate into a solution graph G_S without cycles. Given a feasible schedule, PJ_v and SJ_v denote respectively the predecessor and successor of v in the job sequence, and PM_v and SM_v the predecessor and successor of v in its machine sequence. Figure 1 shows a solution to a problem with 3 jobs and 3 machines; dotted arcs belong to H, while continuous arcs belong to A.

The TWT of a schedule S is determined by a set of critical paths in G_S. A critical path is defined as a largest cost path from node $start$ to a node end_i $1 \le i \le N$. The length of this path is the completion time of the operation end_i and so it determines the contribution of job J_i to the solution cost. Nodes and arcs in a critical path are also termed critical. A critical path may be represented as a sequence of the form $start, B_1, \dots, B_r, end_i$, $1 \le i \le N$, where each B_k, $1 \le k \le r$, is a critical block, a maximal subsequence of consecutive operations in the critical path requiring the same machine. These concepts are of major importance for job scheduling problems due to the fact that most formal properties, solution methods and neighborhood structures rely on them. The neighborhood structure used in this paper relies on reversing the processing order of operations in a critical block, as similarly done in (Van Laarhoven, Aarts, and Lenstra 1992).

On the other hand, notice that this disjunctive graph representation is not that useful for NPE minimization, as this measure does not directly depend on finding largest cost paths in a graph representation.

3 The multi-objective approach

To optimize the two objective functions defined in Section 2, we shall use a dominance-based approach. In general, for a minimization problem with f_i, $i = 1, \dots, n$ objective functions, a solution S is said to be *dominated* by another solution S', denoted $S' \succ S$ if and only if for each objective function f_i, $f_i(S') \le f_i(S)$ and there exists at least one objective function such that $f_i(S') < f_i(S)$. Our goal will then be to find non-dominated solutions to our problem with respect to TWT and NPE. To achieve this, we propose a dominance-based hybrid method, combining a multi-objective evolutionary algorithm with a multi-objective hill climbing local search and a linear programming approach.

We have seen that the NPE is a non-regular objective function. Some papers have already considered minimizing non-regular objectives in a job shop. As an example, (Brandimarte and Maiocco 1999) tackles a single-objective case and proposes to decompose the overall problem into sequencing and timing subproblems. We follow a similar approach, in the sense that we represent the solutions as permutations in the genetic algorithm and in the local search in order to solve the sequencing subproblem. To solve the timing subproblem we introduce a low-polynomial energy post-optimization procedure when evaluating each solution, and also a more computationally expensive optimal linear programming approach to further improve the NPE of the final set of non-dominated solutions.

3.1 Multi-objective evolutionary algorithm

Our proposal is based on the well-known NSGA-II template for a dominance-based evolutionary algorithm (Deb et al. 2002). Roughly speaking, an initial population Pop_0 of size $populationSize$ is randomly created and evaluated and then the algorithm iterates over $numGenerations$ generations, keeping a set of non-dominated solutions. At each iteration i a new population $\mathrm{Off}(Pop_i)$ is built from the current one Pop_i by applying the genetic operators of selection, crossover and mutation, and finally a replacement strategy is applied to obtain the next generation Pop_{i+1}.

Representation Solutions are codified into chromosomes using permutations with repetition, as introduced in (Bierwirth 1995) for the JSP. This is a permutation of the set of operations, each being represented by its job number, which represents a linear ordering compatible with precedence constraints. For example, if we have a problem instance with 3 jobs: $J_1 = \{\theta_{11}, \theta_{12}\}$, $J_2 = \{\theta_{21}, \theta_{22}, \theta_{23}, \theta_{24}\}$, $J_3 = \{\theta_{31}, \theta_{32}, \theta_{33}\}$, then the ordering of operations $\pi = \{\theta_{21}, \theta_{11}, \theta_{22}, \theta_{31}, \theta_{23}, \theta_{32}, \theta_{33}, \theta_{24}, \theta_{12}\}$ is represented by the chromosome $v = (2\ 1\ 2\ 3\ 2\ 3\ 3\ 2\ 1)$.

Evaluating a chromosome A given chromosome is evaluated by generating an associated schedule and then computing its TWT and NPE. To do this, each operation is scheduled using an insertion strategy following the sequence given by the chromosome, which is a method commonly used (Palacios et al. 2014). More precisely, given an operation u that is to be scheduled, we define a *feasible insertion interval* as a time interval $[t_S, t_E]$ in which its required machine

Algorithm 1 The energy post-optimization procedure

```
Require: A problem instance $Ins$ and a feasible schedule
    (an ordering $O$ and a set of starting times $s$)
  $k \leftarrow |\Omega|$
  while $k \geq 1$ do
    $a = O[k]$;
    if $a$ is the last operation processed in a machine then
      $s'_a = s_a$;
    else
      if $a$ is the last operation of its job then
        $j \leftarrow$ job of operation $a$;
        $s'_a = \min\{\max\{d_j, s_a + p_a\}, s'_{SM_a}\} - p_a$;
      else
        $s'_a = \min\{s'_{SJ_a}, s'_{SM_a}\} - p_a$;
      end if
    end if
    $k \leftarrow k - 1$;
  end while
  return The new set of starting times $s'$ for the ordering
    $O$ (the TWT is the same and the NPE is lower or equal)
```

is idle and such that u can be processed within that time interval without violating precedence constraints. In our case, this means that $t_S + p_u \leq t_E$, and $t_S \geq s_{PJ_u} + p_{PJ_u}$ (if u is the first operation of its job, then s_{PJ_u} and p_{PJ_u} are taken to be 0). Then, s_u is given the t_S value of the earliest feasible insertion interval for u, and hence its completion time is $s_u + p_u$.

Energy post-optimization procedure The procedure described before for evaluating a chromosome would be able to produce a Pareto optimal schedule if the objective functions were regular. However, as we have pointed in Section 2, the NPE is a non-regular objective function, and therefore scheduling each operation as soon as possible may not be the best option. For this reason, we propose an additional energy post-optimization procedure that, given a schedule, is able to reduce the NPE while not increasing the TWT.

Basically, the idea of this procedure is to delay all the operations of each machine as much as possible, with the exception of the last one, increasing the tardiness of none of the jobs. The procedure is detailed in Algorithm 1. Notice that, by not increasing the starting times of the last operation of each machine, we ensure that the resulting NPE after applying the procedure is lower than or equal to the original one. Furthermore, the TWT of the resulting solution is not increased either, as the completion time of the last operation of a job may only be delayed if it is lower than the due date, and in this case it is at most delayed up to this due date.

This procedure is executed inside the solution evaluation method, just after the schedule is built. Therefore, it is applied to evaluate every chromosome generated in the genetic algorithm and every neighbor considered in the local search.

The time taken by adding this procedure to the scheduler is quite reasonable; it increases by 25% the execution time of the memetic algorithm, but at the same time the results significantly improve, as we will see in Section 4.

Genetic operators The selection phase selects the chromosomes that will undergo crossover and mutation, and is based on a tournament strategy. In particular, we select $tournamentSize$ chromosomes at random and choose the best one to be the first parent, according to non-domination rank and crowding distance (see next section). Then, we select another $tournamentSize$ chromosomes at random and choose the best one to be the second parent. Finally, the crossover operator is applied to these two parents with probability $crossoverProb$ to obtain two offspring solutions.

For chromosome mating we have considered the Job Order Crossover (JOX) (Bierwirth 1995). Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so as to maintain their relative ordering. A second offspring is generated inverting the role of the parents.

In order to preserve the diversity of individuals and prevent the algorithm from getting stuck in local optima, a mutation strategy is also introduced. Just after the crossover, each offspring is mutated with probability $mutationProb$. In particular we use the swap mutation operator, which swaps two positions of the chromosome chosen at random.

Replacement strategy The replacement strategy establishes how population Pop_i of size $populationSize$ and population $\text{Off}(Pop_i)$ that results from applying selection, crossover and mutation to Pop_i are combined to generate the new population Pop_{i+1} for the next iteration of the algorithm. Here we adopt a strategy based on the non-dominated sorting approach with diversity preservation from (Deb et al. 2002). Initially, for each individual j in the pool $Pop_i \cup \text{Off}(Pop_i)$ a non-domination rank ($rank(j)$) and a crowding distance ($dist(j)$) are calculated. The former sorts the pool into different non-domination levels while the latter estimates the density of solutions in the area of the non-domination level where the individual lies. Population Pop_{i+1} is then formed by the best $populationSize$ individuals from the pool $Pop_i \cup \text{Off}(Pop_i)$ according to the lexicographical order defined by $(rank, dist)$. That is, solutions belonging to a lower (better) non-domination rank are preferred and, between two solutions in the same non-dominance level, we prefer the solution located in the less crowded region.

In order to provide greater diversity to the algorithm, we have included an additional step in the above strategy. Specifically, we propose to start by removing from the pool of individuals $Pop_i \cup \text{Off}(Pop_i)$ those which are repeated, in the sense that there exists in the pool at least another individual having identical values for all objective functions.

Only after this elimination is the above strategy based on $(rank, dist)$ applied. In the case that such elimination causes the pool to contain less than $populationSize$ individuals, all the non-repeated individuals pass onto the next generation Pop_{i+1}, which is later completed with some of the repeated individuals. To do that, we first remove the repeated individuals that are in the pool of repeated individuals, and then the remaining ones are sorted again into non-domination ranks and crowding distance, and the best repeated individuals according to $(rank, dist)$ are selected. If needed, these procedure is repeated until the pool contains at least $populationSize$ individuals.

3.2 Local search

Local search is often used in combination with other metaheuristics in such a way that the local search provides exploitation while the other metaheuristic provides exploration. As we have already mentioned in Section 1, one of the most common problems when designing a multi-objective memetic algorithm is choosing a suitable local search method. There are some Pareto-based local searchers in the literature, as for example PAES (Pareto Archived Evolution Strategy) proposed in (Knowles and Corne 2000), which starts from a single initial solution and performs the selection based on dominance, keeping and returning an archive of limited size of non-dominated solutions. The Pareto Local Search proposed in (Paquete, Schiavinotto, and Stützle 2007) is also an interesting alternative.

One inconvenience of these local searchers is that they are too computationally costly to be combined with a genetic algorithm (notice that the local search will be launched many times during a single run). For this reason, we propose a less time-consuming local search procedure that provides a single (hopefully improved) output solution, what is called "one-point iteration" in (Lara et al. 2010).

Another issue when applying local search to a multi-objective setting is how to establish a selection criterion for the best neighbor. In general, there is not a single "best" neighbor, since the dominance relation $\succ$ defines a partial order. In the literature we can find different approaches to this issue. For instance, in (Ishibuchi et al. 2009) and (Jaszkiewicz 2003) the authors propose to scalarise the objective function vector to guide the search. Other authors propose instead to define acceptance criteria based on a dominance relation; for instance, in (Knowles and Corne 2000) the local search provides a set of candidate solutions by keeping an archive of non-dominated ones.

In this paper we actually need a local search as fast and efficient as possible so it can be applied to every chromosome generated by the genetic algorithm. To this end, we propose a local search based on hill climbing in which the selection of the neighbor is based on dominance but, at the same time, considers the solutions in the current non-dominated set of solutions of the genetic algorithm. Therefore, a number of neighbors may be chosen, even if they do not dominate the current solution, as long as they are actually interesting. In particular, our procedure starts with a solution provided by the genetic algorithm, and generates neighbors of the solution one by one, until it finds one neighbor that fulfills one of the following conditions:

1. The neighbor dominates the current solution.
2. The neighbor would enter in the current set of non-dominated solutions of the genetic algorithm (i.e. no solution of the population dominates the neighbor and also no solution has the exact same fitness values as the neighbor), while the current solution would not enter.

Algorithm 2 Multi-objective hill climbing local search

```
Require: A problem instance $Ins$ and a feasible schedule $S$
  $S' \leftarrow S$;
  $continue \leftarrow True$;
  while $continue = True$ do
    $NeighborSelected \leftarrow False$;
    $N(S') \leftarrow$ neighborhood of $S'$;
    $k \leftarrow 1$;
    while $NeighborSelected = False$ and $k \leq |N(S')|$ do
      $S'' \leftarrow N(S')[k]$;
      Evaluate $S''$;
      if $S''$ dominates $S'$, or $S''$ would enter in the current set of non-dominated solutions of the genetic algorithm and $S'$ would not then
        $NeighborSelected \leftarrow True$;
      end if
      $k \leftarrow k + 1$;
    end while
    if $NeighborSelected = False$ then
      $continue \leftarrow False$
    else
      $S' \leftarrow S''$
    end if
  end while
  return The (hopefully) improved solution $S'$ for $Ins$;
```

As soon as one such neighbor is found, the procedure swaps the current solution for the newly found solution and repeats the process. On the other hand, if no such neighbor exists the procedure ends, returning the current solution.

Notice that the second condition is very useful so we can select very interesting neighbors that may not dominate the current solution. However, if the current solution already would enter in the non-dominated set of solutions of the genetic algorithm, we prefer to not deviate the search and limit ourselves to dominating solutions.

The local search is detailed in Algorithm 2. We propose combining it with the genetic algorithm described in Section 3.1. As this local search is actually not very time consuming, it may be applied to all initial chromosomes and to all generated offsprings. Notice that this would not be reasonable for other alternatives such as the Pareto Local Search (Paquete, Schiavinotto, and Stützle 2007), given their greater computational load.

After the local search is applied, the chromosome is rebuilt from the improved schedule obtained by the local search, so its characteristics can be transferred to subsequent offsprings. This effect is known as Lamarckian evolution.

Neighborhood structure Several neighborhoods have been proposed in the literature for the JSP, and most of them rely on the concepts of critical path and critical block (see Section 2.1). Here we adopt the neighborhood structure initially proposed in (Van Laarhoven, Aarts, and Lenstra 1992). This structure is based on reversing critical arcs in a graph representation of a schedule S and exhibits some nice properties, in particular, it always generates feasible neighbors, avoiding the need of repairing procedures.

In TWT minimization there is an added difficulty, as the cost of a solution can be given by up to N critical paths. For each node end_i, a critical path from *start* to end_i is considered whenever its cost is greater than the due date of job J_i, d_i (González et al. 2012). The most computationally expensive part of a local search is usually the neighbor evaluation. In order to limit the computational burden of the local search we opted to limit the number of neighbors, considering only the critical path of the job J_i that contributes the most to the TWT of the overall schedule. A similar idea was proposed in (González et al. 2012). In summary, in our structure we consider the neighbors created by reversing single critical arcs in the critical path that contributes the most to the TWT.

Even if this structure is mainly designed to minimize the TWT, we have empirically seen that most neighbors that improve the TWT also improve the NPE at the same time.

3.3 The linear programming approach

The solution returned by the energy optimization procedure described in Section 3.1 could be further improved in the following way: keeping the processing ordering of the operations on the machines, delaying the starting time of the last operation of some machine may allow the first operation of another machine to be delayed as well, giving rise to a reduction in the overall energy consumption. Of course, checking for all these possibilities becomes computationally expensive and so such procedure could not be applied after each chromosome evaluation. However, it can be applied, for example, to the solutions in the Pareto set approximation returned by the memetic algorithm. With this purpose, given the problem definition of Section 2 and an input solution S, we consider the following relaxed Linear Programming (LP) problem.

$$min \quad NPE = \sum_{k=1,\dots,M} TEM_k$$

$$s.t. \quad s_v + p_v \le s_{SJ_v} \quad v \in \Omega \tag{5a}$$

$$s_v + p_v \le s_{SM_v} \\ v \in M_k - \{\omega_k\}, k = 1, \dots, M \tag{5b}$$

$$start \le s_v \quad v \text{ is the first operation of job } J_i \\ i = 1, \dots, N \tag{5c}$$

$$end_i - start \le \max\{C_i, d_i\} \quad i = 1, \dots, N \tag{5d}$$

$$start = 0 \tag{5e}$$

Decision variables are the starting times of the operations $s_v \in \Omega$ extended with the fictitious operations $start$ and end_i, $i = 1, \dots, N$. Constraints (5a) represent the linear orderings imposed on the set of operations Ω by the jobs J, note that we assume $SJ_v = end_i$ when v is the last operation of job J_i. The processing orderings on the machines in S are represented by constraints (5b). Constraints (5c) impose to the first operation of each job J_i to start after the reference operation $start$. The imposed time bounds (5d) guarantee that the final value of the TWT is less than or equal to that of the input solution S. Finally, (5e) sets the starting time of the schedule to 0. As it is easy to verify, all the imposed temporal constraints are of the kind $x - y \le c$. So in accordance with (Papadimitriou and Steiglitz 1982; Sakkout and Wallace 2000) the coefficient matrix of the above LP is *totally unimodular* (TU), and therefore all the optimal solutions of the LP problem remain discrete values and they provide the optimal NPE given the processing ordering established by the input solution S. Similar considerations are proposed in (Brandimarte and Maiocco 1999), where the optimal timing problem for non-regular single objectives in a job-shop are reduced to a minimum cost flow problem.

We propose to apply this linear programming approach in all solutions of the Pareto front obtained in the last generation of the memetic algorithm, in order to further improve its final results.

4 Experimental results

In this section we provide an empirical evaluation of the proposed algorithms, showing how they outperform the results of the state-of-the-art, in terms of improvement of both TWT and total NPE of the obtained solutions.

4.1 Test instances

Experiments were made on instances available in the literature (Liu et al. 2014). Specifically we considered one instance that was generated based on the well-known FT10 instance of the JSP, of size 10×10, adding due dates, job weights and the idle power consumption of each machine. The due dates were assigned using the following expression:

$$d_i = k \times \sum_{j=1}^{M} p_{ij}, \tag{6}$$

where M is the number of machines, that coincides with the number of operations per job. k is a parameter that controls the tightness of the due dates, being 1.5, 1.6, 1.7 and 1.8 in our work. Therefore, there are 4 instances in all. See (Liu et al. 2014) for more details on these instances.

The memetic algorithm (including the post-optimization procedure) is implemented in C++ using a single thread, while the Linear Programming step is implemented with IBM CPLEX Optimizer 12.6. Our experiments were carried out on a Intel Core i5-2450M CPU at 2.5 GHz with 4 GB of RAM, running on Windows 10 Pro.

4.2 Parameter tuning

As a result of a preliminary parametric analysis, the parameter setup for our proposal is as follows: $populationSize = 1000$, $numGenerations = 2000$, $tournamentSize = 2$, $crossoverProb = 1.0$ and $mutationProb = 0.2$.

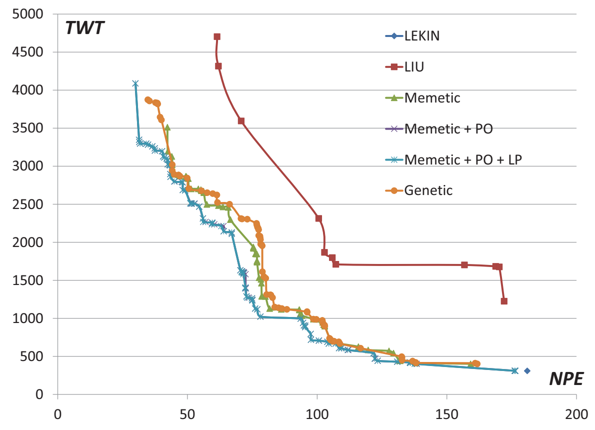


Figure 2: Pareto fronts obtained with $k = 1.5$

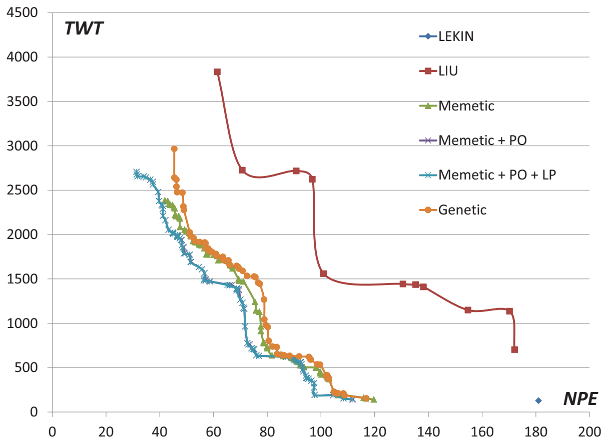


Figure 3: Pareto fronts obtained with $k = 1.6$

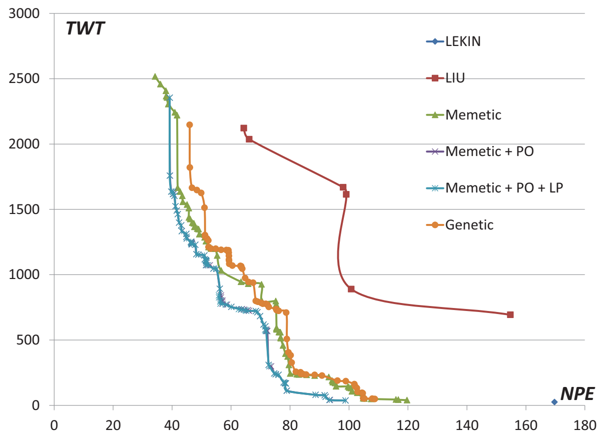


Figure 4: Pareto fronts obtained with $k = 1.7$

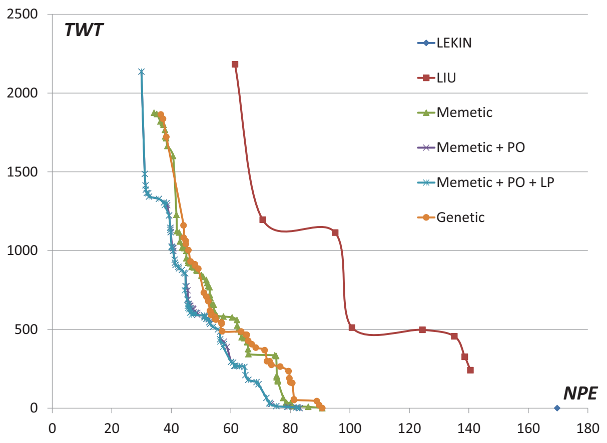


Figure 5: Pareto fronts obtained with $k = 1.8$

Using this configuration, the running time is reasonable, with an average of 4 minutes per run. The time spent in each part of the algorithm is roughly 73% for the local search, 27% for the genetic algorithm, and less than 1% for the linear programming approach. Even if the time spent by the linear programming approach is less than 1 second, it would be too computationally expensive to apply it in every generation of the algorithm.

4.3 Results and comparison with the state-of-the-art

Figures 2, 3, 4 and 5 show the Pareto fronts obtained with the presented methods, and compared with those obtained in (Liu et al. 2014), for all values of the due date tightness parameter $k = 1.5, 1.6, 1.7$ and 1.8.

The proposal of (Liu et al. 2014) (labelled LIU in all figures) is a standard NSGA-II algorithm, using OOX crossover operator and swap mutation. The crossover probability is set at 1.0 and the mutation probability at 0.6. The population size varies between 800 and 1000 depending on the instance, and the total number of generations vary between 25000 and 40000. As the authors do not report the computational time used in their runs, we have implemented a version of their method and concluded that the running time used in their experiments is considerably higher than that of our approach, about 15 minutes per run.

To have a reference value for weighted tardiness, in (Liu et al. 2014) the authors also report results using the software LEKIN. In particular, they use the Shifting Bottleneck and Local Search heuristics, both provided by that software. These metaheuristics are used to perform a single-objective optimization of the weighted tardiness, and therefore the result in this case is a single solution instead of a Pareto front. It is labelled $LEKIN$ in all figures.

The plot labelled $Memetic$ depicts the Pareto front obtained through the presented memetic algorithm (NSGA-II + multi-objective hill-climbing local search procedure), while the plot labelled $Memetic + PO$ represents the Pareto front

Table 1: Hypervolumes computed for all procedures.

	Genetic	Memetic	Memetic + PO	Memetic + PO + LP	Liu
$k = 1.5$	0.6272	0.6238	0.6654	0.6656	0.3862
$k = 1.6$	0.6801	0.6964	0.7364	0.7366	0.4553
$k = 1.7$	0.7237	0.7620	0.7635	0.7637	0.5264
$k = 1.8$	0.7842	0.7941	0.8203	0.8206	0.6038

obtained with the memetic algorithm enhanced with the low-polynomial Post-Optimization procedure. The plot labelled $Genetic$ depicts the Pareto front obtained through the genetic algorithm alone, without any local search. Lastly, the plot labelled $Memetic + PO + LP$ depicts the Pareto front obtained by further post-processing the final solutions obtained from the $Memetic + PO$ approach by means of the Linear Programming step described in Section 3.3.

We adjusted the stopping condition of our methods so that running times are similar (about 4 minutes per run), in order to achieve a comparison as fair as possible. For instance, when the energy optimization procedure is not used (plot labelled $Memetic$), we have increased $numGenerations$ to 2500, and when the local search is not used (plot labelled $Genetic$) we set $numGenerations = 8000$.

Despite all the plots demonstrate that the $Memetic$ algorithm significantly outperforms the NSGA-II multi-objective optimization algorithm used in (Liu et al. 2014), we would like to stress how the post-optimization procedure introduced in this paper ($Memetic + PO$ plots) represents a further and remarkable performance boost. In fact, it should be underscored at this point that the results obtained with the post-optimization step are indeed extremely close to those that can be obtained by applying the optimal Linear Programming approach (the two respective plots are almost completely coincident in all figures), thus proving the effectiveness of the post-optimization (PO) procedure. It can also be seen that the hybridization with local search improves the performance of the genetic algorithm, as the results improve in most cases.

Figure 6 shows two different solutions, respectively *before* and *after* the application of the post-optimization algorithm presented in Section 3.1, in the $k = 1.5$ case. By visual inspection, it is clear that the post-optimization procedure does not increase the TWT value, as the end times of the last operations on every machine are constrained to their original values in both solutions 6(a) and 6(b). On the contrary, the NPE value is significantly improved by the application of the post-optimization algorithm, whose aim is to produce a different timing on the operations by introducing delays on the start times of the initial operations of the machines (e.g., the most evident delays in Figure 6 are those related to machines R_4, R_5, and R_9), with a consequent readjustment of the idle times of every machine caused by the time compression.

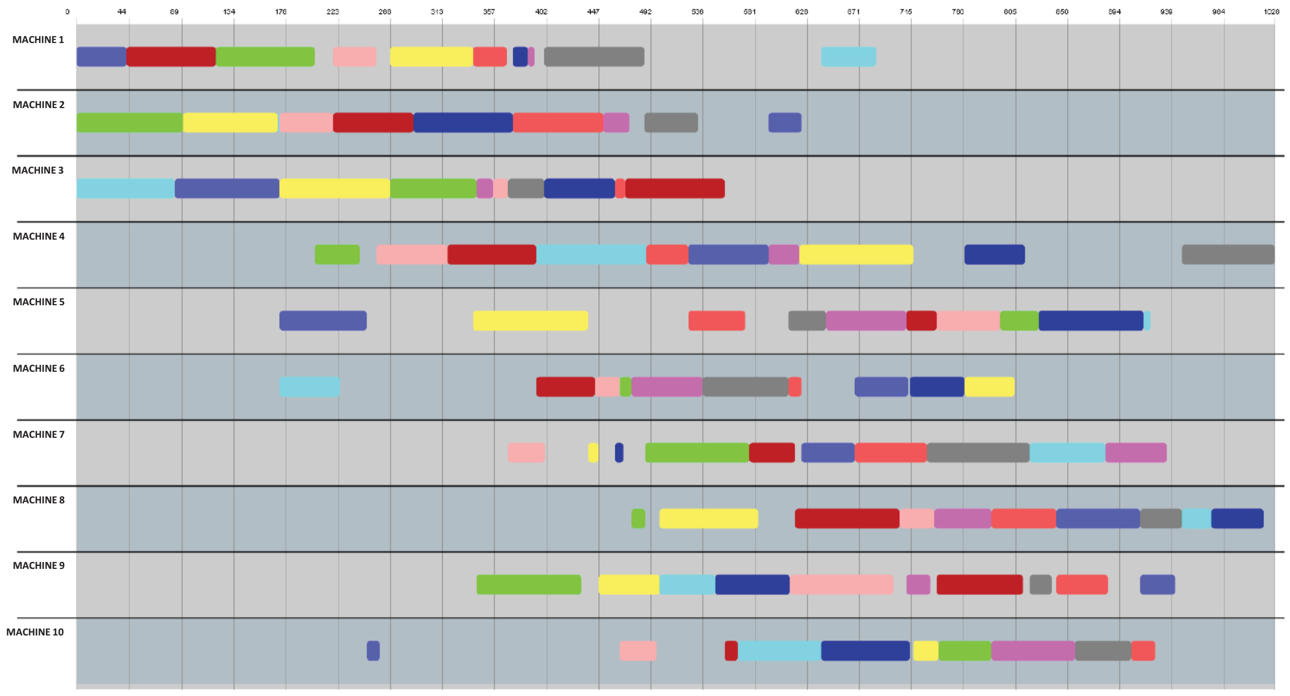
Finally, Table 1 summarizes the hypervolume values of all Pareto fronts shown in Figures 2, 3, 4 and 5. The values in the table numerically confirm the superiority of the memetic algorithm presented in this paper over the results obtained in (Liu et al. 2014), and they also confirm the effectiveness of the post-optimization procedure. It is also very remarkable how close the hypervolumes obtained with the $Memetic + PO$ approach are with those obtained with LP. However, despite the total hypervolume improvement is very small, the Linear Programming procedure was able to improve a significant number of solutions w.r.t. the $Memetic + PO$ approach (23 solutions on 76 in the $k = 1.5$ case, 31 solutions on 74 in the $k = 1.6$ case, 31 solutions on 68 in the $k = 1.7$ case, and 35 solutions on 73 with $k = 1.8$).

5 Conclusions

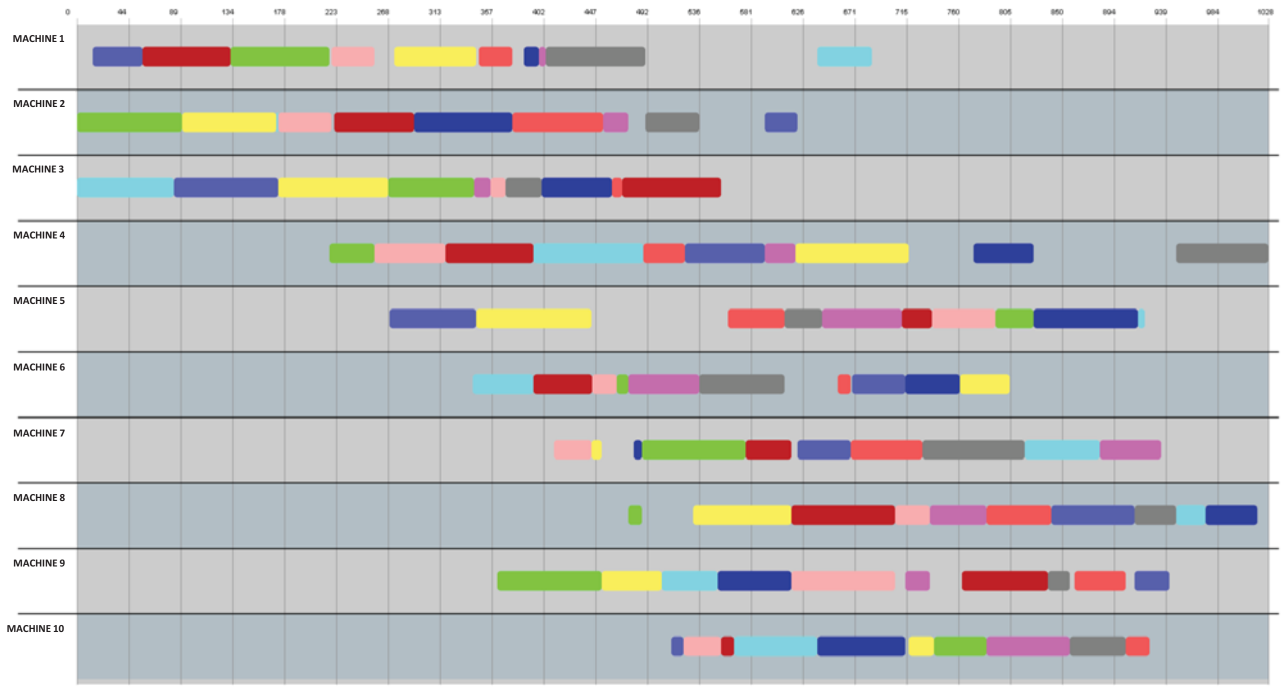
We have considered the problem of minimizing both the total weighted tardiness and the energy consumption in a job shop. To this end, we have proposed a multi-objective approach that hybridizes a NSGA-II based evolutionary algorithm with a multi-objective local search. As we have discussed, the energy consumption is a non-regular objective function, and to optimize it we have designed two methods, a fast, low-polynomial procedure to be included in the chromosome evaluation algorithm, and a linear programming approach which is more costly and applied only to the final set of non-dominated solutions, to further improve them. In the experimental study we have proven the efficiency of the proposed energy-optimization procedures and we have seen that our approach improves the results of the state-of-the-art.

In our opinion, the remarkable performance of our algorithm is due to the combination of the diversification provided by the NSGA-II combined with the intensification provided by the local search. The fast local search and the reduction of the neighborhood allowed us to apply it to every solution in a reasonable computational time. Additionally, the proposed energy optimization methods significantly improved the quality of the solutions.

For future work we will try different multi-objective algorithms, as for example MOEA-D, PAES or multi-objective scatter search, as well as constraint programming techniques. The design of neighborhoods for local search aimed at reducing the energy consumption is also a subject of further study. Additionally, we plan to design a benchmark with more instances. Another very interesting possibility is to consider more realistic energy consumption models. For example models that consider non-uniform energy costs, models that allow varying the energy consumed by varying the processing mode of operations, or the model described in (Mouzon, Yildirim, and Twomey 2007), where the machines can be Turn off/Turn on.



(a) Before post-optimization ($TWT = 3347$, $NPE = 91.98$ KWh)



(b) After post-optimization ($TWT = 3347$, $NPE = 31.35$ KWh)

Figure 6: Improvement of the NPE value as a consequence of the application of the post-optimization procedure (Algorithm 1).

Acknowledgements

We would like to thank Ying Liu for providing us with the detailed results of his work. This research has been supported by the Spanish Government under research project TIN2013-46511-C2-2-P. ISTC-CNR authors were supported by the ESA Contract No. 4000112300/14/D/MRP "Mars Express Data Planning Tool MEXAR2 Maintenance".

References

Baker, K. 1974. *Introduction to Sequencing and Scheduling*. Wiley.

Bierwirth, C. 1995. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum* 17:87–92.

Brandimarte, P., and Maiocco, M. 1999. Job shop scheduling with a non-regular objective: a comparison of neighbourhood structures based on a sequencing/timing decomposition. *International Journal of Production Research* 37(8):1697–1715.

Bülbül, K. 2011. A hybrid shifting bottleneck-tabu serach heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research* 38:967–783.

Dabia, S.; Talbi, E.-G.; van Woensel, T.; and De Kok, T. 2013. Approximating multi-objective scheduling problems. *Computers & Operations Research* 40:1165–1175.

Dai, M.; Tang, D.; Giret, A.; Salido, M. A.; and Li, W. 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing* 29:418–429.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

Essafi, I.; Mati, Y.; and Dauzère-Pérès, S. 2008. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research* 35:2599–2616.

González, M. A.; González-Rodríguez, I.; Vela, C.; and Varela, R. 2012. An efficient hybrid evolutionary algorithm for scheduling with setup times and weighted tardiness minimization. *Soft Computing* 16:2097–2113.

Ishibuchi, H.; Hitotsuyanagi, Y.; Tsukamoto, N.; and Nojima, Y. 2009. Use of biased neighborhood structures in multiobjective memetic algorithms. *Soft Computing* 13(8–9):795–810.

Jaszkiewicz, A. 2003. Do multiple-objective metaheuristics deliver on their promises? A computational experiment on the set-covering problem. *IEEE Transactions on Evolutionary Computation* 7(2):133–143.

Knowles, J. D., and Corne, D. W. 2000. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation* 8(2):149–172.

Kreipl, S. 2000. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling* 3:125–138.

Kuhpfahl, J., and Bierwirth, C. 2016. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research* 66:44–57.

Lara, A.; Sánchez, G.; Coello Coello, C. A.; and Schütze, O. 2010. HCS: A new local search strategy for memetic multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 14(1):112–132.

Liefooghe, A.; Humeau, J.; Mesmoudi, S.; Jourdan, L.; and Talbi, E.-G. 2012. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics* 18(2):317–352.

Liu, Y.; Dong, H.; Lohse, N.; Petrovic, S.; and Gindy, N. 2014. An investigation into minimising total energy consumption and total weighted tardiness in job shops. *Journal of Cleaner Production* 65:87–96.

Mati, Y.; Dauzere-Peres, S.; and Lahlou, C. 2011. A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research* 212:33–42.

Mouzon, G.; Yildirim, M. B.; and Twomey, J. 2007. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research* 45(18–19):4247–4271.

Palacios, J. J.; Vela, C. R.; González-Rodríguez, I.; and Puente, J. 2014. Schedule generation schemes for job shop problems with fuzziness. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 687–692. IOS Press.

Panwalkar, S.; Dudek, R.; and Smith, M. 1973. *Symposium on the Theory of Scheduling and Its Applications*. New York: Springer-Verlag. chapter Sequencing research and the industrial scheduling problem, 29–38.

Papadimitriou, C., and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Dover Books on Computer Science. Dover Publications.

Paquete, L.; Schiavinotto, T.; and Stützle, T. 2007. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research* 156:83–97.

Sakkout, H., and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* 5(4):359–388.

Singer, M., and Pinedo, M. 1998. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions* 30:109–118.

Singer, M., and Pinedo, M. 1999. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics* 46(1):1–17.

Van Laarhoven, P.; Aarts, E.; and Lenstra, K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40:113–125.

Job-Shop Scheduling Solver Based on Quantum Annealing

Davide Venturelli
Quantum Artificial Intelligence Laboratory (QuAIL), NASA Ames
U.S.R.A. Research Institute for Advanced Computer Science (RIACS)

Dominic J.J. Marchand
1QB Information Technologies (1QBit)

Galo Rojo
1QB Information Technologies (1QBit)

Abstract

Quantum annealing is emerging as a promising near-term quantum computing approach to solving combinatorial optimization problems. A solver for the job-shop scheduling problem that makes use of a quantum annealer is presented in detail. Inspired by methods used for constraint satisfaction problem (CSP) formulations, we first define the makespan-minimization problem as a series of decision instances before casting each instance into a time-indexed quadratic unconstrained binary optimization. Several pre-processing and graph-embedding strategies are employed to compile optimally parametrized families of problems for scheduling instances on the D-Wave Systems' Vesuvius quantum annealer (D-Wave Two). Problem simplifications and partitioning algorithms, including variable pruning, are discussed and the results from the processor are compared against classical global-optimum solvers.

I. Introduction

The commercialization and independent benchmarking (Johnson et al. (2010); Boixo et al. (2014a); Rønnow et al. (2014); McGeoch and Wang (2013)) of quantum annealers based on superconducting qubits has sparked a surge of interest for near-term practical applications of quantum analog computation in the optimization research community. Many of the early proposals for running useful problems arising in space science (Smelyanskiy et al. (2012)) have been adapted and have seen small-scale testing on the D-Wave Two processor (Rieffel et al. (2015)). The best procedure for comparison of quantum analog performance with traditional digital methods is still under debate (Rønnow et al. (2014); Hen et al. (2015); Katzgraber et al. (2015)) and remains mostly speculative due to the limited number of qubits on the currently available hardware. While waiting for the technology to scale up to more significant sizes, there is an increasing interest in the identification of small problems which are nevertheless computationally challenging and useful. One approach in this direction has been pursued in Rieffel et al. (2014), and consisted in identifying parametrized ensembles of random instances of operational planning problems of increasing sizes that can be shown to be on the verge of a solvable-unsolvable phase transition. This condition should be sufficient to observe an asymptotic exponential scaling of runtimes, even for instances of relatively small size, potentially testable on current- or next-generation D-Wave hardware. An empirical takeaway from Rieffel et al. (2015) (validated also by experimental results in O'Gorman et al. (2015b); Venturelli et al. (2015)) was that the established programming and program running techniques for quantum annealers seem to be particularly amenable to scheduling problems, allowing for an efficient mapping and good performance compared to other applied problem classes like automated navigation and Bayesian-network structure learning (O'Gorman et al. (2015a)).

Motivated by these first results, and with the intention to challenge current technologies on hard problems of practical value, we herein formulate a quantum annealing version of the job-shop scheduling problem (JSP). The JSP is essentially a general paradigmatic constraint satisfaction problem (CSP) framework for the problem of optimizing the allocation of resources required for the execution of sequences of operations with constraints on location and time. We provide compilation and running strategies for this problem using original and traditional techniques for parametrizing ensembles of instances. Results from the D-Wave Two are compared with classical exact solvers. The JSP has earned a reputation for being especially intractable, a claim supported by the fact that the best general-purpose solvers (CPLEX, Gurobi Optimizer, SCIP) struggle with instances as small as 10 machines and 10 jobs (10 x 10) (Ku and Beck (2016)). Indeed, some known 20 x 15 instances often used for benchmarking still have not been solved to optimality even by the best special-purpose solvers (Jain and Meeran (1999)), and 20 x 20 instances are typically completely intractable. We note that this early work constitutes a wide-ranging survey of possible techniques and research directions and leave a more in-depth exploration of these topics for future work.

Problem definition and conventions

Typically the JSP consists of a set of jobs $\mathcal{J} = \{\mathbf{j}_1, \dots, \mathbf{j}_N\}$ that must be scheduled on a set of machines $\mathcal{M} = \{\mathbf{m}_1, \dots, \mathbf{m}_M\}$. Each job consists of a sequence of operations that must be performed in a predefined order

$$\mathbf{j}_n = \{O_{n1} \to O_{n2} \to \cdots \to O_{nL_n}\}.$$

Job $\mathbf{j}_n$ is assumed to have L_n operations. Each operation O_{nj} has an integer execution time p_{nj} (a value of zero is allowed) and has to be executed by an assigned machine $\mathbf{m}_{q_{nj}} \in \mathcal{M}$, where q_{nj} is the index of the assigned machine. There can only be one operation running on any given machine at any given point in time and each operation of a job needs to complete before the following one can start. The usual objective is to schedule all operations in a valid sequence while minimizing the makespan (i.e., the completion time of the last running job), although other objective functions can be used. In what follows, we will denote with $\mathcal{T}$ the minimum possible makespan associated with a given JSP instance.

As defined above, the JSP variant we consider is denoted $\mathbf{J}M\big|p_{nj}\in[p_{\min},\dots,p_{\max}]\big|C_{\max}$ in the well-known $\alpha|\beta|\gamma$ notation, where $p_{\min}$ and $p_{\max}$ are the smallest and largest execution times allowed, respectively. In this notation, $\mathbf{J}M$ stands for job-shop type on M machines, and $C_{\max}$ means we are optimizing the makespan.

For notational convenience, we enumerate the operations in a lexicographical order in such a way that

$$\begin{aligned} \mathbf{j}_1 &= \{O_1 \to \dots \to O_{k_1}\}, \\ \mathbf{j}_2 &= \{O_{k_1+1} \to \dots \to O_{k_2}\}, \\ &\dots \\ \mathbf{j}_N &= \{O_{k_{N-1}+1} \to \dots \to O_{k_N}\}. \end{aligned} \tag{1}$$

Given the running index over all operations $i \in \{1,\dots,k_N\}$, we let q_i be the index of the machine $\mathbf{m}_{q_i}$ responsible for executing operation O_i. We define I_m to be the set of indices of all of the operations that have to be executed on machine $\mathbf{m}_m$, that is, $I_m = \{i : q_i = m\}$. The execution time of operation O_i is now simply denoted p_i.

A priori, a job can use the same machine more than once, or use only a fraction of the M available machines. For benchmarking purposes, it is customary to restrict a study to the problems of a specific family. In this work, we define a ratio θ that specifies the fraction of the total number of machines that is used by each job, assuming no repetition when $\theta \le 1$. For example, a ratio of 0.5 means that each job uses only $0.5M$ distinct machines.

Quantum annealing formulation

In this work, we seek a suitable formulation of the JSP for a quantum annealing optimizer (such as the D-Wave Two). The optimizer is best described as an oracle that solves an Ising problem with a given probability (Boros and Hammer (2002)). This Ising problem is equivalent to a quadratic unconstrained binary optimization (QUBO) problem (O'Gorman et al. (2015b)). The binary polynomial associated with a QUBO problem can be depicted as a graph, with nodes representing variables and values attached to nodes and edges representing linear and quadratic terms, respectively. The QUBO solver can similarly be represented as a graph where nodes represents qubits and edges represent the allowed connectivity. The optimizer is expected to find the global minimum with some probability which itself depends on the problem and the device's parameters. The device is not an ideal oracle: its limitations, with regard to precision, connectivity, and number of variables, must be considered to achieve the best possible results. As is customary, we rely on the classical procedure known as embedding to adapt the connectivity of the solver to the problem at hand. This procedure is described in a number of quantum annealing papers (Rieffel et al. (2015); Venturelli et al. (2015)). During this procedure, two or more variables can be forced to take on the same value by including additional constraints in the model. In the underlying Ising model, this is achieved by introducing a large ferromagnetic (negative) coupling J_{F} between two spins. The embedding process modifies the QUBO problem accordingly and one should not confuse the *logical* QUBO problem value, which depends on the QUBO problem and the state considered, with the Ising problem energy seen by the optimizer (which additionally depends on the extra constraints and the solver's parameters, such as J_{F}).

We distinguish between the *optimization* version of the JSP, in which we seek a valid schedule with a minimal makespan, and the *decision* version, which is limited to validating whether or not a solution exists with a makespan smaller than or equal to a user-specified timespan T. We focus exclusively on the decision version and later describe how to implement a full optimization version based on a binary search. We note that the decision formulation where jobs are constrained to fixed time windows is sometimes referred in the literature as the job-shop CSP formulation (Cheng and Smith (1997); Garrido et al. (2000)), and our study will refer to those instances where the jobs share a common deadline T.

II. QUBO problem formulation

While there are several ways the JSP can be formulated, such as the rank-based formulation (Wagner (1959)) or the disjunctive formulation (Manne (1960)), our formulation is based on a straightforward time-indexed representation particularly amenable to quantum annealers (a comparative study of mappings for planning and scheduling problems can be found in O'Gorman et al. (2015b)). We assign a set of binary variables for each operation, corresponding to the various possible discrete starting times the operation can have:

$$x_{i,t} = \begin{cases} 1 & : \text{operation } O_i \text{ starts at time } t, \\ 0 & : \text{otherwise.} \end{cases} \tag{2}$$

Here t is bounded from above by the timespan T, which represents the maximum time we allow for the jobs to complete. The timespan itself is bounded from above by the total work of the problem, that is, the sum of the execution times of all operations.

Constraints

We account for the various constraints by adding penalty terms to the QUBO problem. For example, an operation must start once and only once, leading to the constraint and associated penalty function

$$\left(\sum_t x_{i,t} = 1 \text{ for each } i\right) \to \sum_i \left(\sum_t x_{i,t} - 1\right)^2 . \tag{3}$$

a)

	operation 1	operation 2	operation 3
j_1	$\mathbf{m}_1$, $p = 2$	$\mathbf{m}_2$, $p = 1$	$\mathbf{m}_3$, $p = 1$
j_2	$\mathbf{m}_3$, $p = 2$	$\mathbf{m}_1$, $p = 1$	$\mathbf{m}_2$, $p = 2$
j_3	$\mathbf{m}_2$, $p = 1$	$\mathbf{m}_1$, $p = 1$	$\mathbf{m}_3$, $p = 2$

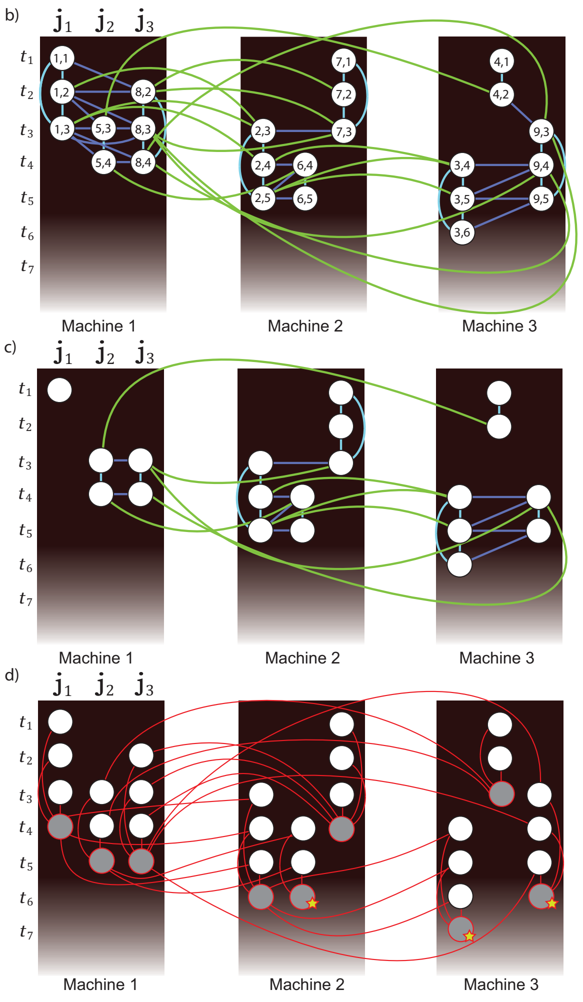


Figure 1: a) Table representation of an example 3 x 3 instance whose execution times have been randomly selected to be either 1 or 2 time units. b) Pictorial view of the QUBO mapping of the above example for $H_{T=6}$. Green, purple, and cyan edges refer respectively to h_1, h_2, and h_3 quadratic coupling terms (Eqs. 7–9). Each circle represents a bit with its i, t index as in Eq. 2. c) The same QUBO problem as in (b) after the variable pruning procedure detailed in the section on QUBO formulation refinements. Isolated qubits are bits with fixed assignments that can be eliminated from the final QUBO problem. d) The same QUBO problem as in (b) for $H_{T=7}$. Previously displayed edges in the above figure are omitted. Red edges/circles represent the variations with respect to $H_{T=6}$. Yellow stars indicate the bits which are penalized with local fields for timespan discrimination.

There can only be one job running on each machine at any given point in time, which expressed as quadratic constraints yields

$$\sum_{(i,t,k,t')\in R_m} x_{i,t}x_{k,t'} = 0 \text{ for each } m, \tag{4}$$

where $R_m = A_m \cup B_m$ and

$$\begin{aligned} A_m &= \{(i,t,k,t') : (i,k) \in I_m \times I_m, \\ &\quad i \neq k, 0 \leq t, t' \leq T, 0 < t' - t < p_i\}, \\ B_m &= \{(i,t,k,t') : (i,k) \in I_m \times I_m, \\ &\quad i < k, t' = t, p_i > 0, p_j > 0\}. \end{aligned}$$

The set A_m is defined so that the constraint forbids operation O_j from starting at t' if there is another operation O_i still running, which happens if O_i started at time t and $t' - t$ is less than p_i. The set B_m is defined so that two jobs cannot start at the same time, unless at least one of them has an execution time equal to zero. Finally, the order of the operations within a job are enforced with

$$\sum_{\substack{k_{n-1}<i<k_n \\ t+p_i>t'}} x_{i,t}x_{i+1,t'} \quad \text{for each } n, \tag{5}$$

which counts the number of precedence violations between consecutive operations only.

The resulting classical objective function (Hamiltonian) is given by

$$H_T(\bar{x}) = \eta h_1(\bar{x}) + \alpha h_2(\bar{x}) + \beta h_3(\bar{x}), \tag{6}$$

where

$$h_1(\bar{x}) = \sum_n \left(\sum_{\substack{k_{n-1}<i<k_n \\ t+p_i>t'}} x_{i,t}x_{i+1,t'} \right), \tag{7}$$

$$h_2(\bar{x}) = \sum_m \left(\sum_{(i,t,k,t')\in R_m} x_{i,t}x_{k,t'} \right), \tag{8}$$

$$h_3(\bar{x}) = \sum_i \left(\sum_t x_{i,t} - 1 \right)^2, \tag{9}$$

and the penalty constants η, α, and β are required to be larger than 0 to ensure that unfeasible solutions do not have a lower energy than the ground state(s). As expected for a decision problem, we note that the minimum of H_T is 0 and it is only reached if a schedule satisfies all of the constraints. The index of H_T explicitly shows the dependence of the Hamiltonian on the timespan T, which affects the number of variables involved. Figure 1-b illustrates the QUBO problem mapping for $H_{T=6}$ for a particular 3 x 3 example (Figure 1-a).

Simple variable pruning

Figure 1-b also reveals that a significant number of the NMT binary variables required for the mapping can be pruned by applying simple restrictions on the time index t

(whose computation is polynomial as the system size increases and therefore trivial here). Namely, we can define an effective release time for each operation corresponding to the sum of the execution times of the preceding operations in the same job. A similar upper bound corresponding to the timespan minus all of the execution times of the subsequent operations of the same job can be set. The bits corresponding to these invalid starting times can be eliminated from the QUBO problem altogether since any valid solution would require them to be strictly zero. This simplification eliminates an estimated number of variables equal to $NM\left(M\left\langle p\right\rangle - 1\right)$, where $\langle p\rangle$ represents the average execution time of the operations. This result can be generalized to consider the previously defined ratio θ, such that the total number of variables required after this simple QUBO problem pre-processing is $\theta NM[T - \theta M\langle p\rangle + 1]$.

III. QUBO formulation refinements

Although the above formulation proves sufficient for running JSPs on the D-Wave machine, we explore a few potential refinements. The first pushes the limit of simple variable pruning by considering more advanced criteria for reducing the possible execution window of each task. The second refinement proposes a compromise between the decision version of the JSP and a full optimization version.

Window shaving

In the time-index formalism, reducing the execution windows of operations (i.e., shaving) (Martin and Shmoys (1996)), or in the disjunctive approach, adjusting the *heads* and *tails* of operations (Carlier and Pinson (1994); Péridy and Rivreau (2005)), or more generally, by applying constraints propagation techniques (e.g., Caseau and Laburthe (1994)), together constitute the basis for a number of classical approaches to solving the JSP. Shaving is sometimes used as a pre-processing step or as a way to obtain a lower bound on the makespan before applying other methods. The interest from our perspective is to showcase how such classical techniques remain relevant, without straying from our quantum annealing approach, when applied to the problem of pruning as many variables as possible. This enables larger problems to be considered and improves the success rate of embeddability in general (see Figure 3), without significantly affecting the order of magnitude of the overall time to solution in the asymptotic regime. Further immediate advantages of reducing the required number of qubits become apparent during the compilation of JSP instances for the D-Wave device due to the associated embedding overhead that depends directly on the number of variables. The shaving process is typically handled by a classical algorithm whose worst-case complexity remains polynomial. While this does not negatively impact the fundamental complexity of solving JSP instances, for *pragmatic* benchmarking the execution time needs to be taken into account and added to the quantum annealing runtime to properly report the time to solution of the whole algorithm.

Different elimination rules can be applied. We focus herein on the iterated Carlier and Pinson (ICP) procedure (Carlier and Pinson (1994)) reviewed in the supplemental material with worst-case complexity given by $\mathcal{O}(N^2M^2T\log(N))$. Instead of looking at the one-job subproblems and their constraints to eliminate variables, as we did for the simple pruning, we look at the one-machine subproblems and their associated constraints to further prune variables. An example of the resulting QUBO problem is presented in Figure 1-c.

Timespan discrimination

We explore a method of extracting more information regarding the actual optimal makespan of a problem within a single call to the solver by breaking the degeneracy of the ground states and spreading them over some finite energy scale, distinguishing the energy of valid schedules on the basis of their makespan. Taken to the extreme, this approach would amount to solving the full optimization problem. We find that the resulting QUBO problem is poorly suited to a solver with limited precision, so a balance must be struck between extra information and the precision requirement. A systematic study of how best to balance the amount of information obtained versus the extra cost will be the subject of future work.

We propose to add a number of linear terms, or local fields, to the QUBO problem to slightly penalize valid solutions with larger makespans. We do this by adding a cost to the last operation of each job, that is, the set $\{O_{k_1}, \ldots, O_{k_N}\}$. At the same time, we require that the new range of energy over which the feasible solutions are spread stays within the minimum logical QUBO problem's gap given by $\Delta E = \min\{\eta, \alpha, \beta\}$. If the solver's precision can accomodate K distinguishable energy bins, then makespans within $[T - K, T]$ can be immediately identified from their energy values. The procedure is illustrated in Figure 1-d and some implications are discussed in the supplemental material appended to a longer version of this work (Venturelli, Marchand, and Rojo (2015)).

IV. Ensemble pre-characterization and compilation

We now turn to a few important elements of our computational strategy for solving JSP instances. We first show how a careful pre-characterization of classes of random JSP instances, representative of the problems to be run on the quantum optimizer, provides very useful information regarding the shape of the search space for $\mathcal{T}$. We then describe how instances are compiled to run on the actual hardware.

Makespan Estimation

In Figure 2, we show the distributions of the optimal makespans $\mathcal{T}$ for different ensembles of instances parametrized by their size $N = M$, by the possible values of task durations $\mathcal{P}_p = \{p_{\min}, \ldots, p_{\max}\}$, and by the ratio $\theta \leq 1$ of the number of machines used by each job. Instances are generated randomly by selecting θM distinct machines for each job and assigning an execution time to each operation randomly. For each set of parameters, we can compute solutions with a classical exhaustive solver in order to identify the median of the distribution $\langle\mathcal{T}\rangle(N, \mathcal{P}_p, \theta)$ as well as the other quantiles. These could also be inferred from previously solved instances with the proposed annealing solver. The resulting information can be used to guide the binary search required to solve the optimization problem. Figure 2 indicates that a normal distribution is an adequate approximation, so we need only to estimate its average $\langle\mathcal{T}\rangle$ and variance σ^2. Interestingly, from the characterization of the families of instances up to $N = 10$ we find that, at least in the region explored, the average minimum makespan $\langle\mathcal{T}\rangle$ is proportional to the average execution time of a job $\langle p\rangle\theta N$, where $\langle p\rangle$ is the mean of $\mathcal{P}_p$. This linear ansatz allows for the extrapolation of approximate resource requirements for classes of problems which have not yet been pre-characterized, and it constitutes an educated guess for classes of problems which cannot be precharacterized due to their difficulty or size. The accuracy of these functional forms was verified by computing the relative error of the prediction versus the fit of the makespan distribution of each parametrized family up to $N = M = 9$ and $p_{\max} = 20$ using 200 instances to compute the makespan histogram. The prediction for $\langle\mathcal{T}\rangle$ results are consistently at least 95% accurate, while the one for σ has at worst a 30% error margin, a very approximate but sufficient model for the current purpose of guiding the binary search.

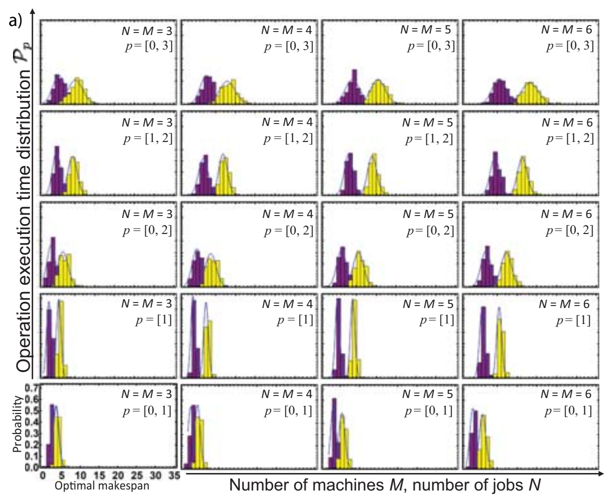


Figure 2: a) Normalized histograms of optimal makespans $\mathcal{T}$ for parametrized families of JSP instances with $N = M$, $\mathcal{P}_p$ on the y-axis, $\theta = 1$ (yellow), and $\theta = 0.5$ (purple). The distributions are histograms of occurrences for 1000 random instances, fitted with a Gaussian function of mean $\langle\mathcal{T}\rangle$. We note that the width of the distributions increases as the range of the execution times $\mathcal{P}_p$ increases, for fixed $\langle p\rangle$. The mean and the variance are well fitted respectively by $\langle\mathcal{T}\rangle = A_\mathcal{T} N p_{\min} + B_\mathcal{T} N p_{\max}$ and $\sigma = \sigma_0 + C_\sigma\langle\mathcal{T}\rangle + A_\sigma p_{\min} + B_\sigma p_{\max}$, with $A_\mathcal{T} = 0.67$, $B_\mathcal{T} = 0.82$, $\sigma_0 = 0.7$, $A_\sigma = -0.03$, $B_\sigma = 0.43$, and $C_\sigma = 0.003$.

Compilation

The graph-minor embedding technique (abbreviated simply "embedding") represents the de facto method of recasting the Ising problems to a form compatible with the layout of the annealer's architecture (Kaminsky and Lloyd (2004); Choi (2011)), which for the D-Wave Two is a Chimera graph (Johnson et al. (2010)). Formally, we seek an isomorphism between the problem's QUBO graph and a graph minor of the solver. This procedure has become a standard in solving applied problems using quantum annealing (Rieffel et al. (2015); Venturelli et al. (2015)) and can be thought of as the analogue of compilation in a digital computer programming framework during which variables are assigned to hardware registers and memory locations. A more detailed version of this work with supplemental material covering this process is available in (Venturelli, Marchand, and Rojo (2015)). An example of embedding for a 5 x 5 JSP instance with $\theta = 1$ and $T = 7$ is shown in Figure 3-a, where the 72 logical variables of the QUBO problem are embedded using 257 qubits of the Chimera graph. Finding the optimal tiling that uses the fewest qubits is NP-hard (Adler et al. (2010)), and the standard approach is to employ heuristic algorithms (Cai, Macready, and Roy (2014)). In general, for the embedding of time-indexed mixed-integer programming QUBO problems of size N into a graph of degree k, one should expect a quadratic overhead in the number of binary variables on the order of aN^2, with $a \leq (k-2)^{-1}$ depending on the embedding algorithm and the hardware connectivity (Venturelli et al. (2015)). This quadratic scaling is apparent in Figure 3-b where we report on the compilation attempts using the algorithm in Cai, Macready, and Roy (2014). Results are presented for the D-Wave chip installed at NASA Ames at the time of this study, for a larger chip with the same size of Chimera block and connectivity pattern (like the latest chip currently being manufactured by D-Wave Systems), and for a speculative yet-larger chip where the Chimera block is twice as large. We deem a JSP instance embeddable when the respective $H_{T=\mathcal{T}}$ is embeddable, so the decrease in probability of embedding with increasing system size is closely related to the shift and spreading of the optimal makespan distributions for ensembles of increasing size (see Figure 2). What we observe is that, with the available algorithms, the current architecture admits embedded JSP instances whose total execution time $NM\theta\langle p\rangle$ is around 20 time units, while near-future (we estimate 2 years) D-Wave chip architectures could potentially double that. As noted in similar studies (e.g., Rieffel et al. (2015)), graph connectivity has a much more dramatic impact on embeddability than qubit count.

Once the topological aspect of embedding has been solved, we set the ferromagnetic interactions needed to adapt the connectivity of the solver to the problem at hand. For the purpose of this work, this should be regarded as a technicality necessary to tune the performance of the experimental analog device and we include the results for completeness. Introductory details about the procedure can be found in (Rieffel et al. (2015); Venturelli et al. (2015)). In Figure 3-c we show a characterization of the ensemble of JSP instances (parametrized by N, M, θ, and $\mathcal{P}_p$, as described at the be-

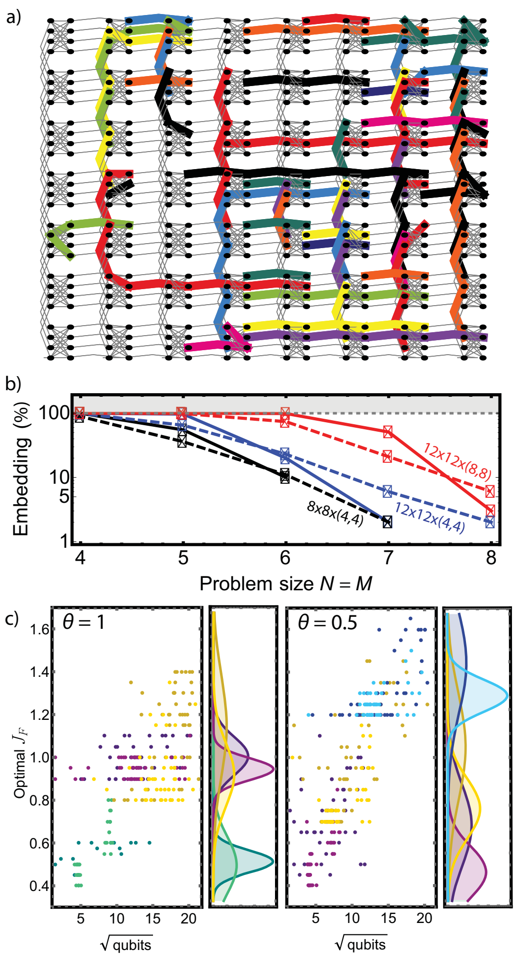


Figure 3: a) Example of an embedded JSP instance on NASA's D-Wave Two chip. Each chain of qubits is colored to represent a logical binary variable determined by the embedding. For clarity, active connections between the qubits are not shown. b) Embedding probability as a function of $N = M$ for $\theta = 1$ (similar results are observed for $\theta = 0.5$). Solid lines refer to $\mathcal{P}_p = [1, 1]$ and dashed lines refer to $\mathcal{P}_p = [0, 2]$. 1000 random instances have been generated for each point, and a cutoff of 2 minutes has been set for the heuristic algorithm to find a valid topological embedding. Results for different sizes of Chimera are shown. c) Optimal parameter-setting analysis for the ensembles of JSP instances we studied. Each point corresponds to the number of qubits and the optimal J_F (see main text) of a random instance, and each color represents a parametrized ensemble (green: 3 x 3, purple: 4 x 4, yellow: 5 x 5, blue: 6 x 6; darker colors represent ensembles with $\mathcal{P}_p = [1, 1]$ as opposed to lighter colors which indicate $\mathcal{P}_p = [0, 2]$). Distributions on the right of scatter plots represent Gaussian fits of the histogram of the optimal J_F for each ensemble. Runtime results are averaged over an ungauged run and 4 additional runs with random gauges (Perdomo-Ortiz et al. (2015a)).

ginning of this section). We present the best ferromagnetic couplings found by runs on the D-Wave machine under the simplification of a uniform ferromagnetic coupling by solving the embedded problems with values of J_F from 0.4 to 1.8 in relative energy units of the largest coupling of the original Ising problem. The run parameters used to determine the best J_F are the same as we report in the following sections, and the problem sets tested correspond to Hamiltonians whose timespan is equal to the sought makespan $H_{T=\mathcal{T}}$. This parameter-setting approach is similar to the one followed in Rieffel et al. (2015) for operational planning problems, where the instance ensembles were classified by problem size before compilation. What emerges from this preliminary analysis is that each parametrized ensemble can be associated to a distribution of optimal J_F that can be quite wide, especially for the ensembles with $p_{\min} = 0$ and large $p_{\max}$. This spread might discourage the use of the mean value of such a distribution as a predictor of the best J_F to use for the embedding of new untested instances. However, the results from this predictor appear to be better than the more intuitive prediction obtained by correlating the number of qubits after compilation with the optimal J_F. This means that for the D-Wave machine to achieve optimal performance on structured problems, it seems to be beneficial to use the information contained in the structure of the logical problem to determine the best parameters. We note that this "offline" parameter-setting could be used in combination with "online" performance estimation methods such as the ones described in Perdomo-Ortiz et al. (2015a) in order to reach the best possible instance-specific J_F with a series of quick experimental runs. The application of these techniques, together with the testing of alternative offline predictors, will be the subject of future work.

V. Results of test runs and discussion

A complete quantum annealing JSP solver designed to solve an instance to optimality using our proposed formulation will require the independent solution of several embedded instances $\{H_T\}$, each corresponding to a different timespan T. Assuming that the embedding time, the machine setup time, and the latency between subsequent operations can all be neglected, due to their being non-fundamental, the running time $\mathtt{T}$ of the approach for a specific JSP instance reduces to the expected *total* annealing time necessary to find the optimal solution of each embedded instance with a specified minimum target probability $\simeq 1$. The probability of ending the annealing cycle in a desired ground state depends, in an essentially unknown way, on the embedded Ising Hamiltonian spectrum, the relaxation properties of the environment, the effect of noise, and the annealing profile. Understanding through an ab initio approach what is the best computational strategy appears to be a formidable undertaking that would require theoretical breakthroughs in the understanding of open-system quantum annealing (Boixo et al. (2014b); Smelyanskiy et al. (2015)), as well as a tailored algorithmic analysis that could take advantage of the problem structure that the annealer needs to solve. For the time being, and for the purposes of this work, it seems much more practical to limit these early investigations to the most relevant instances, and to lay out empirical procedures that work under some general assumptions. More specifically, we focus on solving the CSP version of JSP, not the full optimization problem, and we therefore only benchmark the Hamiltonians with $\mathcal{T} = T$ with the D-Wave machine. We note however that a full optimization solver can be realized by leveraging data analysis of past results on parametrized ensembles and by implementing an adaptive binary search. Full details can be found in a longer version of this work (Venturelli, Marchand, and Rojo (2015)).

On the quantum annealer installed at NASA Ames (it has 509 working qubits; details are presented in Perdomo-Ortiz et al. (2015b)), we run hundreds of instances, sampling the ensembles $N = M \in \{3, 4, 5, 6\}$, $\theta \in \{0.5, 1\}$, and $P_p \in \{[1, 1], [0, 2]\}$. For each instance, we report results, such as runtimes, at the most optimal J_F among those tested, assuming the application of an optimized parameter-setting procedure along the lines of that described in the previous section. Figure 4-a displays the total annealing repetitions required to achieve a 99% probability of success on the ground state of $H_{\mathcal{T}}$, with each repetition lasting $t_A = 20$ μs, as a function of the number of qubits in the embedded (and pruned) Hamiltonian. We observe an exponential increase in complexity with increasing Hamiltonian size, for both classes of problems studied. This likely means that while the problems tested are small, the analog optimization procedure intrinsic to the D-Wave device's operation is already subject to the fundamental complexity bottlenecks of the JSP. It is, however, premature to draw conclusions about performance scaling of the technology given the current constraints on calibration procedures, annealing time, etc. Many of these problems are expected to be either overcome or nearly so with the next generation of D-Wave chip, at which point more extensive experimentation will be warranted.

In Figure 4-b, we compare the performance of the D-Wave device to two exhaustive classical algorithms in order to gain insight on how current quantum annealing technology compares with paradigmatic classical optimization methods. Leaving the performance of approximate solutions for future work, we chose not to explore the plethora of possible heuristic methods as we operate the D-Wave machine, seeking the global optimum.

The first algorithm, B, detailed in Brucker, Jurisch, and Sievers (1994), exploits the disjunctive graph representation and a branch-and-bound strategy that very effectively combines a branching scheme based on selecting the direction of a single disjunctive edge (according to some single-machine constraints), and a technique introduced in Carlier and Pinson (1991) for fixing additional disjunctions (based on a preemptive relaxation). It has publicly available code and is considered a well-performing complete solver for the small instances currently accessible to us, while remaining competitive for larger ones even if other classical approaches become more favorable (Beck, Feng, and Watson (2011)). B has been used in Streeter and Smith (2006) to discuss the possibility of a phase transition in the JSP, demonstrating that the random instances with $N = M$ are particularly hard families of problems, not unlike what is observed for the

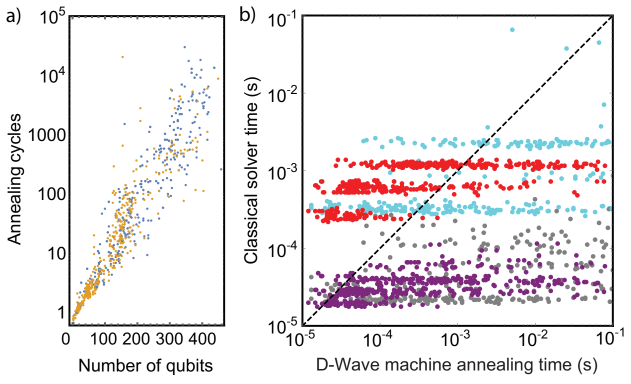


Figure 4: a) Number of repetitions required to solve $H_{\mathcal{T}}$ with the D-Wave Two with a 99% probability of success (see the supplemental material (Venturelli, Marchand, and Rojo (2015))). The blue points indicate instances with $\theta = 1$ and yellow points correspond to $\theta = 0.5$ (they are the same instances and runtimes used for Figure 3-c). The number of qubits on the x-axis represents the qubits used after embedding. b) Correlation plot between classical solvers and the D-Wave optimizer. Gray and violet points represent runtimes compared with algorithm B, and cyan and red are compared to the MS algorithm, respectively, with $\theta = 1$ and $\theta = 0.5$. All results presented correspond to the best out of 5 gauges selected randomly for every instance. In case the machine returns embedding components whose values are discordant, we apply a majority voting rule to recover a solution within the logical subspace (Venturelli et al. (2015); Rieffel et al. (2015); Perdomo-Ortiz et al. (2015a); King and McGeoch (2014); Pudenz, Albash, and Lidar (2014)). We observe a deviation of about an order of magnitude on the annealing time if we average over 5 gauges instead of picking the best one, indicating that there is considerable room for improvement if we were to apply more-advanced calibration techniques (Perdomo-Ortiz et al. (2015b)).

quantum annealing implementation of planning problems based on graph vertex coloring (Rieffel et al. (2014)).

The second algorithm, MS, introduced in Martin and Shmoys (1996), proposes a time-based branching scheme where a decision is made at each node to either schedule or delay one of the available operations at the current time. The authors then rely on a series of shaving procedures such as those proposed by Carlier and Pinson (1994) to determine the new bound and whether the choice leads to valid schedules. This algorithm is a natural comparison with the present quantum annealing approach as it solves the decision version of the JSP in a very similar fashion to the time-indexed formulation we have implemented on the D-Wave machine, and it makes use of the same shaving technique that we adapted as a pre-processing step for variable pruning. However, we should mention that the variable pruning that we implemented to simplify $H_{\mathcal{T}}$ is employed at each node of the classical branch and bound algorithm, so the overall computational time of MS is usually much more important than our one-pass pre-processing step, and in general its runtime does not scale polynomially with the problem size.

What is apparent from the correlation plot in Figure 4-b is that the D-Wave machine is easily outperformed by a classical algorithm run on a modern single-core processor, and that the problem sizes tested in this study are still too small for the asymptotic behavior of the classical algorithms to be clearly demonstrated and measured. The comparison between the D-Wave machine's solution time for $H_{\mathcal{T}}$ and the full optimization provided by B is confronting two very different algorithms, and shows that B solves all of the full optimization problems that have been tested within milliseconds, whereas D-Wave's machine can sometimes take tenths of a second (before applying the multiplier factor $\simeq 2$, due to the binary search; see the supplemental material (Venturelli, Marchand, and Rojo (2015))). When larger chips become available, however, it will be interesting to compare B to a quantum annealing solver for sizes considered B-intractable due to increasing memory and time requirements.

The comparison with the MS method has a promising signature even now, with roughly half of the instances being solved by D-Wave's hardware faster than the MS algorithm (with the caveat that our straightforward implementation is not fully optimized). Interestingly, the different parametrized ensembles of problems have distinctively different computational complexity characterized by well-recognizable average computational time to solution for MS (i.e., the points are "stacked around horizontal lines" in Figure 4-b), whereas the D-Wave machine's complexity seems to be sensitive mostly to the total qubit count (see Figure 4-a) irrespective of the problem class. We emphasize again that conclusions on speedup and asymptotic advantage still cannot be confirmed until improved hardware with more qubits and less noise becomes available for empirical testing.

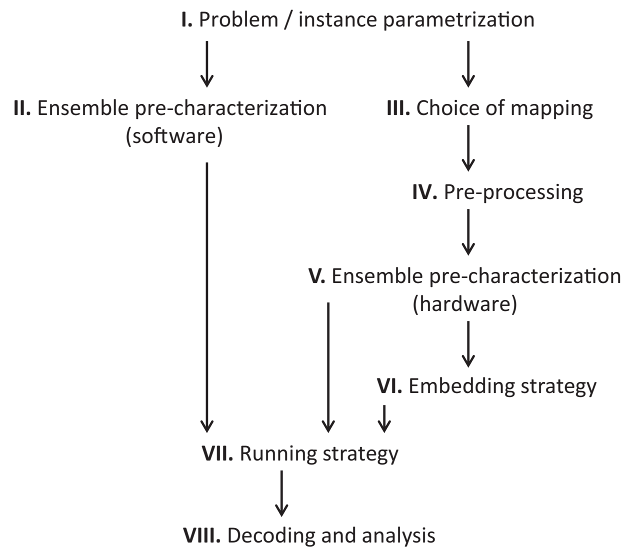

I. Problem / instance parametrization
II. Ensemble pre-characterization (software)
III. Choice of mapping
IV. Pre-processing
V. Ensemble pre-characterization (hardware)
VI. Embedding strategy
VII. Running strategy
VIII. Decoding and analysis

Figure 5: I–II) Appropriate choice of benchmarking and classical simulations is discussed in Section IV. III–IV) Mapping to QUBO problems is discussed in Sections II and III. V–VI) Pre-characterization for parameter setting is described in Section VI. VII) Structured run strategies adapted to specific problems have not to our knowledge been discussed before. We discuss a prescription in the supplementary material in (Venturelli, Marchand, and Rojo (2015)). VIII) The only decoding required in our work is majority voting within embedding components to recover error-free logical solutions. The time-indexed formulation then provides QUBO problem solutions that can straightforwardly be represented as Gantt charts of the schedules.

VI. Conclusions

Although it is probable that the quantum annealing-based JSP solver proposed herein will not prove competitive until the arrival of an annealer a few generations away, the implementation of a provably tough application from top to bottom was missing in the literature, and our work has led to noteworthy outcomes we expect will pave the way for more advanced applications of quantum annealing. Whereas part of the attraction of quantum annealing is the possibility of applying the method irrespective of the structure of the QUBO problem, we have shown how to design a quantum annealing solver, mindful of many of the peculiarities of the annealing hardware and the problem at hand, for improved performance. Figure 5 shows a schematic view of the streamlined solving process describing a full JSP optimization solver. The pictured scheme is not intended to be complete, for example, the solving framework can benefit from other concepts such as performance tuning techniques (Perdomo-Ortiz et al. (2015a)) and error-correction repetition lattices (Vinci et al. (2015)). The use of the decision version of the problem can be combined with a properly designed search strategy (the simplest being a binary search) in order to be able to seek the minimum value of the common deadline of feasible schedules. The proposed timespan discrimination further provides an adjustable compromise between the full optimization and decision formulations of the problems, allowing for instant benefits from future improvements in precision without the need for a new formulation or additional binary variables to implement the makespan minimization as a term in the objective function. As will be explored further in future work, we found that instance precharacterization performed to fine tune the solver parameters can also be used to improve the search strategy, and that it constitutes a tool whose use we expect to become common practice in problems amenable to CSP formulations as the ones proposed for the JSP. Additionally, we have shown that there is great potential in adapting classical algorithms with favorable polynomial scaling as pre-processing techniques to either prune variables or reduce the search space. Hybrid approaches and metaheuristics are already fruitful areas of research and ones that are likely to see promising developments with the advent of these new quantum heuristics algorithms.

Acknowledgements

The authors would like to thank J. Frank, M. Do, E.G. Rieffel, B. O'Gorman, M. Bucyk, P. Haghnegahdar, and other researchers at QuAIL and 1QBit for useful input and discussions. This research was supported by 1QBit, Mitacs, NASA (Sponsor Award Number NNX12AK33A), and by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IAA 145483 and by the AFRL Information Directorate under grant F4HBKC4162G001.

Note: Supplemental introductory material can be found in the longer version of this work (Venturelli, Marchand, and Rojo (2015)).

References

Adler, I.; Dorn, F.; Fomin, F. V.; Sau, I.; and Thilikos, D. M. 2010. Faster parameterized algorithms for minor containment. In Kaplan, H., ed., *Algorithm Theory - SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 322–333.

Beck, J. C.; Feng, T. K.; and Watson, J.-P. 2011. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing* 23(1):1–14.

Boixo, S.; Ronnow, T. F.; Isakov, S. V.; Wang, Z.; Wecker, D.; Lidar, D. A.; Martinis, J. M.; and Troyer, M. 2014a. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics* 10(3):218–224.

Boixo, S.; Smelyanskiy, V. N.; Shabani, A.; Isakov, S. V.; Dykman, M.; Denchev, V. S.; Amin, M.; Smirnov, A.; Mohseni, M.; and Neven, H. 2014b. Computational role of collective tunneling in a quantum annealer. *arXiv:1411.4036 [quant-ph]*.

Boros, E., and Hammer, P. L. 2002. Pseudo-boolean optimization. *Discrete Applied Mathematics* 123(1-3):155–225.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49:107 – 127.

Cai, J.; Macready, W. G.; and Roy, A. 2014. A practical heuristic for finding graph minors. *arXiv:1406.2741 [quant-ph]*.

Carlier, J., and Pinson, E. 1991. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research* 26(1-4):269–287.

Carlier, J., and Pinson, E. 1994. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research* 78(2):146 – 161.

Caseau, Y., and Laburthe, F. 1994. Improved clp scheduling with tasks intervals. *Proc. of the 11th International Conference on Logic Programming*.

Cheng, C.-C., and Smith, S. F. 1997. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research* 70:327–357.

Choi, V. 2011. Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. *Quantum Information Processing* 10(3):343–353.

Garrido, A.; Salido, M. A.; Barber, F.; and López, M. 2000. Heuristic methods for solving job-shop scheduling problems. In *Proc. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000)*, 44–49.

Hen, I.; Job, J.; Albash, T.; Rønnow, T. F.; Troyer, M.; and Lidar, D. 2015. Probing for quantum speedup in spin glass problems with planted solutions. *arXiv:1502.01663 [quant-ph]*.

Jain, A., and Meeran, S. 1999. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research* 113(2):390 – 434.

Johnson, M. W.; Bunyk, P.; Maibaum, F.; Tolkacheva, E.; Berkley, A. J.; Chapple, E. M.; Harris, R.; Johansson, J.; Lanting, T.; Perminov, I.; Ladizinsky, E.; Oh, T.; and Rose, G. 2010. A scalable control system for a superconducting adiabatic quantum optimization processor. *Superconductor Science and Technology* 23(6):065004.

Kaminsky, W. M., and Lloyd, S. 2004. Scalable architecture for adiabatic quantum computing of np-hard problems. In Leggett, A. J.; Ruggiero, B.; and Silvestrini, P., eds., *Quantum Computing and Quantum Bits in Mesoscopic Systems*. Springer US. 229–236.

Katzgraber, H. G.; Hamze, F.; Zhu, Z.; Ochoa, A. J.; and Munoz-Bauza, H. 2015. Seeking quantum speedup through spin glasses: The good, the bad, and the ugly*. *Physical Review X* 5:031026.

King, A. D., and McGeoch, C. C. 2014. Algorithm engineering for a quantum annealing platform. *arXiv:1410.2628 [cs.DS]*.

Ku, W.-Y., and Beck, J. C. 2016. Revisiting off-the-shelf mixed integer programming and constraint programming models for job shop scheduling. *Computers & Operations Research*.

Manne, A. S. 1960. On the job-shop scheduling problem. *Operations Research* 8(2):219–223.

Martin, P., and Shmoys, D. B. 1996. A new approach to computing optimal schedules for the job-shop scheduling problem. In Cunningham, W. H.; McCormick, S.; and Queyranne, M., eds., *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 389–403.

McGeoch, C. C., and Wang, C. 2013. Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, 23:1–23:11. New York, NY, USA: ACM.

O'Gorman, B.; Babbush, R.; Perdomo-Ortiz, A.; Aspuru-Guzik, A.; and Smelyanskiy, V. 2015a. Bayesian network structure learning using quantum annealing. *The European Physical Journal Special Topics* 224(1):163–188.

O'Gorman, B.; Rieffel, E. G.; Minh, D.; and Venturelli, D. 2015b. Compiling planning into quantum optimization problems: a comparative study. In *ICAPS 2015, Research Workshop Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

Perdomo-Ortiz, A.; Fluegemann, J.; Biswas, R.; and Smelyanskiy, V. N. 2015a. A Performance Estimator for Quantum Annealers: Gauge selection and Parameter Setting. *arXiv:1503.01083 [quant-ph].*

Perdomo-Ortiz, A.; O'Gorman, B.; Fluegemann, J.; Biswas, R.; and Smelyanskiy, V. N. 2015b. Determination and correction of persistent biases in quantum annealers. *arXiv:1503.05679 [quant-ph].*

Péridy, L., and Rivreau, D. 2005. Local adjustments: A general algorithm. *European Journal of Operational Research* 164(1):24 – 38.

Pudenz, K. L.; Albash, T.; and Lidar, D. A. 2014. Error-corrected quantum annealing with hundreds of qubits. *Nature Communications* 5.

Rieffel, E.; Venturelli, D.; Do, M.; Hen, I.; and J., F. 2014. Parametrized families of hard planning problems from phase transitions.

Rieffel, E. G.; Venturelli, D.; O'Gorman, B.; Do, M. B.; Prystay, E. M.; and Smelyanskiy, V. N. 2015. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing* 14(1):1–36.

Rønnow, T. F.; Wang, Z.; Job, J.; Boixo, S.; Isakov, S. V.; Wecker, D.; Martinis, J. M.; Lidar, D. A.; and Troyer, M. 2014. Defining and detecting quantum speedup. *Science* 345(6195):420–424.

Smelyanskiy, V. N.; Rieffel, E. G.; Knysh, S. I.; Williams, C. P.; Johnson, M. W.; Thom, M. C.; Macready, W. G.; and Pudenz, K. L. 2012. A Near-Term Quantum Computing Approach for Hard Computational Problems in Space Exploration. *arXiv:1204.2821 [quant-ph].*

Smelyanskiy, V. N.; Venturelli, D.; Perdomo-Ortiz, A.; Knysh, S.; and Dykman, M. I. 2015. Quantum annealing via environment-mediated quantum diffusion. *arXiv:1511.02581 [quant-ph].*

Streeter, M. J., and Smith, S. F. 2006. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research* 26:247–287.

Venturelli, D.; Mandrà, S.; Knysh, S.; O'Gorman, B.; Biswas, R.; and Smelyanskiy, V. 2015. Quantum Optimization of Fully Connected Spin Glasses. *Physical Review X* 5(3):031040.

Venturelli, D.; Marchand, D. J. J.; and Rojo, G. 2015. Quantum Annealing Implementation of Job-Shop Scheduling. *arXiv:1506.08479 [quant-ph].*

Vinci, W.; Albash, T.; Paz-Silva, G.; Hen, I.; and Lidar, D. A. 2015. Quantum annealing correction with minor embedding. *Physical Review A* 92(4):042310.

Wagner, H. M. 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly* 6(2):131–140.

Assessment of a multi agent system for energy aware off-line scheduling from a real case manufacturing data set

Giancarlo Nicolò, Miguel Salido, Adriana Giret, Federico Barber
Universidad Politécnica de Valencia,
Camino de vera s/n, 46022,
Valencia, Spain
giani1@dsic.upv.es, msalido@dsic.upv.es, agiret@dsic.upv.es, fbarber@dsic.upv.es

Abstract

State-of-the-art approaches to energy aware scheduling can be centralized or decentralized, predictive or reactive, and they can use methods ranging from optimal to heuristic. In this paper an agent-based distributed model is proposed for off-line scheduling in energy intensive manufacturing systems, by using a real industrial case, specifically manufacturing by injection moulding. A multi-objective scheduling problem requiring the minimization of the total job tardiness, total setup times and energy consumption is faced. The multi-agent approach is evaluated respect to its internal solving strategy (optimal or heuristic) and compared with a centralized approach. Advantages and drawbacks are pointed out for off-line energy-aware scheduling, giving useful reflection on how to face the field with new techniques.

1 Introduction

Nowadays, industrial sustainability plays a fundamental role within manufacturing systems. Accordingly, energy efficiency (EE) interventions are increasingly gaining practical interest as a component of sustainability strategy (Seow and Rahimifard 2011; Tonelli, Evans, and Taticchi 2013). Energy aware scheduling is considered a fundamental issue for sustainable manufacturing implementation in order to improve efficiency of input energy usage and energy consumption (Bruzzone et al. 2012; Dai et al. 2013; Salido et al. 2015).

Current and emergent state-of-the-art approaches to energy aware scheduling can be centralized or decentralized, predictive or reactive, and they can use optimal or heuristic methods (Paolucci, Anghinolfi, and Tonelli 2015; Pach et al. 2014; Tang et al. 2015).

In this paper, an agent-based distributed model is evaluated for off-line scheduling through its application to energy intensive manufacturing systems by using a real industrial case. In particular, one of the most widespread manufacturing industry, plastic production by injection moulding, which is also one of the greatest industrial energy consumer ($2.06 \cdot 10^8$ GJ per year only in USA) is considered.

The faced problem consists in scheduling off-line a set of orders on a set of parallel injection moulding presses, where each order is characterized by a product type and a penalty cost for late delivery. A set of alternative presses is available for each order and both the processing time and the energy consumption depend on the order-machine pair. Since mould change and cleaning are required between two successive operations on the same injection press, also setup times must be considered.

Accordingly, the examined case is a multi-objective scheduling problem in which the total tardiness, total setup time and total energy consumption must be minimized (Lu et al. 2012). Incidentally, the structure of the studied case is analogous to other manufacturing optimization problems characterized by the scheduling of independent jobs on unrelated parallel machines with sequence and machine dependent setups (Allahverdi 2015). Since this problem includes as a special case well-known computational intractable problems, such as the single machine total weighted tardiness problem (Lawler 1977), the discussed case belongs to the class of NP-hard problems.

Among the possible alternative methods to face multi-objective scheduling problems, a distributed approach exploiting a multi-agent system (MAS) is proposed and evaluated in terms of his internal solving strategy. From that evaluation, a comparison with a centralized approach, based on mixed integer programming (MIP) proposed in (Paolucci, Anghinolfi, and Tonelli 2015), is done in order to assess their peculiarities and establish criteria for a thorough choice between the two approaches. This paper provides an experimental evaluation, considering efficiency, scalability and solution quality, applied to the off-line energy aware scheduling problem.

2 Problem description

The presented problem consists in scheduling a set of orders, each corresponding to a job, on a set of unrelated parallel machines (injection moulding presses). Each job has a release date, namely, the earliest start date for processing, and a due date. The jobs have different priority, expressed by weights that are used to penalize the jobs tardiness, i.e. the delay of the orders after their due date. The jobs are processed by the machines selected from a set of alternative ones. The processing time and energy consumption of the jobs depend on the selected machine, specifically different presses may need different quantities of energy to carry out the same job: a new machine can be faster and can require less energy than an older. In order to process a job the machines must be setup by changing the mould, cleaning the machinery etc. A setup time between two successive jobs producing different products on the same machine is considered; setup times may also depend on the kind and peculiarities of the machine; consequently setup time depends on the machine and job sequence.

The problem is multi-objective since three different objective functions must be simultaneously optimized: the total weighted tardiness of the jobs $TT(s)$, the total energy consumption $EN(s)$ and the total setup time $ST(s)$. The solution s^* can be obtained by minimizing a 3-dimensional objective function:

$$s^* = \arg\min_{s \in S} [TT(s), EN(s), ST(s)] \tag{1}$$

where S denotes the feasibility space for the problem solutions.

To represent the model and the three components of the objective function to optimize($TT(s)$, $EN(s)$ and $ST(s)$), a list of notations used for sets, constant parameters and variables used along the paper is presented.

Sets:

- $J = \{1, \ldots, n\}$ the set of jobs, indexes 0 and $n+1$ denote two fictious jobs corresponding to the first and last jobs on each machine
- $M = \{1, \ldots, m\}$ the set of machines (i.e. the presses)
- $M_j, \forall j \in J$, the set of machines that can execute job j
- $J_k, \forall k \in M$, the set of jobs that can be executed by machine k

Parameters:

- B a suffiently large constant
- $D_j, \forall j \in J$, the due date of job j
- $R_j, \forall j \in J$, the release date of job j
- $W_j, \forall j \in J$, the tardiness penalty of job j
- $P_{jk}, \forall j \in J, \forall k \in M_j$, the processing time of job j on the eligible machine k
- $E_{jk}, \forall j \in J, \forall k \in M_j$, the energy consumption for processing job j on the eligible machine k
- $S_{ijk}, \forall i, j \in J, \forall k \in M_j,, i \neq j$ the setup time on machine k between the completion of job i and the start of the subsequent job j
- $\Pi_g, g = 1, 2, 3$ the weights of the objective function components (i.e. total weighted tardiness, total energy consumption and total setup time)

Variables:

- $c_j, \forall j \in J \cup \{0\}$, the completion time of job j
- $t_j, \forall j \in J$, the tardiness of job j with respect to its due date
- $x_{ijk} \in \{0, 1\}, \forall i, j \in J \cup \{0, n+1\}, k \in M_i \cap M_j, i \neq j$ binary sequencing variables (i.e. $x_{ijk} = 1$ denotes that job i immediately precedes job j on machine k)
- $y_{jk} \in \{0, 1\}, \forall j \in J, k \in M_j$, binary assignment variables (i.e. $y_{jk} = 1$ denotes that job j is processed by machine k)

Following the approach proposed in (Paolucci, Anghinolfi, and Tonelli 2015), the three objective functions in (1) are aggregated into a scalar function. Since the objective components in (1) have different dimensions (time and energy) their conversion to a common dimension (e.g. cost) may not always be practical. Moreover, decision makers may have difficulties to express preference information through numerical weights, considering the original dimension of the objective function components. Thereby, a minimum deviation method is adopted to aggregate the three components in the following normalized scalar objective function F:

$$F(s) = \sum_{g=1}^{3} \Pi_g \cdot \frac{f_g(s) - f_g^-}{f_g^+ - f_g^-} \tag{2}$$

where $f_g(s)$, $g \in \{1, 2, 3\}$, represents the three objective function components, $TT(s)$, $EN(s)$ and $ST(s)$ that are expressed as a function of the model variables as follows:

$$TT(s) = \sum_{j \in J} W_j \cdot t_j \tag{3}$$

$$EN(s) = \sum_{j \in J} \sum_{k \in M_j} E_{jk} \sum_{\substack{i \in J_k \\ i \neq j}} x_{ijk} \tag{4}$$

$$ST(s) = \sum_{k \in M} \sum_{i \in J_k} \sum_{\substack{j \in J_k \\ i \neq j}} S_{ijk} \cdot x_{ijk} \tag{5}$$

The quantity f_g^- in (2) represents the best (i.e. minimum) value for the g-th component when it is optimized individually; f_g^+ is an estimation of the worse value for $f_g(s)$ that can be fixed as $f_g^+ = \max_{h \neq g} f_g(s_h^*)$, where (s_h^*) is the optimal solution found when the objective $f_h(s)$ is individually optimized. The weights Π_g, $g \in \{1, 2, 3\}$, in (2) express the relative importance given by the decision maker to the different objective components and are selected such that $\sum_g \Pi_g = 1$.

3 Multi-agent system model

To find a solution to the proposed multi-objective function (1) for the problem under observation, a multi-agent system is introduced as a decentralized approach for the solution searching.

The overall idea under the proposed MAS model is to use an intelligent master agent to decompose the problem into sub-problems and delegate to intelligent agents (i.e. solver agent) the solving of each sub-problem. Then, the master agent composes a global solution from the partial solutions provided by each solver agents.

In the following, the architecture of the MAS is specified; the interaction sequence to solve the problem is described together with the detailed set of messages exchanged between the agents, and the internal functions of the agents to compute the solutions is explained.

3.1 MAS architecture

The proposed multi-agent system follows a standard architecture where is possible to identify two different type of agents: *master* and *solver*.

The number of master agents is fixed a priori to one, while, the number of solver agents depends on the problem instance and is duty of the master to instantiate the right number of solver agents for the system.

Solver agent. Each solver agent $a \in A = \{1, \ldots, m\}$ is committed to solve a partial problem by scheduling all jobs assigned to an individual machine $m \in M$. A solver agent receives its partial problem ($DataSet_{J_m}$) and build the partial solution choosing between two type of strategies:

- use an optimal approach implementing the MIP model proposed in the section 4.5 (adapting the description of the global problem to the partial one straightforwardly).
- use an heuristic approach from one of the greedy heuristics described in section 4.

To reach a fair evaluation between solving strategies, the strategy to follow for each solver agent is fixed to be the same for all agents at the beginning of the execution.

Master agent. Master agent is an agent committed to distribute the problem into sub-problems and carry out the coordination among the solver agents to compose the global solution. To this end, the master agent receives the problem instance specification and generates as many solver agents as parallel machines are involved in the problem. Then, it assigns to solver agent $k \in A$ all jobs J_k that can only be executed in machine $k \in M$. Every job j that can be processed by different machines M_j is analyzed in terms of processing time PT, energy consumption EN and balance constraints to each available machine $m \in M_j$ in order to decide which machine (solver agent) will manage it. Thus, the master agent selects for each job j that can be managed by different machines M_j the machine/solver agent that minimizes the following expression:

$$m' = \arg\min_{m \in M_j} \left[\frac{P_{jm}}{\max_{m \in M_j}(P_{jm})} + \frac{E_{jm}}{\max_{m \in M_j}(E_{jm})} \right] \quad (6)$$

Moreover, the master agent is able to balance the workload of solver agents to avoid a bottleneck or energy constraint in a given solver agent. Once all jobs are distributed among the solver agents, the master agent is able to determine the total energy consumption of the resultant solution. This is due to the fact that all jobs have been distributed and thus the energy consumption of each job is assigned. This feature is fundamental for the MAS model since the user knows in advance the total energy consumption required by each machine. Thus, if there are energy constraints for the machines, these constraints can be included in the master agent knowledge.

3.2 Distributed solving sequence

Figure 1 depicts the interaction sequence among the agents. The list of exchanged message and their sequence are overviewed in this figure. The vertical line (from each agent) in

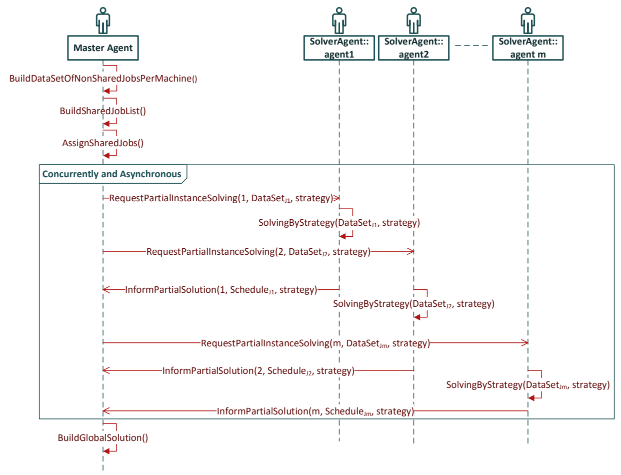

Master Agent
SolverAgent:: agent1
SolverAgent:: agent2
SolverAgent:: agent m
BuildDataSetOfNonSharedJobsPerMachine()
BuildSharedJobList()
AssignSharedJobs()
Concurrently and Asynchronous
RequestPartialInstanceSolving(1, DataSet$_{J1}$, strategy)
SolvingByStrategy(DataSet$_{J1}$, strategy)
RequestPartialInstanceSolving(2, DataSet$_{J2}$, strategy)
InformPartialSolution(1, Schedule$_{J1}$, strategy)
SolvingByStrategy(DataSet$_{J2}$, strategy)
RequestPartialInstanceSolving(m, DataSet$_{Jm}$, strategy)
InformPartialSolution(2, Schedule$_{J2}$, strategy)
SolvingByStrategy(DataSet$_{Jm}$, strategy)
InformPartialSolution(m, Schedule$_{Jm}$, strategy)
BuildGlobalSolution()

Figure 1: Interaction sequence to solve the distributed scheduling problem

the diagram represents time and the execution thread of each agent. An arrow between two agents represents a message passing from one agent to the other (i.e. the sending agent is requesting the execution of a given function from the receiving agent, or the sending agent is informing the receiving agent a given data is ready).

The solving sequence is initiated by the master agent executing the function $BuildDataSetOfNonSharedJobsPerMachine()$, in order to compose a partial problem ($DataSet_{J_m}$) for each machine m,

$DataSet_{J_m} = \{j, D_j, R_j, T_j, P_{jm}, E_{jm}, S_{ijm}, \Pi_g\}$ $\forall j \in J_m$

When building the partial problem only the jobs that can just be executed in the given machine are considered.

In a second step the master agent executes the $BuildSharedJobsList()$ function to compose a list of the shared jobs: the jobs that can be processed on alternative machines:

$SharedList = \{< j, M_j >\} \forall j \in J, |M_j| > 1$

In the third step, the master agent executes the function $AssignSharedJobs()$. This function operates as follow:

1. For each shared job in $SharedList$, it analyses the processing time and energy consumption of the pair $< job, machine >$ according to equation 6 and selects the machine with the best performance.
2. This shared job is assigned to the selected machine, appending the job to the $DataSet_{J_m}$ with its corresponding processing time and energy consumption.

The following steps in the distributed sequence are a set of concurrent and asynchronous message exchanged between the master and solver agents:

1. The master agent sends to every solver agent m the message $RequestPartialInstanceSolving(m, DataSet_{J_m}, strategy)$ asking to solve the partial problem with a predefined strategy.

2. The solver agents react to the previous masters message executing the required solving strategy and returning values to their local variables ($Schedule_{J_m}$).
3. Every solver agent m returns to the master agent the computed schedule (an assignment of its local variables) for the partial problem by means of the message $InformPartialSolution(m, Schedule_{J_m}, strategy)$.

Once all the partial solutions are received by the master agent, the function $BuildGlobalSolution()$ generates the global schedule s from the different partial schedules $Schedule_{J_m}$ and the global multi-objective function F is calculated according to formula 2.

4 Solving strategies

As described in the sub-section (3.1), the possible strategies that a solver agent can use to solve his assigned $DataSet_{J_m}$ are divided in two category: optimal and heuristic.

Different heuristic strategies are proposed, mostly based on a greedy procedure that evaluates a best choice at each searching step with no guarantee to found an optimal solution, but with an efficient time consumption proportional to the size of the problem. In contrast to the previous, the used optimal method is based on mathematical integer programming (Paolucci, Anghinolfi, and Tonelli 2015), a method that guarantee to find an optimal solution in a time that could be exponential in the size of the problem.

In the following each solving strategy is described and where possible, an illustrative pseudo-code is listed.

4.1 Naive heuristic

This *naive* heuristic follows a different approach from the standards, because rather than focusing on the objective function to minimize, takes under analysis a problem feature, the idle time of the machines, and try to minimize its value and evaluate how the solution quality will be affected.

At the operational level, after analyzed the $DataSet_{J_m}$, the solver agent executes a greedy algorithm that creates the scheduling solution $[job_1, \dots, job_{|J_m|}]$ iteratively, deciding at each step the job_x, from the J_m, that will be assigned at the position i-th of the solution. The decision step follows the rule of choosing, from the remaining jobs to schedule, the one with the earliest release time.

In the following, the pseudo-code for the naive heuristic is reported:

```
Data: list of JobToSchedule $J_m$
Result: $Schedule_{J_m}$
$J'_m = J_m$;
while $J'_m \neq$ EmptyList do
    find $j_x = \min_{j \in J'_m} R_j$;
    add $j_x$ to $Schedule_{J_m}$;
    remove $j_x$ from $J'_m$;
end
```

Algorithm 1: Naive heuristic.

4.2 Completion time greedy heuristic

This greedy heuristic has a similar behavior to the naive one, but rather than considering a problem feature, it focus its analysis on the possible completion time of the jobs to schedule.

At the operational level, this heuristic follows the same approach of the previous (iteratively creates the scheduling solution), changing the rule in the decision step. In this case, for each job in the list of remaining jobs to schedule, it is calculated the *starting time of execution* as the maximum value between the completion time of the previously scheduled job and the release time of the job under examination. Then, it's calculated the possible completion time for each remaining job as the sum of the *starting time of execution* plus the processing time (a constant value from the problem data). After that, the job with the minimum value of completion time is selected.

In the following, the pseudo-code for the completion time heuristic is reported:

```
Data: list of JobToSchedule $J_m$
Result: $Schedule_{J_m}$
$J'_m = J_m$;
while $J'_m \neq$ EmptyList do
    find $j_x = \min_{j \in J'_m} c_j$;
    add $j_x$ to $Schedule_{J_m}$;
    remove $j_x$ from $J'_m$;
end
```

Algorithm 2: Greedy heuristic based on completion time.

4.3 Weighted tardiness greedy heuristic

This greedy heuristic, differently from the others, tries to minimize the multi-objective function 1, focusing on the total weighted tardiness objective function 3 and aiming to minimize it at each decision step. To do so, it chooses as the job_x to schedule, the one with the maximum calculated weighted tardiness. This decision is made on the assumption that postponing the schedule of the chosen job_x will lead to a highest worsening for the total weighted tardiness objective function.

At the operational level, this heuristic follows the same approach of the previous ones (iteratively create the scheduling solution), changing the rule for the decision step. Similarly to the heuristic in section 4.2, for each job in the list of remaining jobs to schedule, it is calculated the *starting time of execution* and the possible completion time. Then, the tardiness value, respect to the job due date, is calculated and weighted with the weight cost (a constant value from the problem data). After that, the job with the maximum value of weighted tardiness is chosen, if there is a tie, the job with the earliest release time is chosen.

In the following, the pseudo-code for the weighted tardiness heuristic is reported:

After a brief reflection, it is possible to note that this heuristic can lead to a large idle time for the machine (e.g. it is selected to schedule as job_i, a job with a starting execution time far away from the last job completion time). To improve

```
Data: list of JobToSchedule $J_m$
Result: $Schedule_{J_m}$
$J'_m = J_m$;
while $J'_m \neq$ EmptyList do
    find $j_x = \max_{j \in J'_m} W_j \cdot t_j$;
    add $j_x$ to $Schedule_{J_m}$;
    remove $j_x$ from $J'_m$;
end
```

Algorithm 3: Greedy heuristic based on the weighted tardiness.

this, an extended version of this heuristic is presented in the next section.

4.4 Weighted tardiness greedy heuristic with backward searching

This greedy heuristic extends the previous one (section 4.3) using a backward search, after the greedy decision step, aiming at reducing large idle times for the machines.

At operational level, the backward search tries to find a $job_{i'}$ that can be scheduled before the job_i chosen from the previous decision step. This $job_{i'}$ has to fulfill a new temporal constraint that ensures the *starting time* of the previous fixed job (job_i) does not change, due to the new setup cost between $job_{i'}$ and job_i. If there are more than one job that satisfy the temporal constraint, the job with the maximum weighted tardiness is chosen. The backward search is recursively invoked till it's possible to find a job that fulfill the new temporal constraint. Otherwise the heuristic moves forward to find a new job to schedule after the previously fixed job, job_i.

In the following, the pseudo-code for the weighted tardiness heuristic with backward search is reported:

```
Data: list of JobToSchedule $J_m$
Result: $Schedule_{J_m}$
$J'_m = J_m$;
while $J'_m \neq$ EmptyList do
    find $j_x = \max_{j \in J'_m} W_j \cdot t_j$;
    $TC(j_x)$ = temporal constraint for $j_x$;
    remove $j_x$ from $J'_m$;
    $s'$ = BackwardSearch($J'_m$,$TC(j_x)$);
    add $s'$ to $Schedule_{J_m}$;
    add $j_x$ to $Schedule_{J_m}$;
end
```

Algorithm 4: Greedy heuristic based on the weighted tardiness with backward search.

4.5 Mathematical programming model

The mixed integer programming (MIP) model uses the mathematical formulation described in section 2 extending it with the mathematical definition of the objective functions (7) and their constraint (8-17) as follow:

```
Data: remaining JobToSchedule $J'_m$, temporal
      constraint $TC(j_x)$
Result: sub-scheduling solution $s'$
while $J'_m \neq$ EmptyList $\wedge \exists j' | satisfy(j', TC(jb_x))$ do
    find $j'_x = \max_{j' \in J'_m} W_{j'} \cdot t_{j'}$;
    $TC(j'_x)$ = temporal constraint for $j'_x$;
    remove $j'_x$ from $J'_m$;
    $s'$ = BackwardSearch($J'_m$,$TC(j'_x)$);
    add $s'$ to $s$;
    add $j_x$ to $s$;
end
```

Algorithm 5: Backward search.

$$\min \Pi_1 \cdot \frac{\sum_{j \in J} W_j \cdot t_j - f_1^-}{f_1^+ - f_1^-} + \Pi_2 \cdot \frac{\sum_{j \in J} \sum_{k \in M_j} E_{jk} \sum_{\substack{i \in J_k \\ i \neq j}} x_{ijk}}{f_2^+ - f_2^-} + \Pi_3 \cdot \frac{\sum_{k \in M} \sum_{i \in J_k} \sum_{\substack{j \in J_k \\ i \neq j}} S_{ijk} \cdot x_{ijk}}{f_3^+ - f_3^-} \tag{7}$$

subject to

$$\sum_{\substack{i \in J_k \\ i \neq j}} x_{ijk} = y_{jk} \qquad \forall j \in J, k \in M_j \tag{8}$$

$$\sum_{\substack{j \in J_k \\ j \neq i}} x_{ijk} = y_{ik} \qquad \forall i \in J, k \in M_i \tag{9}$$

$$\sum_{k \in J_k} y_{jk} = 1 \qquad \forall j \in J \tag{10}$$

$$\sum_{j \in J_k} x_{0jk} \leq 1 \qquad \forall k \in M \tag{11}$$

$$c_j \geq R_j + \sum_{k \in M_j} P_{jk} y_{jk} \qquad \forall j \in J \tag{12}$$

$$t_j \geq c_j - D_j \qquad \forall j \in J \tag{13}$$

$$c_j \geq c_i + P_{jk} + S_{ijk} - B \cdot (1 - x_{ijk}) \quad \forall k \in M, \forall i, j \in J_k, i \neq j \tag{14}$$

$$c_0 = 0 \tag{15}$$

$$c_j \geq 0, t_j \geq 0 \tag{16}$$

$$\begin{aligned} x_{ijk} &\in \{0,1\} && \forall i, j \in J, i \neq j, k \in M_i \cap M_j, \\ y_{jk} &\in \{0,1\} && \forall j \in J, k \in M_j \end{aligned} \tag{17}$$

Constraints (8) and (9) impose that each job, assigned to a machine, must be sequenced on that machine, precisely it must have a predecessor and a successor on the machine. The job preceded by the fictitious job 0 is the first job on a machine, whereas the job followed by job $n+1$ is the last one on the machine. Constraints (10) guarantee that each job is

Table 1: Average multi-objective values reached in the different classes of instances.

Instances		Multi-objective value					
n	m	MAS					Centralized MIP
		Naive	CT	WT	WT+BT	MIP	
30	4	0.09782	0.06984	0.40928	0.16874	0.02642	0.01978
50	6	0.07509	0.06043	0.30887	0.12797	0.02056	0.01672
100	10	0.06299	0.05182	0.18418	0.08604	0.01781	0.01531
250	20	0.03963	0.03754	0.16438	0.03575	0.03982	0.03987

Table 2: Average runtime measurement in the different classes of instances.

Instances		Runtime [s]					
n	m	MAS					Centralized MIP
		Naive	CT	WT	WT+BT	MIP	
30	4	0.0001	0.0005	0.0006	0.0003	12.2094	319.7636
50	6	0.0009	0.0003	0.0016	0.0013	17.4823	766.4818
100	10	0.0024	0.0062	0.0049	0.0048	56.6454	1957.126
250	20	0.0155	0.0348	0.0308	0.0687	41.8345	2647.119

assigned to a single machine among the ones eligible to process it. Constraints (11) impose that, at most, a single job is the first scheduled on each machine. Constraints (12) define the lower bound for the job completion time, and (13) define the job tardiness. Constraints (14) control the job completion times ensuring that each machine processes one job at a time and that the setup time between two successive jobs is satisfied. Constraints (15) fix the completion time for the dummy job_0 and (16) and (17) define the problem variables.

5 Experimental comparison

In order to evaluate the solving strategy in terms of efficiency, scalability and solution quality, a set of random instances were generated starting from real data acquired by studying a large plastic injection moulding factory of a company leader in supplying plastic trigger sprayers and pump dispensers (Salido, Barber, and Nicoló 2016). For each moulding cycle, the injection presses produce multiple pieces, according to the selected stamp tool, with electrical power between 46 and 69 kW. These instances are grouped by the number of jobs n and involved machines m. Four different sets of 125 instances were generated and run for all the proposed solving strategies of the distributed model on a 2.4 GHz Intel Core 2 Duo. The calculated average values for each instances run are compared with the values from the centralized MIP system (Paolucci, Anghinolfi, and Tonelli 2015) and shown in Tables 1, 2, 3, 4, 5 and 6. As a facilitation to read the values, each cell that present the best value for an instance set, has been colored with shaded gray.

Tables 1 and 2 summarizes the average multi-objective value (dimensionless) and the runtime. The weights expressing the relative importance of the objectives in (2) were fixed to $\Pi_1 = 0.6$, $\Pi_2 = 0.35$ and $\Pi_3 = 0.05$ according to the preference elicitation method introduced in (Paolucci, Anghinolfi, and Tonelli 2015).

Both MIP solving strategy for the distributed and centralized system obtained a similar multi-objective value for all instances (both were set with a similar timeout of 3600 seconds). It must be observed that the values were close to 0, since (7) uses a minimum deviation method, i.e. each of the three single objectives is compared with the best solution found by solving them individually. However, due to the complexity of the problem, the runtime for the centralized system had an exponential behavior, whereas the MAS model with MIP solving strategy had a lower value of two magnitude orders. The distributed system indeed solved all instances in less than 290 seconds, whereas centralized system aborted the execution in a significant number of instances (Timeouts in Table 3). The centralized system achieved the optimal solution in $63,8\%$ of the total instances within the established timeout.

Tables 4, 5 and 6 show the percentage variation of the different objectives (tardiness, setup time and energy consumption) of the distributed MAS model against centralized MIP approach. The MAS model was able to obtain a better behavior in total tardiness and energy consumption in almost all instances, whereas the centralized MIP model returned better values for the setup times. This is due to the fact that the master agent selects the jobs that can be assigned to different machines (shared jobs) according to the

Table 3: Reached timeouts in the different classes of instances.

Instances		Timeouts	
n	mn	MAS*	Centralized MIP
30	4	0	9
50	6	0	22
100	10	0	64
250	20	0	86

Table 4: Percentage variation from the centralized system of the average total weighted tardiness objective function.

Instances		MAS vs Centralized MIP [$\%TT$]				
n	m	Naive	CT	WT	WT+BT	MIP
30	4	-29.34%	-2.03%	119.91%	25.41%	4.46%
50	6	-39.97%	-53.82%	23.62%	-29.2%	-32.31%
100	10	-29.20%	-41.04%	50.76%	-17.03%	-24.23%
250	20	-80.49%	-80.92%	-58.73%	-75.40%	-71.84%

Table 5: Percentage variation from the centralized system of the average total energy consuption objective function.

Instances		MAS vs Centralized MIP [$\%EN$]				
n	m	Naive	CT	WT	WT+BT	MIP
30	4	-0.08%	-0.08%	-0.08%	-0.08%	-0.08%
50	6	-0.13%	-0.13%	-0.13%	-0.13%	-0.13%
100	10	-0.08%	-0.08%	-0.08%	-0.08%	-0.08%
250	20	-5.66%	-5.66%	-5.66%	-5.66%	-5.66%

Table 6: Percentage variation from the centralized system of the average total setup time objective function.

Instances		MAS vs Centralized MIP [$\%ST$]				
n	m	Naive	CT	WT	WT+BT	MIP
30	4	166.6%	162.3%	159.99%	161.96%	7.31%
50	6	206.94%	206.78%	205.70%	205.76%	11.24%
100	10	274.85%	273.75%	271.16%	273.45%	12.93%
250	20	230.71%	230.07%	232.77%	229.08%	31.66%

energy consumption and processing time. However the master agent cannot consider the setup time because the machine sequence is not known in advance. In any case, the improvement is mainly significant in tardiness values.

6 Conclusion

While several approaches for off-line energy-aware scheduling have been presented in literature, the lack of a benchmark prevents a sound comparison of alternative methods. In this paper the availability of a set of instances matching the statistics computed for a real industry, specifically the production of plastic components by injection moulding, permitted to evaluate a multi-agent approach with different solving strategies and compare it with a centralized approach using a mixed integer programming model.

The comparative analysis of the experimental results of these approaches allows to decide under which conditions, such as problem size, temporal constraints, etc., one approach is better than the other. Actually the multi-agent approach shows a better performance when obtaining optimized solutions for large-scale instances in a given execution time. Anyway new heuristic and metaheuristic solving strategy (GRASP, Genetic Algorithms, etc) can be embedded in the proposed multi agent system. They are incomplete techniques but they achieve good solutions in an efficiency way. Moreover, these techniques can also be designed into a centralized system giving the possibility to compare a single technique under both centralized and distributed perspective.

To conclude, MAS model seems to be a suitable approach with interesting potentialities to be used for energy aware off-line scheduling in addition to their well-known ability to react in real-time.

References

Allahverdi, A. 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research* 246(2):345–378.

Bruzzone, A.; Anghinolfi, D.; Paolucci, M.; and Tonelli, F. 2012. Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops. *CIRP Annals-Manufacturing Technology* 61(1):459–462.

Dai, M.; Tang, D.; Giret, A.; Salido, M. A.; and Li, W. D. 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing* 29(5):418–429.

Lawler, E. L. 1977. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of discrete Mathematics* 1:331–342.

Lu, N.-y.; Gong, G.-x.; Yang, Y.; and Lu, J.-h. 2012. Multi-objective process parameter optimization for energy saving in injection molding process. *Journal of Zhejiang University SCIENCE A* 13(5):382–394.

Pach, C.; Berger, T.; Sallez, Y.; Bonte, T.; Adam, E.; and Trentesaux, D. 2014. Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields. *Computers in Industry* 65(3):434–448.

Paolucci, M.; Anghinolfi, D.; and Tonelli, F. 2015. Facing energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiency in a moulding industry. *Soft Computing* 1–12.

Salido, M. A.; Barber, F.; and Nicoló, G. 2016. Data sets for evaluation of resource constrained scheduling in a plastic moulding factory. http://gps.webs.upv.es/open-shop-energy/.

Salido, M. A.; Escamilla, J.; Giret, A.; and Barber, F. 2015. A genetic algorithm for energy-efficiency in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology* 1–12.

Seow, Y., and Rahimifard, S. 2011. A framework for modelling energy consumption within manufacturing systems. *CIRP Journal of Manufacturing Science and Technology* 4(3):258–264.

Tang, D.; Dai, M.; Salido, M. A.; and Giret, A. 2015. Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. *Computers in Industry*.

Tonelli, F.; Evans, S.; and Taticchi, P. 2013. Industrial sustainability: challenges, perspectives, actions. *International Journal of Business Innovation and Research* 7(2):143–163.