

Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems

Philippe Laborie, Daniel Godard

ILOG, 9 rue de Verdun, 94253 Gentilly, France, {plaborie, dgodard}@ilog.fr

Providing robust scheduling algorithms that can solve a large variety of scheduling problems with good performance is one of the biggest challenge of practical schedulers today. In this paper we present a robust scheduling algorithm based on Self-Adapting Large Neighborhood Search and apply it to a large panel of single-mode scheduling problems. The approach combines Large Neighborhood Search with a portfolio of neighborhoods and completion strategies together with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved. The algorithm is evaluated on a set of 21 scheduling benchmarks, most of which are well established in the scheduling community. Despite the generality of the approach, for 17 benchmarks out of 21, its mean relative distance to state-of-the-art problem specific algorithms is less than 4%. It even outperforms state-of-the-art problem-specific algorithms on 7 benchmarks clearly showing that our algorithm offers a valuable compromise between robustness and performance.

Keywords: Constraint Logic Programming, Heuristic Search, Local Search, Machine Scheduling, Meta-heuristic Search, Production Scheduling, Real World Scheduling, Shop-Floor Scheduling.

1 Introduction

There exists a large variety of scheduling problems and scheduling applications each of them featuring different types of resources, different types of temporal network topology (jobs, precedence network, work breakdown structure), different objective functions, etc. Facing this variability, the scheduling literature is huge. Most of it is about identifying or providing theoretical or experimental results on a particular type of scheduling problem. For a given well identified problem, for instance the job-shop scheduling or resource-constrained project scheduling (RCPSP) problems, extremely efficient optimization algorithms are available. Experimental evaluations are usually based on a set of specific benchmarks for the problem being studied which also explains the large number of benchmarks available to the scheduling community.

Still, when one is faced with a practical scheduling application the gap between the problem to be solved and state-of-the-art problem specific algorithms is usually too large. It requires an advanced expertise in scheduling to assess their potential applicability and efficiency on the real problem and to adapt them when possible. It explains why actual scheduling applications tend to use in-house heuristics rather than very efficient but often too specific optimization algorithms.

Providing robust scheduling algorithms that can solve a large variety of scheduling problems with good performance is still a challenge. In this paper we present a robust scheduling algorithm based on Self-Adapting Large Neighborhood Search. Section 2 describes the class of scheduling problem we focus on which covers a large panel of single-mode scheduling problems. The algorithm itself is presented in section 3. Section 4 reports an experimental study over 21 scheduling benchmarks most of which are well established in the scheduling community (e.g. job-shop, RCPSP). We show among other things that for 17 benchmarks out of 21, the mean relative distance to state-of-

the-art problem-specific algorithms is less than 4% and that our approach, despite its generality, outperforms the state-of-the-art on 7 benchmarks.

2 Model

Although the algorithm described in section 3 can easily be extended to handle complex scheduling problems involving, for instance, multiple modes, resource minimal capacities or calendars, we focus in this paper on its application to a more restricted but still expressive class of single-mode scheduling problems involving the following features:

- *Non-preemptive activities of fixed or variable duration.* $\mathcal{A} = \{A_1, \dots, A_n\}$ denotes the set of activities of the schedule. Each activity can be specified a release date (minimal start time) and a deadline (maximal end time).
- *General temporal network.* If x_i and x_j denote two time-points (start or end time of some activity), any temporal constraint $x_i - x_j \in [D_{ij}^{min}, D_{ij}^{max}]$, $D_{ij}^{min}, D_{ij}^{max} \in \mathbb{Z}$ can be expressed.
- *Capacity resources (unary, discrete, discrete reservoir) with maximal profiles.* Discrete resources are renewable resources of limited capacity, discrete reservoirs are resources of limited capacity that can be produced and consumed by activities [24]. Each capacity resource R_k can be associated a function $C_k : \mathbb{Z} \rightarrow \mathbb{Z}^+$ that represents its maximal capacity over time.
- *State resources.* State resources are resources whose state can change over time. Two activities requiring a given state resource to be in a different state cannot overlap in time [26].
- *Sequence-dependent setup times on unary resources.* A setup time on a unary resource R_k is specified by a setup matrix M_k with $M_k[i, j] \in \mathbb{Z}^+$ denoting the minimal time that must elapse between the end of A_i and the start of A_j when A_j is executed next to A_i on R_k .
- *Cost expressed as a sum/max aggregation of semi-convex piecewise linear (SCPL) functions on start, end and duration of activities.* A semi-convex function [21] is a function such that, if one draws a horizontal line anywhere in the Cartesian plane corresponding to the graph of the function, the set of x such that $f(x)$ is below the line forms a single interval. Some examples of SCPL functions are depicted on Figure 1.

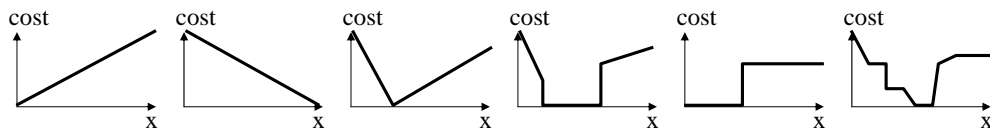


Figure 1: Example of semi-convex piecewise linear functions

Most of the classical single-mode scheduling problems (e.g. job-shop, RCPSP) and classical cost functions (e.g. makespan, earliness/tardiness costs, weighted number of late jobs, duration minimization or maximization, etc) can be represented using this model.

Note also that in this paper we focus on optimization problems rather than on feasibility problems. We assume that finding an initial feasible solution (even of bad quality) is not hard.

3 Self-Adapting Large Neighborhood Search

3.1 Overview

Large Neighborhood Search (LNS) [34] is based upon a process of continual relaxation and re-optimization: a first solution is computed and iteratively improved. Each iteration consists of a relaxation step followed by a re-optimization of the relaxed solution. This process continues until some condition is satisfied, typically, when a time limit is reached. In this paper, we generalize the randomized LNS proposed in [17] along two directions: (1) the scope of the approach is dramatically enlarged, now handling a wide variety of resource types and cost functions, and (2) the approach is robustified by using portfolios of large neighborhoods and completion strategies in combination with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved.

The overall framework of Self-Adapting LNS (denoted SA-LNS) is illustrated on Figure 2. Each large neighborhood LN_i and each completion strategy CS_j in the portfolios are associated a vector of parameters. In a parameter vector $p = (p^1, \dots, p^n)$, each parameter p^k takes its values in a finite set V^k . The learning algorithm maintains two probability distributions $prob(LN_i)$ and $prob(CS_j)$ on the sets of large neighborhoods and completion strategies and, for each parameter p^k , a probability distribution on its possible values in V^k . At each cycle of the LNS, one large neighborhood LN_i together with a corresponding vector of parameter values P_i and one completion strategy CS_j with a corresponding vector of parameter values Q_j are selected based on the current probability distributions. $LN_i[P_i]$ is applied to relax the current best solution then, completion strategy $CS_j[Q_j]$ is applied to re-optimize the relaxed solution. After this cycle, LN_i and CS_j , together with their respective parameter values P_i and Q_j are rewarded according to the efficiency of the cycle defined by the ratio $r = \Delta c / \Delta t$ where Δc is the cost improvement if any (0 otherwise) and Δt is the cycle CPU time. This type of reward tends to favor neighborhoods and strategies that quickly converge on good solutions. The reward increases the probability of the rewarded elements being chosen according to a classical re-enforcement scheme: $weight_{t+1} = (1 - \alpha) \cdot weight_t + \alpha \cdot r$ with learning rate $\alpha \in (0, 1]$ being a parameter of the global approach.

In [12], the authors present an algorithm switching strategy that iteratively runs the whole set of algorithms in the portfolio and adapts, at each cycle, their allocated running times depending on their past results. SA-LNS learns the algorithm selection rather than the algorithm running times which allows for a more fine-grain control of the search and avoids systematically running useless algorithms. The overall framework is actually closer to the one recently described in [31] for Vehicle Routing problems, the main difference being that the our approach also learns the parameter values of each component of the LNS (neighborhoods, completion strategies). That way, it can be seen as a pure black-box search without any parameter.

The sequel of this section describes the portfolios of large neighborhoods and completion strategies. Note that the first solution is built using the same search strategy as the completion strategy *SetJustInTime* described in section 3.3.

3.2 Large neighborhoods

The portfolio of large neighborhoods currently consists of 3 neighborhoods. They are all based on the initial generation of a Partial Order Schedule (POS) [32] constructed from a completely instantiated solution where activities have fixed start times and end times. A POS is a directed graph $G(\mathcal{A}, \mathcal{E})$ where the edges in \mathcal{E} are precedence constraints between activities with the property that any temporal solution to the graph is also a resource-feasible solution. Algorithms for transforming

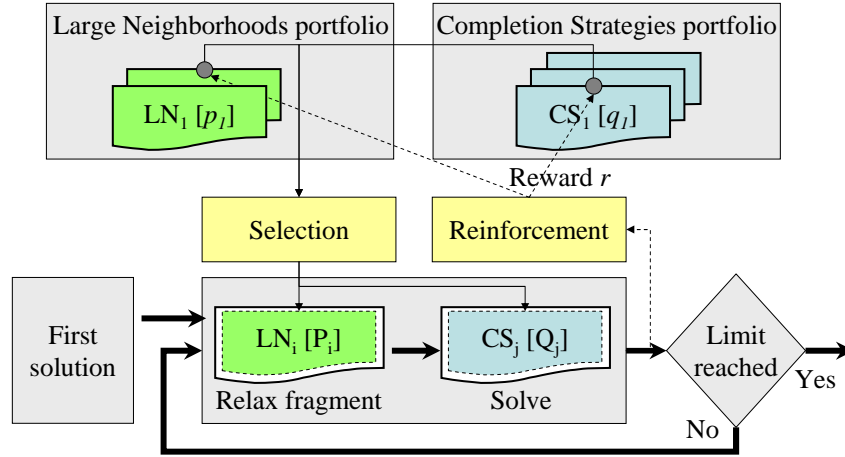


Figure 2: Self-Adapting LNS overview

a fully instantiated solution into a POS are described in [32, 17]. We extend this approach to state resources and discrete reservoirs as sketched below.

- *State resource.* The POS $P(R_k)$ of a state resource R_k contains all the edges $A_i \rightarrow A_j$ such that activities A_i and A_j require incompatible states of R_k and A_i is executed before A_j in the solution.
- *Discrete reservoirs.* The algorithm to generate a POS $P(R_k)$ for discrete reservoirs works in two steps: one that ensures that the reservoir does not underflow and the other that it does not overflow. In the first step, a simple pegging heuristic chronologically creates a directed graph of pegging arcs between producing activities and consuming activities: the first producer is pegged to the first consumer and the pegged quantity is the minimum between the produced quantity and the consumed quantity. The process continues until all consuming activities are provided enough quantity. Let $P_u(R_k)$ be the graph of pegging arcs. In the second step, the pegging arcs are used to build a sub-model to ensure that the reservoir does not overflow: each pegging arc is represented by an activity that requires the pegged quantity of a discrete resource R'_k whose capacity is the maximum capacity of the reservoir. The algorithm described in [17] is applied on this discrete resource to build a POS $P_o(R'_k)$. The POS of the discrete reservoir $P(R_k)$ is then defined as: $P(R_k) = P_u(R_k) \cup P_o(R'_k)$.

The global POS P is defined as $P = \cup_k P(R_k)$. Redundant edges in P are removed. The goal of the neighborhoods is to *select* a subset of activities that will be relaxed in the POS P . As described in [17], the relaxed POS P' is obtained by removing from P all the edges involving at least one selected activity and adding new edges to repair broken paths. The relaxed POS P' is then used to enforce precedence constraints between activities before applying a completion strategy. The portfolio contains the 3 following neighborhoods:

- *RandomizedNHood* $[\alpha_R]$. This is the neighborhood described in [17]. It randomly selects activities with a probability α_R , where α_R is a self-adapting parameter of the neighborhood.
- *TimeWindowNHood* $[\alpha_W, \beta_W]$. Activities are first sorted by increasing start times in an array. The selected activities are those whose index in the sorted array belongs to $[\beta_W \cdot n, (\beta_W + \alpha_W) \cdot n]$ where n is the number of activities of the problem, and α_W and β_W are two self-adapting parameters.

- *TopologicalNHood* $[\alpha_T, \beta_T]$. This neighborhood is similar to the previous one. It only differs in the ordering of activities. The activities are sorted in the following lexicographic order: increasing connected component¹ (CC) indexes, increasing strongly connected component (SCC) indexes, increasing start times. α_T and β_T are two self-adapting parameters. This neighborhood tends to select activities belonging to the same CC (resp. SCC) of the problem.

3.3 Completion strategies

Currently, only one completion strategy *SetJustInTime* $[\gamma]$ is used. This completion strategy explores a search tree with a maximal number of failures equal to $\gamma \cdot n$ where n is the number of activities of the problem and γ is a self-adapting parameter. At the root node, this strategy solves a linear relaxation of the problem that only takes into account activity durations, temporal constraints and a convexification of the SCPL functions of the cost. The optimal solution of this relaxation gives indicative start and end times for each activity. The search is a generalization of the *SetTimes* strategy recapped in [17]. It considers activities by increasing indicative start times and tries to schedule them as close as possible to their indicative times. When a failure occurs, in the right branch, the activity is marked "unselectable" and will remain so until constraint propagation removes from the current domain of the activity the start or end dates that were tried on the left branch. When the objective is regular, this strategy boils down to *SetTimes*. At each LNS step, the completion tries to find a solution that is not worse than the current best solution in term of cost value. If the maximal number of failures is reached before such a solution is found, a new move is tried.

4 Experimental study

SA-LNS has been implemented on top of ILOG CP 1.1 using ILOG CPLEX 10.1 for the linear relaxation of the *SetJustInTime* strategy. We report in this section a comparison of this implementation with state-of-the-art specialized algorithms on 21 scheduling benchmarks, most of which are well established in the scheduling community. It is to be noted that for this experimental study, we consider our method as a pure black-box: there is no tuning of the search for the different benchmarks. The results are summarized² on Table 1. When possible, we compare with the upper-bounds (UB) found by the best specialized algorithm on each benchmark (column "Reference UB") and try to use comparable time limits, otherwise we compare with the best known upper-bounds (which may have been found by different algorithms) and use a time-limit which is a piecewise linear function of the number n of activities, for instance 1800s on a 2GHz laptop for a problem with 500 activities. Note that due to the number of benchmarks, we often had to select a subset of instances. To ensure a fair comparison, these instances were randomly drawn. Column "MRD" measures the average relative distance to the reference upper-bound, a negative value means that in average SA-LNS outperforms the reference algorithm(s). The number of selected instances, together with the number of improved upper bounds compared to the reference algorithm(s) is given in the last column.

Over the 21 benchmarks, the worse average distance of SA-LNS is 14.7% on single-machine problems with common due-date which can be considered as reasonable for a generic approach that do not exploit problem specificities. In fact, the two worse results are single machine problems

¹Connected components and strongly connected components of the temporal network are computed from the initial set of temporal constraints in the problem. They convey important information about the temporal structuration of the problem (jobs, work breakdown structure, etc.).

²The detailed experimental protocol and results are available on scheduler.ilog.fr.

Problem type	Benchmark	Problem size	Reference UB	MRD	# Imp. UBs / # Instances
Trolley	[41]	230-460	[19]	-11.8%	15/15
Hybrid flow-shop	[35]	200-1000	[35]	-8.8%	19/20
Job-shop w/ E/T	[3]	30-200	[3]	-5.6%	32/48
Air traffic management	[19]	2000	[19]	-4.0%	1/1
Flow-shop w/ E/T	[27]	30-400	[14]	-3.0%	4/12
Max. quality RCPSP	[33]	30	[33]	-2.3%	NA/3600
Cumulative job-shop	[28]	150-675	[17]	-0.3%	27/86
Single proc. tardiness	[20]	200-500	[20]	0.2%	0/20
Semiconductor testing	[30]	400	[30]	0.2%	7/18
RCPSP w/ E/T	[42]	30-50	[42]	0.4%	15/60
Open-shop	[9, 40, 18]	64-400	[15, 7, 25]	0.9%	0/28
RCPSP	[23]	120	Best PSPLIB	1.6%	0/600 ³
Shop w/ setup times	[10]	50-200	[2]	2.3%	0/15
Parallel machine w/ E/T	[29]	8-200	[4]	2.6%	2/52
Job-shop	[1, 39, 43, 40]	100-500	Best OR-Lib	2.8%	0/33
Air land	[5]	10-50	[5]	3.4%	0/8
Flow-shop w/ buffers	[40]	100-500	[8]	3.6%	12/30
Flow-shop	[40]	100-500	Best OR-Lib	5.9%	0/22
Aircraft assembly	[16]	575	[13]	8.7%	0/1
Single machine w/ E/T	[11, 37, 29]	8-500	[38]	9.8%	1/100
Common due-date	[6]	100-200	[36]	14.7%	0/20

Table 1: Results of SA-LNS on 21 scheduling benchmarks

without any precedence constraint and SA-LNS currently does not perform any special treatment for unary resources (except for the constraint propagation done in CP1.1). Except for those two very specific scheduling benchmarks, SA-LNS is always less than 9% away from the best performing approaches and for 17 benchmarks out of 21, the mean relative distance is even less than 4%. This illustrates the exceptional robustness of the approach. Moreover SA-LNS outperforms the state-of-the-art on 7 benchmarks which is remarkable given the generality of the approach. For 10 benchmarks (all the ones for which the number of improved upper-bounds with respect to the reference is positive except *Flow-shop w/ E/T* for which we improve on the reference but not on the best bounds of the literature) SA-LNS is able to improve some best known upper bounds ever reported.

The present approach is a generalization of the randomized LNS described and experimented in [17] on cumulative job-shop problems. An interesting fact is that, in spite of this generalization, SA-LNS outperforms the results reported in [17]: 27 instances out of 86 have been improved, 9 have been worsen and the MRD to randomized LNS is -0.3%. On this benchmark, 12 of the best known upper bounds have been improved. It illustrates the added value of online learning which allows automatically tuning the algorithm on the actual instance being solved.

³The average deviation from the path-based lower bound is 32.4%. We estimate the average number of LNS cycles to be slightly less than 50000 which would position SA-LNS in the top 7 best approaches for RCPSP among the 37 approaches reviewed in [22].

5 Conclusion and future work

The Self-Adapting Large Neighborhood Search presented in this paper combines several ingredients which are fundamental to its efficiency and robustness:

- *Large Neighborhood Search*: by freezing some features of a solution and focusing on re-optimizing the unfrozen features the LNS framework provides a general and efficient traversal of the search space. Compared with Tree Search, it avoids being stuck with wrong early decisions. It is more flexible than Local Search for complex problems involving many types of constraints and resources.
- *Partial Order Schedules*: in the context of LNS, POSs provide a very powerful way to inject flexibility into the schedule while keeping interesting features from one solution to the other. As shown, the concept can be extended to various types of resources.
- *Neighborhoods*: taken individually, each of the neighborhoods described in the paper are fairly robust (see for instance [17] for *RandomizedNHood*).
- *Completion strategy*: the *SetJustInTime* completion strategy uses a linear relaxation of the problem and, doing so, has a global vision of the ideal position of activities in time would there be no resource limitation. In the context of LNS where only a part of the POS is unfrozen, this relaxation tends to be very informative as most of the resource constraints are still captured by frozen precedence arcs of the POS. The branching scheme of the strategy allows to exploit constraint propagation and better explore the bottom of the search tree which clearly is a plus compared to more classical non-backtracking greedy algorithms.
- *Learning*: the re-enforcement learning scheme, although quite simple, ensures a quick convergence on the most effective neighborhoods, completion strategies and their associated parameter values. Learning is a key factor in the robustness of the approach.

On-going and future work mainly consist in extending SA-LNS to multi-mode scheduling problems, that is, scheduling problems that have a "resource allocation" dimension (this also accounts for optional activities, alternative resources, alternative recipes or routes, etc.) and to other types of costs such as setup, resource usage or inventory costs.

References

- [1] J. Adams, E. Balas, and D. Zawack (1988), The shifting bottleneck procedure for job shop scheduling, *Management Science* **34**(3), 391 – 401.
- [2] E. Balas, N. Simonetti, and A. Vazacopoulos (2005), Job shop scheduling with setup-times, deadlines and precedence constraints, In *Proc. MISTA-2005*.
- [3] P. Baptiste, M. Flamini, and F. Sourd (2005), Lagrangean bounds and lagrangean heuristics for just in time job-shop scheduling, Technical report, Universita degli Studi di Roma Tre.
- [4] P. Baptiste and R. Sadykov (2007), A new mip formulation for single machine scheduling, In *FRANCORO/ROADEF*.
- [5] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson (2000), Scheduling aircraft landings - the static case, *Transportation Science* **34**, 180 – 197.
- [6] D. Biskup and M. Feldmann (2001), Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers and Op. Research* **28**(8), 787 – 801.

- [7] C. Blum (2005), Beam-ACO - hybridizing ant colony optimization with beam search: an application to open-shop scheduling, *Computers and Operations Research* **32**(6), 1565 – 1591.
- [8] P. Brucker, S. Heitmann and J. L. Hurink (2003), Flow-shop problems with intermediate buffers, *OR Spectrum* **25**(4), 549 – 574.
- [9] P. Brucker, J. Hurink, B. Jurisch and B. Wöstmann (1997), A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics* **76**, 43 – 59.
- [10] P. Brucker and O. Thiele (1996), A branch and bound method for the general shop problem with sequence dependent setup-times, *OR Spektrum* **18**, 145 – 161.
- [11] K. Bulbul, P. Kaminsky and C. Yano (2001), Preemption in single machine earliness/tardiness scheduling, Submitted for publication.
- [12] T. Carchrae and J.C. Beck (2005), Applying machine learning to low knowledge control of optimization algorithms, *Computational Intelligence*, **21**(4), 372 – 387.
- [13] J. Crawford (1996), An approach to resource constrained project scheduling, In *Proc. 1996 Artificial Intelligence and Manufacturing Research Planning Workshop*.
- [14] E. Danna and L. Perron (2003), Structured vs. unstructured large neighborhood search, In *Proc. CP 2003*, 817 – 821.
- [15] U. Dorndorf, E. Pesch and T. Phan-Huy (2001), Solving the open shop scheduling problem, *Journal of Scheduling* **4**, 157 – 174.
- [16] B. Fox and M. Ringer (1995), Planning & scheduling benchmarks, URL: www.neosoft.com/~benchmrx/.
- [17] D. Godard, P. Laborie and W. Nuijten (2005), Randomized Large Neighborhood Search for Cumulative Scheduling, In *Proc. ICAPS-05*, 81 – 89.
- [18] C. Guéret and C. Prins (1999), A new lower bound for the open-shop problem, *Annals of Operations Research* **92**, 165 – 183.
- [19] ILOG (2003), *ILOG OPL Studio 3.7 User's Manual and Reference Manual*, ILOG, S.A.
- [20] B. Kara (2002), SMTTP problem library, <http://www.bilkent.edu.tr/~bkara/start.html>.
- [21] L. Khatib, P. Morris, R. Morris and F. Rossi (2001), Temporal constraint reasoning with preferences, In *Proceedings IJCAI*, 322 – 327.
- [22] R. Kolisch and S. Hartmann (2006), Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem: An update, *European Journal of Operational Research* **174**, 23 – 37.
- [23] R. Kolisch and A. Sprecher (1996), PSPLIB - A project scheduling problem library, *European Journal of Operational Research* **96**, 205 – 216.
- [24] P. Laborie (2003), Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results, *Artificial Intelligence* **143**(2), 151 – 188.
- [25] P. Laborie (2005), Complete MCS-Based Search: Application to Resource Constrained Project Scheduling, In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*

(IJCAI-05).

- [26] C. Le Pape (1994), Implementation of resource constraints in ILOG Schedule, *Intelligent Systems Engineering* **3**(2), 55 – 66.
- [27] T. Morton and D. Pentico (1993), *Heuristic Scheduling Systems*, Wiley.
- [28] W. Nuijten (1994), *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, PhD thesis, Eindhoven University of Technology.
- [29] W. Nuijten, T. Bousonville, F. Focacci, D. Godard and C. Le Pape (2004), Towards an industrial manufacturing scheduling problem and test bed, In *Proc. PMS*.
- [30] I. M Ovacik and R. Uzsoy (1996), Decomposition methods for scheduling semiconductor testing facilities, *International Journal of Flexible Manufacturing Systems* **8**, 357 – 398.
- [31] D. Pisinger and S. Ropke (2005), A general heuristic for vehicle routing problems, Technical report, DIKU, University of Copenhagen.
- [32] N. Policella, A. Cesta, A. Oddi and S.F. Smith (2004), Generating robust schedules through temporal flexibility, In *Proceedings ICAPS-04*, Whistler, Canada, June.
- [33] N. Policella, X. Wang, S.F. Smith and A. Oddi (2005), Exploiting temporal flexibility to obtain high quality schedules, In *Proc. AAAI-2005*.
- [34] P. Shaw (1998), Using constraint programming and local search methods to solve vehicle routing problems, In *Proc. CP-98*, 417 – 431.
- [35] F Sivrikaya-Serifolu and G. Ulusoy (2004), Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach, *Journal of the OR Society* **55**(5), 504 – 512.
- [36] F. Sourd (2006), A reinforced lagrangean relaxation for non-preemptive single machine problem. application to the earliness-tardiness common due date criterion, In *Proc. PMS'06*.
- [37] F. Sourd and S. Kedad-Sidhoum (2003), The one machine scheduling with earliness and tardiness penalties, *Journal of Scheduling* **6**, 533 – 549.
- [38] F. Sourd and S. Kedad-Sidhoum (2005), An efficient algorithm for the earliness-tardiness scheduling problem, In *Optimization Online*.
- [39] R.H. Storer, S.D. Wu, and R. Vaccari (1992), New search spaces for sequencing problems with application to job shop scheduling, *Management Science* **38**(10), 1495 – 1509.
- [40] E. Taillard (1993), Benchmarks for basic scheduling problems, *European Journal of Operations Research* **64**, 278 – 285.
- [41] P. van Hentenryck, L. Michel, P. Laborie, W. Nuijten and J. Rogerie (1999), Combinatorial optimization in OPL Studio, In *Proc. EPIA 1999*, 1 – 15.
- [42] M. Vanhoucke, E. Demeulemeester and W. Herroelen (2001), An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem, *Annals of Operations Research* **102**(1-4), 179 – 196.
- [43] T. Yamada and R. Nakano (1992), A genetic algorithm applicable to large-scale job-shop problems, *Proc. Int. Workshop on Parallel Problem Solving from Nature*, 281 – 290.