Solving Discrete Lot-Sizing and Scheduling by Simulated Annealing and Mixed Integer Programming

Sara Ceschia, Luca Di Gaspero, Andrea Schaerf*

DPIA, University of Udine, Via delle Scienze 206, 33100 Udine, Italy

Abstract

We consider the discrete single-machine, multi-item lot-sizing and scheduling problem and we propose a Simulated Annealing (SA) approach together with a statistically-principled tuning procedure to solve it. We compare our solver with the state-of-the-art methods based on Mixed Integer Programming (MIP), both on publicly-available instances and on a set of new, more challenging ones. In addition, we propose a hybrid SA/MIP method that combines the advantages of the pure methods on the challenging instances. The outcome is that our solver is able to find near-optimal solutions in short time for all instances, including those that are not solved by MIP methods. Instances and solutions are made available on the web for inspection and future comparisons.

Key words: Lot-sizing, Scheduling, Simulated Annealing, Metaheuristics, Mixed Integer Programming

1. Introduction

Lot-sizing and scheduling (LSS) is a production planning problem that is central in the field of production economics. Several versions of the LSS problem have been proposed in the literature, depending on the specific application domain. In this work, we consider the discrete, single-machine, single-level, multi-item version of the general LSS problem with sequence-dependent setup costs. According to the notation proposed by Pochet and

*Corresponding author

Email addresses: sara.ceschia@uniud.it (Sara Ceschia), luca.digaspero@uniud.it (Luca Di Gaspero), schaerf@uniud.it (Andrea Schaerf)

Wolsey (2006), this version of the problem is referred as MI-DLS-CC-SC (*Multi Item, Discrete, Constant Capacity, Setup Cost*), and it has been recently included in the CSPlib library (Gent and Walsh, 1999, Problem 058).

We propose a metaheuristic approach for the MI-DLS-CC-SC problem that uses a neighborhood structure based on the *generalized pairwise interchange* (Della Croce, 1995) driven by a Simulated Annealing (SA) metaheuristic. Upon this search method, we also developed a statistically-principled tuning procedure. Due to the scarcity of available instances, we also devised a parametrized instance generator for the problem. Using our generator, we created realistic instances with diverse feature combinations, in terms of items, periods, and production levels.

In order to assess the quality of our solver, we compare it with the state-of-the-art search methods, which are, to the best of our knowledge, the three MILP models by Pochet and Wolsey (2006). In order to obtain up-to-date results and for the purpose of a fair comparison, we reimplemented the three models in CPLEX (CPLEX, 2016, v. 12.7.1), starting from their original source code implemented in Xpress-Mosel, and ran them on the same software environment of our SA solver. We tested our CPLEX implementation on large instances and the outcome is that, using the most effective of the three models of Pochet and Wolsey (2006) (i.e., milp3), it is able to find the optimal solution for a few instances, and a good one for a few others. On the other hand, it is far from the optimal value for many of them. In particular, it is not able to find even a feasible solution for any of the largest instances (500 periods). Our SA solver is somewhat complementary, as it scales nicely and it finds a nearly optimal solution for all instances, but without providing any guarantee about the optimality.

With the aim of merging the advantages of both methods, we propose a solution technique that runs SA for shorter time, and then switches to milp3 for the time left, using the best solution found by SA as initial one (warm start). This hybrid SA/MIP technique, properly tuned, turned out to outperform both pure methods and thus provide a good trade-off between them.

All instances employed in the experimentation and their best solutions are available for verification and future comparisons at our website `https://opthub.uniud.it`, together with the online solution validator, that provides against possible incorrect claims.

The paper is organized as follows. In Section 2, we introduce the MI-DLS-CC-SC problem and show the corresponding MILP models by Pochet

and Wolsey (2006). In Section 3, we summarize the related work in lot-sizing problems. In Section 4, we describe our SA search method and the hybrid one. In Section 5, we show the experimental analysis and its results. Finally, in Section 6, we draw the conclusions and discuss future work.

2. Discrete Lot-Sizing and Scheduling

In this section we illustrate the version of the LSS problem considered in the present work. First, we give a description of the problem (Section 2.1), and afterwards we provide the precise mathematical modeling (Section 2.2).

2.1. Problem description

The problem consists of the following entities.

- A set $I = \{0, \ldots, m-1\}$ of m items that represent the goods that must be produced.
- A set $P = \{1, \ldots, n\}$ of n periods in which it is possible to produce (i.e., the planning horizon). We start counting from 1 as period 0 customarily represents the last period of the previous scheduling phase. We call P' the set $\{0, 1, \ldots, n\}$, that is $P' = \{0\} \cup P$.
- A single machine that, in any period can either produce one piece of an item, or remain idle.
- An integer-valued $m \times n$ matrix D such that $d_t^i \geq 0$ represents the number of pieces of item i requested at time t.
- An integer-valued $m \times m$ matrix C such that $c^{ij} \geq 0$ represents the cost in setting up the machine from producing item i to producing item j. We assume no setup cost for subsequent productions of the same item, i.e., $c^{ii} = 0$ for all i.
- An integer-valued vector H of size m such that $h^i > 0$ represents the cost for stocking one piece of item i for one period.

The problem consists in determining the item produced by the machine in any period of P. A solution is thus a vector V of size n, whose elements are values in the set $\{-1, 0, \ldots, m-1\}$. The value v_t, with $t \in P$, represents the item produced at time t, with the special value $v_t = -1$ representing the absence of production (i.e., t is an idle period).

For each $i \in I$, the number of periods t such that $v_t = i$ must be equal to $\sum_{t \in P} d_t^i$. That is, the total number of pieces produced of an item must equal its total demand.

The problem includes also the following hard constraint:

- NoBacklog: Each piece must be produced not later than when it is requested. This means that for any item i, at any time $t \in P$ the number of pieces produced until t must be greater or equal to the requests of i up to t.

The problem asks to find a solution that is feasible with respect to the NoBacklog constraint, and that minimizes the setup and stocking costs. The objective function is thus composed by the following two components (soft constraints):

- StockingCost: Sum of the stocking periods of all pieces of all items multiplied by the stocking cost h^i of each item i.
- SetupCost: Sum of setup costs for all periods $t \in P$. It is assumed that no setup cost is added for the first produced piece. The problem enforces the *conservation of setup* rule (Fleischmann and Meyr, 1997), meaning that the setups state is carried over if there are idle periods and there is not setup cost if we continue to produce the same type of product as in the production period just before the idle periods.

Figure 1 shows a file that contains a small instance, written in the file format that we use for our instances, which adheres to the MiniZinc data file format (Nethercote et al., 2007).

The vector $V = \langle -1, 1, 1, 1, 2, 0, -1, 0 \rangle$ is an optimal solution of this instance. As shown in Figure 2, we see that no backlog occurs, its setup cost is $c^{1,2} + c^{2,0} = 175 + 101 = 276$, and its stocking cost is $h^1 + h^1 + 3h^1 + 2h^2 = 15 + 15 + 3 \cdot 15 + 2 \cdot 12 = 99$. The total cost is then $276 + 99 = 375$.

2.2. MILP models

For the sake of self-containedness, in the following we show the three MILP models proposed by Pochet and Wolsey (2006, §14.4), that we use for comparison with our solver. We refer to the cited book for the explanation of the models and a discussion about the intuitions underlying them.

```
Periods = 8;
Items = 3;
Demands = [|0, 0, 0, 0, 0, 1, 0, 1
           |0, 0, 1, 1, 0, 0, 1, 0
           |0, 0, 0, 0, 0, 0, 1, 0|];
StockingCosts = [10, 15, 12];
SetupCosts = [|0, 131, 109
              |193, 0, 175
              |101, 136, 0|];
```

Figure 1: A file containing a toy instance in MiniZinc format.

2.2.1. Model milp1.

The first model, that we call milp1, makes use of the following variables:

Variable	Range	Domain	Description
x_t^i	$t \in P$, $i \in I$	$\{0,1\}$	1 if item i is produced in period t, 0 otherwise
y_t^i	$t \in P'$, $i \in I$	$\{0,1\}$	1 if machine is ready for the production of item i in period t, 0 otherwise
s_t^i	$t \in P'$, $i \in I$	$\mathbb{R}_+$	number of pieces of item i on stock at the end of period t
χ_t^{ij}	$t \in P$, $i,j \in I$	$\mathbb{R}_+$	1 if in period t the production changes from item i to item j

The mathematical formulation of the problem is as follows:

$$\min \sum_{i=0}^{m-1} \sum_{t=1}^{n} h^i s_t^i + \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{t=1}^{n} c^{ij} \chi_t^{ij} \tag{1}$$

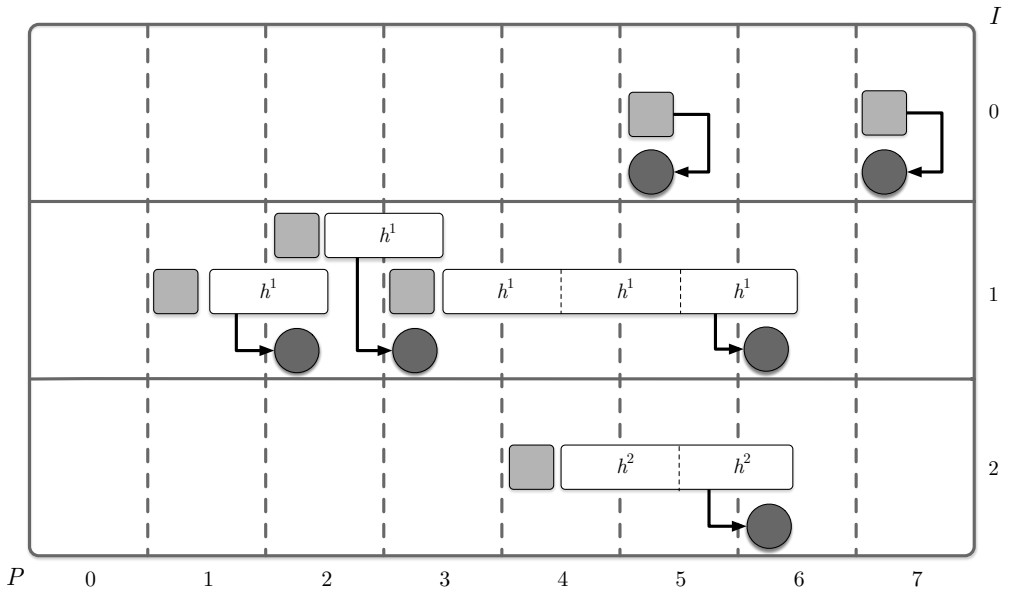

Figure 2: Visual representation of the instance in Figure 1 and its optimal solution. Circles represent the demands, whereas the item production is denoted by a gray square. Item stocking is shown as a white rectangle whose width is the number of stocking periods; the contribution of a given item to the stocking cost is represented within the rectangle.

$$s^i_{t-1} + x^i_t = d^i_t + s^i_t, \qquad \forall\ i \in I,\ t \in P \tag{2}$$

$$s^i_0 = 0, \qquad \forall\ i \in I \tag{3}$$

$$x^i_t \leq y^i_t, \qquad \forall\ i \in I,\ t \in P \tag{4}$$

$$\sum_{i=0}^{m-1} y^i_t = 1, \qquad \forall\ t \in P \tag{5}$$

$$\chi^{ij}_t \geq y^i_{t-1} + y^j_t - 1, \qquad \forall\ i, j \in I,\ t \in P \tag{6}$$

2.2.2. Model milp2.

The second model, milp2, uses the same variables as milp1, but augments it with the following ones:

Variable	Range	Domain	Description
z^i_t	$t \in P$, $i \in I$	$\{0,1\}$	1 if the machine is ready for item i in period t but not in period $t-1$
w^i_t	$t \in P'$, $i \in I$	$\{0,1\}$	1 if the machine is ready for item i in period t but not in period $t+1$

In order to obtain model milp2, Equation (6) of milp1 is replaced by Equations (7-10) below.

$$\sum_{i=0}^{m-1} \chi_t^{ij} = y_t^j, \qquad \forall\ j \in I,\ t \in P \tag{7}$$

$$\sum_{j=0}^{m-1} \chi_t^{ij} = y_{t-1}^i, \qquad \forall\ i \in I,\ t \in P \tag{8}$$

$$\sum_{i=0}^{m-1} y_0^i = 1, \tag{9}$$

$$y_t^j - z_t^j = y_{t-1}^j - w_{t-1}^j = \chi_t^{jj} \qquad \forall\ j \in I,\ t \in P \tag{10}$$

2.2.3. Model milp3.

For model milp3 it is assumed, without loss of generality (Miller and Wolsey, 2003, Observation 1), that $d_t^i \in \{0, 1\}$ for all periods t and items i.

Model milp3 is the same as milp2, with the addition of a set of valid inequalities. To express them, we need to introduce the following definition $d_{tl}^i = \sum_{k=t}^{l} d_k^i$. For all items i and for all intervals $[t, l] \subseteq [1, n]$ with $d_l^i = 1$, we add the following inequalities, in which, for the purpose of readability, like (Pochet and Wolsey, 2006, §10.5), we write d_{tl}^i as p.

$$s_{t-1}^i + \sum_{u=t}^{t+p-1} y_u^i + \sum_{u=t+1}^{t+p-1} [d_{ul}^i - (t+p-u)] z_u^i + \sum_{u=t+p}^{l} d_{ul}^i z_u^i \geq p \qquad \forall\ i \in I,\ t, l \in P \tag{11}$$

3. Related Work

The literature on lot-sizing distinguishes between discrete and continuous formulations, normally called Discrete Lot-Sizing and Scheduling Problem (DLSP) and Capacitated Lot-Sizing Problem (CLP), respectively. Both consider a finite and discrete planning period, time-varying dynamic demand and production capacity limited per period. However, while in DLSP the quantity produced in each period is either zero or one (*all or nothing production*), CLP allows for any production quantity, as far as it does not exceed the production capacity.

The differences between DLSP, CLP, and other similar problems (i.e. Economic Lot Sizing, Job Scheduling) are discussed by Salomon et al. (1991). The authors also introduce a problem classification taking into account the layout of the production line, the number of machines, the number of items, the production cost structure, and the setup time and cost structures.

Given that our problem belongs to the DLSP category, we focus our literature review on discrete formulations, and refer the reader to Drexl and Kimms (1997); Jans and Degraeve (2007); Copil et al. (2017) for a comprehensive coverage and to de Araujo et al. (2015); Fragkos et al. (2016) for recent developments on the continuous ones.

To the best of our knowledge, most of the work on DLSP deals with the single-machine, single-level variant of the problem, therefore, unless stated explicitly, we assume these specifications in the following. The multi-item DLSP problem has been firstly proposed by Fleischmann (1990). The author solves the problem using a branch-and-bound procedure based on Lagrangian relaxation to determine both lower bounds and feasible solutions, while the solution of the relaxed problem is obtained by a dynamic programming method.

An Integer Programming model for the single-item variant of the problem has been proposed by Van Hoesel et al. (1994). The model has been approached with three alternative solution methods: the first one derives some inequalities studying the structure of the DLSP polyhedron, the second one reformulates it as an assignment problem, and the last one uses dynamic programming. Problem reduction has been also employed by Salomon et al. (1997), who develop an exact solution procedure for the multi-item DLSP with sequence-dependent setup costs and setup times, based on a transformation into a Traveling Salesman Problem with Time Windows.

Miller and Wolsey (2003) formulate the multi-item DLSP with sequence-independent setup costs as a network flow problem, and present tight MIP formulations for different extensions (with backlogging, with safety stocks, with initial stock). Several variants and MIP formulations of the DLSP have been proposed and extensively discussed also by Pochet and Wolsey (2006). In particular, Gicquel et al. (2009b) present a tight formulation and derive some valid inequalities for the multi-item DLSP with sequence-dependent setup costs and times, which is an extension of the formulation proposed by Wolsey (2002). Gicquel et al. (2009a) propose a new way to model the multi-item DLSP with sequence-dependent setup costs that exploits the knowledge about some relevant physical attributes of items, such as color, dimension,

and quality level. This allows them to look at the setup costs at an aggregate level, and consequently to reduce significantly the number of related variables and constraints in the MIP model.

Moving to a different modeling paradigm, Houndji et al. (2014) introduce a new global constraint, that they call `StockingCost`, to efficiently handle the MI-DLS-CC-SC problem with Constraint Programming (CP). The authors test it on new generated benchmarks, and the experimental results show that `StockingCost` is more effective in filtering with respect to other decomposition techniques commonly used in the CP community.

Besides these works on the fundamental versions of the problem, other researchers extended the problem including constraints on the cost structures or related to the supply-chain organization. A variant of the single-item version of the problem with costs satisfying the property of *absence of speculative motive for early production* has been studied by van Eijl and van Hoesel (1997), who give a linear description of the convex hull of feasible solutions.

Moving to the supply-chain related extensions, lot availability and temporal features of orders are considered as additional constraints. In particular, Brüggemann and Jahnke (2000) extend the multi-item DLSP model with sequence-independent setup times to treat the case of batch availability, i.e., items produced become available to satisfy demand only after the whole lot has been completed. They develop a two-phase Simulated Annealing approach that first searches for a feasible production schedule and then tries to optimize it while preserving feasibility.

Supithak et al. (2010) extend the classical multi-item DLSP by introducing the concept of *order*, which has its own due date, earliness and tardiness penalties. As a consequence, backlogging is allowed and a tardiness cost is added to the objective function. They consider different variants of increasing complexity (without setup costs, with sequence-independent setup cost, with sequence-dependent setup cost) and tackle them with several approaches, ranging from an exact assignment algorithm to *ad hoc* heuristic procedures, and to a genetic metaheuristic.

4. Search Method

Our search method is based on local search, and in particular is driven by a Simulated Annealing metaheuristic. Unlike Brüggemann and Jahnke (2000), we implement a single-stage approach that is in charge of both the

solution feasibility and the cost optimization at the same time. In order to illustrate the solution method, we introduce the search space, the initial solution procedure, the cost function, the neighborhood relation, and finally the SA procedure that guides the local search.

4.1. Search space, initial solution, and cost function

As search space we use a direct representation of a solution, as explained in Section 2.1. That is, each state in the space is a vector V of size n, whose elements are values in the set $\{-1, 0, \ldots, m-1\}$. The value v_t represents the item produced at time t (indexes start from 0), with the value $v_t = -1$ representing the absence of production, so that t is an idle period.

The initial state is generated at random, starting from a vector V with all values equal to -1, and placing iteratively each requested production in a random idle period of vector V in a uniform way. This way all items are produced in the correct quantity, but it is possible that an item is produced later than when it is requested, thus NoBacklog violations are possible.

Hard constraints violations are therefore included in the cost function but considerably penalized. In detail, each period of lateness of one production is counted as one violation and their sum is added to the cost function, multiplied by a high weight (set to 1000 in the experiments). The cost function is then composed of three components: StockingCost, SetupCost, and NoBacklog.

4.2. Neighborhood relation

As neighborhood relation, we consider the set union of two basic ones:

Swap: Take two distinct elements of V and swap their values. A move belonging to the Swap neighborhood is thus represented by a pair $\langle t_1, t_2 \rangle$, with $t_1, t_2 \in P$, such that t_1 and t_2 are the positions of the two items. The corresponding move is denoted by $\mathsf{Swap}(t_1, t_2)$

Insert: Take an element of V and move it, either backward or forward, to a new position. A move belonging to the Insert neighborhood is represented by a pair $\langle t_1, t_2 \rangle$, with $t_1, t_2 \in P$, such that t_1 is the current position of the element and t_2 is the new one. The corresponding move is denoted by $\mathsf{Insert}(t_1, t_2)$

Up to our knowledge, the composite neighborhood Swap $\cup$ Insert has been firstly proposed by Della Croce (1995) with the name *generalized pairwise*

interchanges (GPI). In order to apply this neighborhood to our problem, we need to further refine it. In fact, due to the specific situation that the same production is repeated several times, the neighborhood includes moves that do not change the state to which they are applied (*null moves*) and pairs of moves that lead to the same state (*duplicate moves*).

For example, the move $\mathsf{Swap}(t_1, t_2)$ is null in any state in which $v_{t_1} = v_{t_2}$. In addition, the moves $\mathsf{Swap}(t_1, t_2)$ and $\mathsf{Insert}(t_1, t_2)$ are always duplicates when $|t_1 - t_2| = 1$. Furthermore, given the state $V = \langle -1, 1, 1, 1, 2, 0, -1, 0\rangle$, the move Insert(1,3) would lead to V itself, due to the sequence of identical values in positions 1-3 (remind that indexes start from 0). Similarly, the three moves Insert(1,7), Insert(2,7), and Insert(3,7) would lead to the same state $V' = \langle -1, 1, 1, 2, 0, -1, 0, 1\rangle$.

Therefore, in order to apply $\mathsf{Swap} \cup \mathsf{Insert}$ effectively, we need to detect all these situations and exclude null moves and copies of the duplicate ones from the exploration.

The cost difference of a move (also called the *delta function*) is computed in constant time for the SetupCost component, for both move types Swap and Insert. Conversely, StockingCost and NoBacklog costs are computed in $O(n)$ time (where n is the number of periods). In detail, for an $\mathsf{Insert}(t_1, t_2)$ move, the computational cost is linear with respect to $|t_1 - t_2|$, as the move affects the costs of all productions included in the interval $[t_1, t_2]$. For a $\mathsf{Swap}(t_1, t_2)$ move, it is linear in the number of productions of the two items involved in the move in the interval $[t_1, t_2]$, as the swap changes the specific demand that they satisfy, whereas all the other items are not affected.

4.3. Simulated Annealing

Our SA procedure is detailed in Algorithm 1 and is the instantiation of a generic Simulated Annealing metaheuristic parametrized by the local search modeling of the problem (line 1), i.e., the specific search space S, cost function F and the two neighborhoods $\mathcal{N}_{insert}$ and $\mathcal{N}_{swap}$ as described in the previous section.

Starting from a randomly constructed initial state s (line 3), the procedure draws at each iteration a random move. To this regard it proceeds in two stages: first it selects the neighborhood (Swap or Insert on line 9), and then the specific move in the neighborhood (lines 10, 12). The latter selection is made according to a uniform distribution, whereas the former is biased on the basis of a parameter β (called *insert rate*), so that Insert is selected with probability β, and Swap with probability $1 - \beta$. The bias β is

Algorithm 1 The pseudocode of the Iteration-based Simulated Annealing with cut-off approach employed in this work.

```
1: template ⟨S, F, N_insert, N_swap⟩
2: function SimulatedAnnealing(T_0, α, n_s, n_a, β, I_max)
3:     s ← RandomState(S)                          ▷ Initial random state
4:     T ← T_0                                     ▷ Initial temperature
5:     sampled ← 0
6:     accepted ← 0
7:     i ← 0
8:     while i < I_max do                          ▷ Iteration based stop criterion
9:         if U(0,1) < β then                      ▷ Biased move type choice
10:            m ← RandomMove(N_insert(s))         ▷ Insert move choice
11:        else
12:            m ← RandomMove(N_swap(s))           ▷ Swap move choice
13:        sampled ← sampled + 1
14:        Δ ← F(s ⊕ m) − F(s)                     ▷ Cost Improvement due to m
15:        if Δ ≤ 0 or e^{−Δ/T} > U(0,1) then
16:            s ← s ⊕ m
17:            accepted ← accepted + 1
18:        if sampled > n_s or accepted > n_a then ▷ Cut-off
19:            T ← α · T
20:            sampled ← 0
21:            accepted ← 0
22:        i ← i + 1
23:    return s
```

fixed experimentally, according to the statistical tuning procedure that will be detailed in Section 5.2.

As customary for SA, the move is always accepted if it is improving or sideways (i.e., same cost), whereas it is accepted based on time-decreasing exponential distribution in case it is worsening (line 15).

In detail, a worsening move is accepted with probability $e^{-\Delta/T}$, where Δ is the difference of total cost induced by the move, and T is the *temperature*. The temperature starts at value T_0 (line 4), and it is decreased during the search (line 19) by multiplying it by a value α (with $0 < \alpha < 1$) after a fixed number of samples n_s has been drawn. The temperature evolves according to the standard geometric cooling scheme of SA.

In order to speed up the early stages of the SA procedure, we use the *cut-off* mechanism (Johnson et al., 1989). To this aim we add a new parameter n_a, representing the maximum number of accepted moves at each temperature. That is, the temperature is decreased when the first of the following two conditions occurs (line 18): (i) the number of sampled moves reaches n_s, (ii) the number of accepted moves reaches n_a. This allows us to save computational time in early stages, which could be exploited later during the search.

We decided to use as stop criterion (line 8) the total number of iterations I_{max}. This choice has the advantage with respect to a threshold temperature that the running time is the same for all configurations. With respect to a fixed time limit, it has the advantage that it is not dependent on the experimental environment and it is deterministic (given the random seed), so that each run can be reproduced exactly. For our experimental analysis, the total number of iterations has been set to $I_{max} = 3 \cdot 10^8$, corresponding to a running time of about 300s for the largest instances.

The running time is equal for all configurations on the same instance, but it varies from instance to instance, as the computation of the costs is dependent on the size of the instance. We computed the function relating the running time t to the size of the instance using a linear regression on the experimental data. The resulting time t is given by the formula $t = 0.39n + 121.14$ (seconds), where n is the number of periods (it is independent from the number of items m).

5. Experimental Analysis

All code is written in standard C++11 and compiled using `gcc` (v. 5.4.0); the MILP models are implemented using the C++ API provided by the CPLEX/Concert technology. All experiments ran on an Ubuntu Linux 16.04 machine with 16 Intel® Xeon E5-2660 (2.20 GHz) cores and 32GB of RAM memory. A single core has been dedicated to each experiment. More precisely, for SA we run 16 experiments in parallel one for each core, whereas for CPLEX, we configured the process to use only one core, but, since this solution method is very memory-intensive, we run just one experiment at the time, in order to allow it to use all the available memory.

5.1. Benchmarks

For the MI-DLS-CC-SC problem, the only publicly-available instances are the `pigment` ones proposed by Houndji et al. (2014) (10 instances, available at `http://becool.info.ucl.ac.be/resources/discrete-lot-sizing-problem`) and the `PSP` ones, recently included in CSPlib (12 instances, available at `http://www.csplib.org/Problems/prob058/data/`).

The `pigment` instances turned out to be relatively simple to solve, and our method has been able to reach the optimal value in all runs. For them therefore the challenge is on the time to optimality, rather than on the value of the objective function. The `PSP` instances instead resulted to be more challenging. However, in order to have a larger number of instances to test our search method on the ground of solution quality, we decided to develop a parametrized generator. Our generator receives as input the number of items m, the number of periods n, and the density of requests δ, where δ is defined as $(\sum_{i=0}^{m-1} \sum_{t=1}^{n} d_t^i)/n$, and it produces a random instance with those features.

The generator works in such a way that the instance produced is feasible, in the sense that it is always possible to generate a schedule that satisfies the NoBacklog constraints. In addition, in order to use also model milp3, we generate only 0/1 requests, even though our SA solver accepts any positive integer request. We consider the values 10, 20, 30 for m, the values 200, 300, 400, 500 for n, and the values 0.8, 0.9, 1.0 for δ. We thus have $3 \times 4 \times 3 = 36$ combinations and we generated 6 instances for each combination. We used 180 instances (5 for each combination) as *training instances*, on which we performed the tuning procedure, and the remaining 36 as *validation instances*. We added the 12 CSPlib instances to the validation pool, so that

Parameter	Description	Value
T_0	Start temperature	37.0
α	Cooling rate	0.99
n_s	Moves sampled at each temperature	1204819
n_a	Moves accepted at each temperature	60240
β	Insert neighborhood rate	0.30
I_{max}	Total number of iterations	$3 \cdot 10^8$

Table 1: Parameter configuration for Simulated Annealing.

it comprises 48 instances altogether. Validation instances and their best solutions are available in a problem repository at `https://opthub.uniud.it`.

We have generated also instances with $n \geq 600$, but most of them turned out to be out of reach for the MIP methods even with long time limits, and therefore we decided not to consider them here and exclude them from the comparison.

5.2. Parameter tuning

As displayed in Algorithm 1 on page 12, the SA method has some parameters that need to be properly tuned, which are reported in the first two columns of Table 1 together with their descriptions. The tuning procedure is executed exclusively on the training instances, whereas the results are reported on the validation ones. This procedure provides against the risk of producing results affected by overtuning on the specific instances.

The tuning phase has been performed using the tool JSON2RUN (Urli, 2013), which samples the configurations using the *Hammersley point set* (Hammersley and Handscomb, 1964) and implements the F-Race procedure (Birattari et al., 2010) for comparing them. We set the confidence level of the Friedman rank-sum test employed in the F-Race procedure to 98% (i.e., p-value ≤ 0.02). As already mentioned, the total number of iterations has been set to $I_{max} = 3 \cdot 10^8$, corresponding to a running time of about 300s at the most.

The winning configuration turned out to be the one summarized in the last column of Table 1. The experiments in Section 5.4 and 5.5 have been performed using these values for the parameters.

instance	milp1	milp2	milp3	CP	SA	
pigment15b	—	33.42	0.42	12	0.20	(0.03)
pigment15c	—	36.06	0.40	16	0.20	(0.06)
pigment30a	110.51	1.12	0.04	124	0.25	(0.04)
pigment30b	—	47.37	0.60	244	0.25	(0.04)
pigment30c	†	53.88	0.07	156	0.24	(0.03)
pigment100a	—	90.12	0.51	60	0.46	(0.16)
pigment100b	†	41.87	0.36	10	0.39	(0.17)
pigment100c	—	273.92	6.82	143	0.40	(0.23)
pigment200a	—	1123.44	3.45	854	0.55	(0.38)

Table 2: Comparison of running times (seconds) on CSPlib instances

5.3. Comparison on CSPlib instances

The CSPlib instances turned out to be rather simple and the optimal value is reached in any run of SA.[1] Therefore we compare the results on these instances in terms on running times only.

For this experiment, we do not entirely use the parameter settings of Table 1, and in particular we reduce the total number of iterations I_{max} to $5 \cdot 10^5$ which is the minimum value such that for each instance at least 95% of the SA runs still produces the optimal value.

In Table 2, we compare the running times of our SA with the constraint-based method by Houndji et al. (2014) (as reported in the paper), denoted by CP in the table, and the MILP models by Pochet and Wolsey (2006) (our implementation). For SA, in parentheses, we report also the time (in seconds) at which the optimal value is reached for the first time. For MILP, a dash means that the optimal solution is not reached within a time limit of 1 hour and the *dag* symbol that the optimal value is reached within the time limit but not proven optimal.

The outcome is that our method is faster than the CP method of Houndji et al. (2014) and to milp1 and milp2. In comparison to milp3, our SA is slightly faster but basically at the same level, but, on the other side, it misses the proof of optimality. What emerged from this initial experiment is that more challenging instances are needed to draw any ultimate conclusion, therefore we proceeded with further analyses.

[1]Unfortunately, instance `pigment15a` turned out to be ill-formed as it has 8 items but a 10×10 setup matrix (we removed it from our experimental analysis).

5.4. Comparison on new instances

We ran an experiment that compares the three milp models among themselves on the 48 validation instances. The results (not reported here for brevity) have been that milp3 clearly outperforms the other two models in these instances as well. For this reason, in the following we compare our SA procedure with the milp3 model only.

Table 3 shows average and best results and relative standard deviation (RSD), i.e., the ratio between the standard deviation and the average, of our SA solver out of 30 runs in comparison with the results of milp3. In order to create a common ground, the milp3 solver is allowed to use only one processor and it is given the same time limit of SA, based on the linear function computed above, $t = 0.39n + 121.14$ (seconds).

A dash symbol means that milp3 has not found a feasible solution in the given time limit, whereas the star superscript means that the solution is proven optimal. The best between the two solvers is highlighted in boldface.

For a better assessment of the objective quality of the solutions, we report also the best solutions and the lower bounds found by CPLEX with milp3 using all 16 cores and running for 3 hours. The equal sign means that the LB coincides with the best solution.

Counting the boldface values, we see that SA gives a better average result in 37 out of 48 instances. In the remaining 11 cases, SA is worse than milp3 by at most 0.38%, in particular on instance `ps-300-10-80` with an average of 26477.24, where the IP solution costs is 26376 (and 26477.24/26376 = 1.0038).

In addition, the difference with respect to the LB is at most 1.94% (on instance `ps-500-20-100`). It is worth noticing that such difference does not grow with the number of items n and the number of items m, showing that SA does not degrade performances when increasing the size. Looking at the other robustness measures, it is clear that SA results are very stable, displaying a relative standard deviation of at most 0.59% (on instance `PSP_200_4`).

5.5. Hybrid method

The two methods discussed in this work, namely MILP and SA, are rather complementary, as one is exact and deterministic, whilst the other is heuristic and stochastic. As a consequence, it is thus impossible to compare them in a totally fair way. In fact, our results confirm that, as expected, there is no clear winner, and they both have advantages and disadvantages. Specifically, the MILP models have the advantage of being able to find lower bounds and

Instance	SA			milp3	benchmark	
	avg	RSD	(best)	cost	LB	cost
PSP_100_1	10095.73	(0.14%)	10088	**10088***	=	10088
PSP_100_2	10378.60	(0.26%)	10347	**10359**	=	10347
PSP_100_3	**10342.03**	(0.08%)	10340	10412	=	10340
PSP_100_4	**9007.53**	(0.11%)	8999	9046	=	8999
PSP_150_1	**18043.80**	(0.20%)	17997	20864	=	17997
PSP_150_2	**25727.43**	(0.16%)	25656	—	25512.3	25890
PSP_150_3	**14488.67**	(0.17%)	14457	14909	=	14457
PSP_150_4	**18277.03**	(0.28%)	18171	38295	18123.8	18189
PSP_200_1	**22044.20**	(0.27%)	21951	22596	=	21882
PSP_200_2	16177.83	(0.17%)	16128	**16127**	=	16127
PSP_200_3	18349.53	(0.19%)	18289	**18289***	=	18289
PSP_200_4	**20983.97**	(0.59%)	20741	48989	=	20724
ps-200-10-80	18139.32	(0.22%)	18090	**18089***	=	18089
ps-200-10-90	20294.92	(0.23%)	20236	**20236***	=	20236
ps-200-10-100	**30937.48**	(0.23%)	30812	—	30648	30771
ps-200-20-80	19220.80	(0.17%)	19190	**19190***	=	19190
ps-200-20-90	**23976.14**	(0.19%)	23916	23991	=	23916
ps-200-20-100	26173.40	(0.14%)	26103	**26112**	=	26103
ps-200-30-80	**20544.68**	(0.12%)	20518	20547	=	20518
ps-200-30-90	25850.18	(0.29%)	25754	**25756**	=	25754
ps-200-30-100	**30964.76**	(0.23%)	30847	—	30703.4	30869
ps-300-10-80	26477.24	(0.24%)	26349	**26376**	=	26343
ps-300-10-90	**30654.46**	(0.14%)	30578	37159	=	30567
ps-300-10-100	**50640.66**	(0.18%)	50470	—	49742.7	51238
ps-300-20-80	**30282.38**	(0.15%)	30214	30338	=	30206
ps-300-20-90	**37743.14**	(0.20%)	37641	—	37540.4	37629
ps-300-20-100	**47680.04**	(0.18%)	47504	—	46965.7	47474
ps-300-30-80	33292.68	(0.18%)	33194	**33224**	=	33183
ps-300-30-90	**40281.44**	(0.29%)	40042	—	39915.9	40074
ps-300-30-100	**45752.58**	(0.19%)	45587	—	45273.3	45498
ps-400-10-80	**34345.04**	(0.26%)	34168	—	=	34136
ps-400-10-90	**39026.70**	(0.20%)	38896	—	=	38854
ps-400-10-100	**57780.64**	(0.21%)	57527	—	56914.6	58194
ps-400-20-80	**41457.78**	(0.19%)	41309	—	41226.7	41283
ps-400-20-90	**55095.68**	(0.33%)	54825	—	54508.2	54971
ps-400-20-100	**65300.40**	(0.16%)	65016	—	64386.3	65529
ps-400-30-80	**43084.04**	(0.13%)	42989	45323	=	42959
ps-400-30-90	**50741.74**	(0.17%)	50596	—	50521.8	50558
ps-400-30-100	**67172.34**	(0.23%)	66865	—	66100.1	69071
ps-500-10-80	**45413.92**	(0.24%)	45182	—	44889.7	45064
ps-500-10-90	**54686.44**	(0.20%)	54401	—	53932.4	54390
ps-500-10-100	**105078.16**	(0.17%)	104741	—	103610	166935
ps-500-20-80	**51783.80**	(0.18%)	51616	—	51547.8	51553
ps-500-20-90	**59285.82**	(0.19%)	59040	—	58707.7	59090
ps-500-20-100	**75381.56**	(0.22%)	75075	—	73941.3	113016
ps-500-30-80	**53944.52**	(0.14%)	53786	—	53670.5	53714
ps-500-30-90	**64776.56**	(0.14%)	64566	—	64099.5	64759
ps-500-30-100	**97718.46**	(0.23%)	97248	—	95983.9	—

Table 3: Comparison of results between milp3 and SA

certified optimal solutions. Our SA solver, on the other side, scales nicely and is very robust, as it always finds a near-optimal solution, without any risk of getting stuck in low quality solutions for long time.

In order to merge the advantages of the two methods, we propose a plain combination SA+milp3 that runs SA for a shorter time, and then switches to milp3 using the solution found by SA as initial one. Indeed, CPLEX allows the option, called *warm start* (or *advanced start*), to pass to the solver a set of assignments to be used as initial solution, either partial or complete (it is complete in our case).

The share of time used by each solver has been subject of tuning on the training instances. In detail, we tuned the fraction of time ρ given to SA, by using 11 candidate configurations: $\rho = [0.0, 0.1, ..., 0.9, 1.0]$. This way, the two "pure" methods are also included in the candidate's pool.

In addition, due to the presence of a good initial solution, we tested the possibility offered by CPLEX to use the Relaxation Induced Neighborhood Search (RINS) (Danna et al., 2005), as heuristic during the search. We tuned this option by using 7 configurations of the corresponding parameter, namely $k = [-1, 0, 5, 10, 20, 40, 80]$. A value $k > 0$ means that RINS heuristic is applied every k visited nodes. The special values -1 and 0 denote that the RINS heuristic is turned off and that the solver autonomously decided when to apply it, respectively.

We thus have $11 \times 7 = 77$ configurations, upon which we use again `json2run` and its underlying RACE procedure (with p-value = 0.02), to identify the best one. The winning configuration turned out to be the one with $\rho = 0.7$ and $k = 0$, proving also that SA+milp3 is statistically superior to `milp3` ($\rho = 0$) and SA ($\rho = 1$).

In order to make a fair evaluation for this strategy, we compare it with a hybrid strategy that combines milp3 with a different initial solution construction. We need a strategy that produces always a feasible solution, as an infeasible one would be discarded by milp3. To this aim, we design a *greedy* technique that assigns productions starting form the end of the planning period and selects the item at each stage t only among items that are requested at a later stage $t' \geq t$ and have not been already selected at any later stage $t'' > t$ (a idle period is assigned if no such item exists). This way, we always obtain an initial solution that has no NoBacklog violations, and therefore it is feasible. At each step the item selected is the one with the lowest cost in terms of stocking and setup with respect to the following one. We name this technique Gr+milp3.

Table 4 shows the average result of 50 runs of the two hybrids method on the validation instances, along with the ones of the pure methods of Table 3 (repeated here for clarity) and the corresponding percentage gaps.

We see that Gr+milp3 obviously finds a feasible solutions in the cases in which milp3 fails this task, but in these cases rarely produces a good final solution.

Regarding SA+milp3, we can see that the results are indeed very close to SA, with no clear winner. In particular, we see that up to size $n = 300$, the results of the hybrid method are slightly superior in average, whereas from $n = 400$ upwards the SA solver is still better. In any case, the hybrid method SA+milp3 reaps benefits of both the other two: there are no unsolved instances and in some cases the solution is proven optimal (and the solver runs slightly faster).

In order to obtain a clearer picture of the behavior of SA+milp3, we repeated the tuning and the comparison for a time limit ten times longer. The result is that the best share of time for SA, in this new setting, is 0.2 (instead of 0.7) and the best RINS frequency is not the default one ($k = 0$), but every 5 visited nodes.

Table 5 shows the average results of 10 runs of SA+milp3 on the validation instances, along with the ones of the pure methods and the corresponding percentage gaps. Here, the edge of the hybrid method is more evident, as it is better than SA in 35 instances (equal in 2, worse in 11), and better than `milp3` in 30 (equal in 17, worse in just 1).

6. Conclusions and Future Work

We have proposed a local search approach based on Simulated Annealing that uses a GPI neighborhood for a discrete lot-sizing problem.

We have shown experimentally that, even using such a relatively-simple neighborhood structure, the solver (efficiently implemented and properly tuned) is able to find a nearly-optimal solution on all instances, including the instance sizes for which the best MILP approaches are unable to find a reasonable solution in the same time.

In addition, we have shown a simple hybridization of the two methods, making also use of the RINS heuristic, that is able to combine the advantages of both of them.

Finally, we have proposed a new dataset that tries to fill the gap in comparability due to the shortage of challenging benchmarks. The new dataset

Instance	SA+milp3	Gr+milp3	SA	milp3	gap betweeen SA+milp3 and		
	avg	cost	avg	cost	Gr+milp3	SA	milp3
PSP_100_1	10091.22	10088	10095.73	10088	0.03	-0.04	0.03
PSP_100_2	10374.28	10370	10378.60	10359	0.04	-0.04	0.15
PSP_100_3	10340.00	10340	10342.03	10412	0.00	-0.02	-0.69
PSP_100_4	9011.44	9009	9007.53	9046	0.03	0.04	-0.38
PSP_150_1	18047.64	20736	18043.80	20864	-12.96	0.02	-13.50
PSP_150_2	25750.90	27568	25727.43	—	-6.59	0.09	—
PSP_150_3	14474.76	14617	14488.67	14909	-0.97	-0.10	-2.91
PSP_150_4	18291.72	21052	18277.03	38295	-13.11	0.08	-52.23
PSP_200_1	22052.92	22747	22044.20	22596	-3.05	0.04	-2.40
PSP_200_2	16143.92	16127	16177.83	16127	0.10	-0.21	0.10
PSP_200_3	18313.40	18311	18349.53	18289	0.01	-0.20	0.13
PSP_200_4	21016.86	24681	20983.97	48989	-14.85	0.16	-57.10
ps-200-10-80	18098.52	18089	18139.32	18089	0.05	-0.22	0.05
ps-200-10-90	20255.90	20290	20294.92	20236	-0.17	-0.19	0.10
ps-200-10-100	30949.42	35078	30937.48	—	-11.77	0.04	—
ps-200-20-80	19190.00	19190	19220.80	19190	0.00	-0.16	0.00
ps-200-20-90	23954.58	24099	23976.14	23991	-0.60	-0.09	-0.15
ps-200-20-100	26189.72	26119	26173.40	26112	0.27	0.06	0.30
ps-200-30-80	20533.28	20529	20544.68	20547	0.02	-0.06	-0.07
ps-200-30-90	25782.80	25754	25850.18	25756	0.11	-0.26	0.10
ps-200-30-100	31009.88	31682	30964.76	—	-2.12	0.15	—
ps-300-10-80	26388.42	26343	26477.24	26376	0.17	-0.34	0.05
ps-300-10-90	30674.36	30635	30654.46	37159	0.13	0.06	-17.45
ps-300-10-100	50715.12	57810	50640.66	—	-12.27	0.15	—
ps-300-20-80	30234.84	30334	30282.38	30338	-0.33	-0.16	-0.34
ps-300-20-90	37792.58	37762	37743.14	—	0.08	0.13	—
ps-300-20-100	47694.78	53875	47680.04	—	-11.47	0.03	—
ps-300-30-80	33238.10	33217	33292.68	33224	0.06	-0.16	0.04
ps-300-30-90	40296.32	45951	40281.44	—	-12.31	0.04	—
ps-300-30-100	45776.28	51367	45752.58	—	-10.88	0.05	—
ps-400-10-80	34348.84	34469	34345.04	—	-0.35	0.01	—
ps-400-10-90	39069.72	39347	39026.70	—	-0.70	0.11	—
ps-400-10-100	57892.10	65263	57780.64	—	-11.29	0.19	—
ps-400-20-80	41516.58	41813	41457.78	—	-0.71	0.14	—
ps-400-20-90	55155.98	62925	55095.68	—	-12.35	0.11	—
ps-400-20-100	65329.52	75839	65300.40	—	-13.86	0.04	—
ps-400-30-80	43116.70	43091	43084.04	45323	0.06	0.08	-4.87
ps-400-30-90	50787.70	58296	50741.74	—	-12.88	0.09	—
ps-400-30-100	67232.44	76204	67172.34	—	-11.77	0.09	—
ps-500-10-80	45476.94	52349	45413.92	—	-13.13	0.14	—
ps-500-10-90	54726.44	64760	54686.44	—	-15.49	0.07	—
ps-500-10-100	105128.82	127816	105078.16	—	-17.75	0.05	—
ps-500-20-80	51778.50	58384	51783.80	—	-11.31	-0.01	—
ps-500-20-90	59336.66	66858	59285.82	—	-11.25	0.09	—
ps-500-20-100	75491.96	85955	75381.56	—	-12.17	0.15	—
ps-500-30-80	53966.70	60269	53944.52	—	-10.46	0.04	—
ps-500-30-90	64806.68	72262	64776.56	—	-10.32	0.05	—
ps-500-30-100	97748.54	112690	97718.46	—	-13.26	0.03	—

Table 4: Comparison of results of the hybrid methods with milp3 and SA

Instance	SA+milp3 avg	SA avg	milp3 cost	SA+milp3 gap SA	SA+milp3 gap milp3
PSP_100_1	10088.0	10088.8	10088	-0.0079	0.0000
PSP_100_2	10350.0	10350.5	10347	-0.0048	0.0290
PSP_100_3	10340.0	10340.0	10340	0.0000	0.0000
PSP_100_4	8999.0	8999.0	8999	0.0000	0.0000
PSP_150_1	17997.0	18011.6	17997	-0.0811	0.0000
PSP_150_2	25693.0	25677.6	26013	0.0600	-1.2302
PSP_150_3	14458.5	14462.7	14466	-0.0290	-0.0518
PSP_150_4	18190.5	18189.1	18403	0.0077	-1.1547
PSP_200_1	21882.0	21941.4	21882	-0.2707	0.0000
PSP_200_2	16127.0	16170.1	16127	-0.2665	0.0000
PSP_200_3	18289.0	18317.7	18289	-0.1567	0.0000
PSP_200_4	20733.5	20800.6	20917	-0.3226	-0.8773
ps-200-10-80	18089.0	18100.0	18089	-0.0608	0.0000
ps-200-10-90	20236.0	20266.2	20236	-0.1490	0.0000
ps-200-10-100	30799.3	30821.2	31145	-0.0711	-1.1100
ps-200-20-80	19190.0	19199.2	19190	-0.0479	0.0000
ps-200-20-90	23916.0	23930.7	23916	-0.0614	0.0000
ps-200-20-100	26103.0	26118.8	26103	-0.0605	0.0000
ps-200-30-80	20518.0	20528.2	20518	-0.0497	0.0000
ps-200-30-90	25754.0	25775.3	25754	-0.0826	0.0000
ps-200-30-100	30834.9	30859.5	31305	-0.0797	-1.5017
ps-300-10-80	26343.0	26400.0	26343	-0.2159	0.0000
ps-300-10-90	30567.0	30602.9	30582	-0.1173	-0.0490
ps-300-10-100	50553.3	50483.6	—	0.1381	—
ps-300-20-80	30206.0	30235.4	30207	-0.0972	-0.0033
ps-300-20-90	37605.4	37658.5	37800	-0.1410	-0.5148
ps-300-20-100	47576.2	47529.4	—	0.0985	—
ps-300-30-80	33183.0	33231.9	33183	-0.1471	0.0000
ps-300-30-90	40110.7	40151.0	41468	-0.1004	-3.2731
ps-300-30-100	45528.0	45603.8	46149	-0.1662	-1.3456
ps-400-10-80	34138.2	34293.7	34140	-0.4534	-0.0053
ps-400-10-90	38854.0	38928.6	38854	-0.1916	0.0000
ps-400-10-100	57660.7	57593.6	—	0.1165	—
ps-400-20-80	41287.0	41364.5	41562	-0.1874	-0.6617
ps-400-20-90	54842.3	54888.0	55602	-0.0833	-1.3663
ps-400-20-100	65224.1	65029.1	—	0.2999	—
ps-400-30-80	42960.2	43010.1	42962	-0.1160	-0.0042
ps-400-30-90	50561.4	50681.9	50650	-0.2378	-0.1749
ps-400-30-100	67045.7	66859.7	—	0.2782	—
ps-500-10-80	45076.9	45233.1	46018	-0.3453	-2.0451
ps-500-10-90	54374.8	54491.1	55160	-0.2134	-1.4235
ps-500-10-100	104961.6	104799.4	—	0.1548	—
ps-500-20-80	51553.8	51642.4	51581	-0.1716	-0.0527
ps-500-20-90	59026.3	59135.5	—	-0.1847	—
ps-500-20-100	75287.8	75007.9	—	0.3732	—
ps-500-30-80	53718.4	53861.0	53787	-0.2648	-0.1275
ps-500-30-90	64581.6	64568.0	65456	0.0211	-1.3359
ps-500-30-100	97531.2	97194.0	—	0.3469	—

Table 5: Comparison of results of the hybrid method with milp3 and SA for long runs

comes with suitable web-based validation tools that prevent the possibility of publishing incorrect results, and hopefully will be used for comparison in future research.

For the future, we intend to extend our solver to other versions of the LSS problem. We also plan to design and experiment new neighborhood relations and new local search procedures in order to try to improve the current results. Furthermore, we will investigate a methodology of feature-based tuning, that could improve the tuning phase by relating the parameter values to the main features of the solvers. Finally, we aim to develop and explore other MIP-based heuristics, in order to compare them with our metaheuristic approach.

Acknowledgments

We thank Vinasetan Ratheil Houndji and Pierre Schaus for kindly answering all our questions about their work and Laurence Wolsey for providing us the source code of the models contained in his seminal book.

References

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-race and iterated F-race: An overview. In: Experimental methods for the analysis of optimization algorithms. Springer, Berlin, pp. 311–336.

Brüggemann, W., Jahnke, H., 2000. Discrete lot-sizing and scheduling problem: complexity and modification for batch availability. European Journal of Operational Research 124 (3), 511–528.

Copil, K., Wörbelauer, M., Meyr, H., Tempelmeier, H., 2017. Simultaneous lotsizing and scheduling problems: a classification and review of models. OR Spectrum 39 (1), 1–64.

CPLEX, 2016. CPLEX Optimizer. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`, v. 12.7.1.

Danna, E., Rothberg, E., Le Pape, C., 2005. Exploring relaxation induced neighborhoods to improve mip solutions. Mathematical Programming 102 (1), 71–90.

de Araujo, S. A., De Reyck, B., Degraeve, Z., Fragkos, I., Jans, R., 2015. Period decompositions for the capacitated lot sizing problem with setup times. INFORMS Journal on Computing 27 (3), 431–448.

Della Croce, F., 1995. Generalized pairwise interchanges and machine scheduling. European Journal of Operational Research 83 (2), 310–319.

Drexl, A., Kimms, A., 1997. Lot sizing and scheduling - survey and extensions. European Journal of Operational Research 99 (2), 221–235.

Fleischmann, B., 1990. The discrete lot-sizing and scheduling problem. European Journal of Operational Research 44 (3), 337–348.

Fleischmann, B., Meyr, H., 1997. The general lotsizing and scheduling problem. OR Spektrum 19 (1), 11–21.

Fragkos, I., Degraeve, Z., De Reyck, B., 2016. A horizon decomposition approach for the capacitated lot-sizing problem with setup times. INFORMS Journal on Computing 28 (3), 465–482.

Gent, I. P., Walsh, T., 1999. CSPLib: a benchmark library for constraints. In: Principles and Practice of Constraint Programming (CP'99). Springer, pp. 480–481, available from `http://www.csplib.org`.

Gicquel, C., Miègeville, N., Minoux, M., Dallery, Y., 2009a. Discrete lot sizing and scheduling using product decomposition into attributes. Computers & Operations Research 36 (9), 2690–2698.

Gicquel, C., Minoux, M., Dallery, Y., 2009b. On the discrete lot-sizing and scheduling problem with sequence-dependent changeover times. Operations Research Letters 37 (1), 32–36.

Hammersley, J. M., Handscomb, D. C., 1964. Monte Carlo methods. Chapman and Hall.

Houndji, V. R., Schaus, P., Wolsey, L., Deville, Y., 2014. The StockingCost constraint. In: Principles and Practice of Constraint Programming (CP 2014). pp. 382–397.

Jans, R., Degraeve, Z., 2007. Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. European Journal of Operational Research 177 (3), 1855–1875.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., Schevon, C., 1989. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. Operations Research 37 (6), 865–892.

Miller, A. J., Wolsey, L. A., 2003. Tight MIP formulation for multi-item discrete lot-sizing problems. Operations Research 51 (4), 557–565.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., Tack, G., 2007. MiniZinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming (CP 2007). pp. 529–543.

Pochet, Y., Wolsey, L. A., 2006. Production planning by mixed integer programming. Springer Science & Business Media.

Salomon, M., Kroon, L. G., Kuik, R., Van Wassenhove, L. N., 1991. Some extensions of the discrete lotsizing and scheduling problem. Management Science 37 (7), 801–812.

Salomon, M., Solomon, M. M., Van Wassenhove, L. N., Dumas, Y., Dauzère-Pérès, S., 1997. Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the travelling salesman problem with time windows. European Journal of Operational Research 100 (3), 494–513.

Supithak, W., Liman, S. D., Montes, E. J., 2010. Lot-sizing and scheduling problem with earliness tardiness and setup penalties. Computers and Industrial Engineering 58 (3), 363–372.

Urli, T., 2013. json2run: a tool for experiment design & analysis. CoRR abs/1305.1112.

van Eijl, C., van Hoesel, C., 1997. On the discrete lot-sizing and scheduling problem with wagner-whitin costs. Operations Research Letters 20 (1), 7–13.

Van Hoesel, S., Kuik, R., Salomon, M., Van Wassenhove, L. N., 1994. The single-item discrete lotsizing and scheduling problem: optimization by linear and dynamic programming. Discrete Applied Mathematics 48 (3), 289–303.

Wolsey, L. A., 2002. Solving multi-item lot-sizing problems with an MIP solver using classification and reformulation. Management Science 48 (12), 1587–1602.