Tabu Search for a Discrete Lot Sizing and Scheduling Problem

Filipe Carvalho[*] *Ana Sofia Pereira*[*] *Miguel Constantino*[*]
João Pedro Pedroso[*]

[*] Faculdade de Ciências, Universidade de Lisboa
Campo Grande, Lisboa - Portugal
Email: {filipe.carvalho, anapereira, miguel.constantino, jpp}@fc.ul.pt

1 Introduction

In this paper we consider a multi-item, multi-machine discrete lot sizing and scheduling problems (*DLSP*) which consists of finding a minimal cost production schedule over a finite discrete horizon, where in each period a machine may be either producing one item at full capacity or is idle. The demand for each item in each period is known, and can be satisfied either by production in the period, or from stock carried from the previous periods.

Here we allow the demand to be satisfied from backlogging from subsequent periods as well. Machine capacity (or the quantity produced) varies with the items and periods. We consider costs for production, storage, backlogging and changeover. Production costs are fixed for each combination of item-machine-period. Storage costs are proportional to the quantities left in stock at the end of each period. Backlogging costs are defined by a two step piecewise linear convex function, corresponding to the situation where a large amount of backlogging is highly penalized. Changeover costs are incurred when a machine produces some item (or is idle) in some period and is in a different situation in the subsequent period. They are sequence dependent i.e., for a particular machine and period, they depend on the order items are produced.

This problem is, in general, NP-hard. Even the single machine version of the problem, without backlogging, zero production costs and constant inventory and changeover costs is already NP-hard [1].

In this paper we start by introducing a MIP formulation for the *DLSP* variant considered. We then describe some heuristics for this problem. The heuristics consist of obtaining an initial solution, which may be feasible or not, and then use local search on some neighbourhoods for obtaining a refined solution. We describe several methods for obtaining initial solutions, as well as some neighbourhoods and different ways of exploring these neighbourhoods.

2 Representation of the solutions

The production plan is represented by an order $T \times K$ matrix x, where rows correspond to the periods, columns to the machines, and each element of the matrix holds the item being produced; hence $x_{t,k} = i$ means that product i is being produced on the machine k at period t. Let $x = [x_{t,k}]$ be a solution; the set of all the solutions (feasible and infeasible) considered for this problem is

$$\mathcal{X} = \left\{ x \in \{0, 1, \ldots, P\}^{T \times K} \right\}$$

2.1 Cost calculation

To each solution x there is associated a stock matrix and a backlog matrix. Let the stock for a solution x be a $T \times P$ matrix $s = [s_{t,i}]$, where each element $s_{t,i} \in \mathbb{R}_0^+$ represents the stock of item i in period t. Similarly, the backlog is a $T \times P$ matrix $r = [r_{t,i}]$, with $r_{t,i} \in \mathbb{R}_0^+$.

We define an indicator function

$$I(x_{t,k}, i) = \begin{cases} 1 & \text{if } x_{t,k} = i \\ 0 & \text{oherwise} \end{cases}$$

For all periods $t = 1, \ldots, T$, we determine a value

$$\Delta_i = s_{i,t-1} - r_{i,t-1} - d_{i,t} + \sum_{k=1}^{K} C_{i,k,t} I(x_{t,k}, i)$$

which is the inventory of product i if positive, or its backlog if negative. Therefore

$$s_{t,i} = \begin{cases} \Delta_i & \text{if } \Delta_i > 0 \\ 0 & \text{oherwise} \end{cases}$$

and

$$r_{t,i} = \begin{cases} 0 & \text{if } \Delta_i > 0 \\ -\Delta_i & \text{oherwise} \end{cases}$$

The stock cost is $\sum_{i=1}^{P} \sum_{t=1}^{T} s_{t,i} h_i$ and the sale revenue is $\sum_{i=1}^{P} s_{T,i} v_i$.

The backlog of a solution x is represented by the matrix $r(x) = [r_{t,i}]$, $r_{t,i} \in \mathbb{R}_0^+$. Its cost is given by $\sum_{i=1}^{P} \sum_{t=1}^{T} g_i(r_{t,i})$ where $g_i(r_{i,t})$ is determined by the following expression:

$$g(r_{i,t}) = \begin{cases} b_i^0 r_{i,t} & \text{if } r_{i,t} \leq R_i \\ b_i^0 r_{i,t} + b_i^1 (r_{i,t} - R_i) & \text{if } r_{i,t} > R_i \end{cases}$$

In our model there are changeover costs. These are start-up and switch-off costs when, respectively, the previous or next items to be produced in a given machine is 0 (i.e., the no-production item). The initial and final state of all the machines is no-production; taking this into consideration, we can calculate the changeover costs as:

$$\sum_{k=1}^{K} \left(u_{0,x_{1,k}} + \sum_{t=2}^{T} u_{x_{t-1,k}, x_{t,k}} + u_{x_{T,K},0} \right)$$

Finally, there are production costs associated with item batches. Whenever a machine produces an item, there is a fixed production cost, which only varies according to the item, f_i , $(i = 1, \ldots, P)$. So, the batches production cost is given by

$$\sum_{i=1}^{P} \sum_{t=1}^{T} \sum_{k=1}^{K} f_i I(x_{t,k}, i)$$

In conclusion, the cost of a solution x is:

$$\begin{aligned} c(x) &= \sum_{i=1}^{P} \sum_{t=1}^{T} (s_{t,i} h_i + g_i(r_{t,i})) - \sum_{i=1}^{P} s_{T,i} v_i \\ &+ \sum_{k=1}^{K} \left(u_{0,x_{1,k}} + \sum_{t=2}^{T} u_{x_{t-1,k}, x_{t,k}} + u_{x_{T,K},0} \right) \\ &+ \sum_{i=1}^{P} \sum_{t=1}^{T} \sum_{k=1}^{K} f_i I(x_{t,k}, i) \end{aligned}$$

2.2 Classification of the solutions

A solution is feasible if and only if the demand for all items is completely satisfied within the planning horizon, i.e, there is no backlog at the last period. Solution infeasibility is measured as the sum of the backlog variables at the last period, $\text{Inf}(x) = \sum_{i=1}^{P} r_{T,i}$. The set of feasible solutions $\mathcal{F}$ is defined as $\mathcal{F} = \{x =\in \mathcal{X} : \text{Inf}(x) = 0\}$

When comparing two solutions, we need to decide which one is preferable based on their objective value and on their deviation from feasibility. In the context of this paper, for two solutions $x, y \in \mathcal{X}$ we say that x *is better* than y ($x \prec y$) iff:

- $\text{Inf}(x) < \text{Inf}(y)$ or
- $\text{Inf}(x) = \text{Inf}(y)$ and $c(x) < c(y)$.

3 Construction heuristics

We present two heuristics that randomly construct a solution. They require a very short computational time, but generally fail to obtain a feasible solution. They are useful to generate initial solutions for local search methods; actually, as the quality of their solutions is rather poor, they are supposed to be provide good starting points to show the efficiency of the local search methods.

The information provided by the solution of the linear relaxation of a mixed integer programming model can be used in a construction heuristic to obtain an integer solution. An initial application of the idea that we are using was published in [4].

This heuristic takes the solution of the linear programming (LP) relaxation of the MIP problem and constructs an integer solution with the information obtained on the variables $x_{i,k,t}$, where $x_{i,k,t}$ is a binary variable which indicates whether an item i is produced in a machine k in period t ($x_{i,k,t} = 1$) or not ($x_{i,k,t} = 0$) (for $i = 0, \ldots, P$, $j = 0, \ldots, P$, $t = 1, \ldots, T$). Notice that the sum of the variables concerning a machine in a time period is constrained to 1, in the MIP formulation. Let us call $x^{\text{LP}}_{i,k,t}$ the optimal value of these variables on the LP relaxation of the MIP.

Therefore we will produce item i on machine k in time t with probability $x^{\text{LP}}_{i,k,t}$.

$Prob(x_{t,k} = i) = x^{\text{LP}}_{i,k,t}$, $i = 0, \ldots, P$, $k = 1, \ldots, K$, $t = 1, \ldots, T$

With this heuristic we build solutions that, although still generally infeasible, leads to better solutions than the previous methods when used as initial solutions in tabu search.

4 Neighbourhoods

The local search methods considered here use three types of neighbourhoods: *n-change*, *swap* and *insert*. The *n-change* neighbourhood takes n elements of the solution matrix and for each of them (independently) change the item being produced into another item.

$N^n_{change}(x) = \{\ y \in \mathcal{X} :$ y can be obtained from x by changing at most n elements $x_{t,k}$ from their current value into an element of the set $\{0, 1, \ldots, P\}\}$

The swap neighbourhood of a solution x, $N_{swap}(x)$, is composed of solutions which differ from x on two elements, whose values are interchanged.

$N_{swap}(x) = \{\ y \in \mathcal{X} : y$ can be obtained from x by interchanging the position of two of its elements$\}$

The insert neighbourhood of a solution x consists on solutions obtained by removing an element of a column of x and inserting it on another position of the same column, by shifting all the lines of that column that were between the two positions. Hence, the production plan is modified only on one machine, between the two periods concerned.

$N_{insert}(x) = \{\ y \in \mathcal{X} : y$ can be obtained from x by removing an element $x_{t_1,k}$, shifting all the elements next to t_1 up to another position on the same column t_2, and inserting $x_{t_1,k}$ on $t_2\}$.

The movement can be made upwards or downwards, which means that a cell can be moved to a preceeding or succeeding period.

5 Tabu Search

We propose two tabu search implementations, both based on keeping track of solutions (as opposed to moves) as proposed in Hanafi [5]. So, there will be no need for aspiration criteriums, and also there will be neither intensification nor diversification, as they are not necessary for a convergent tabu search method. We present two tabu search methods that only differ on the way a solution is tentatively improved on each iteration.

In order to escape from local optima, on tabu search, we have to sometimes make non-improving moves. For preventing cycling (falling on the same local optimum after a non-improving move), we keep a list of all the previously visited solutions, by means of a hash table. Moves that lead to a solution on this table are tabu. The only exception is a "blockage" situation.

A tabu search procedure takes an initial solution, constructed using one of the methods in 3, and iteratively tries to improve that solution according to one of the improvement methods that are described below. If it fails to improve a solution then it jumps to the best non-tabu solution in the neighbourhood, and tries to improve again from that solution. The method stops at the end of a predetermined number of iterations.

5.1 TripleChange improvements

Note that there are two possibilities for updating the best solution when searching in a particular neighbourhood.

The first one, called *best-updating* or *breadth-first*, consists of searching the best solution y^* in the entire neighbourhood. If it is better than our current solution x, then replace x by y^*. This method will hence search the whole neighbourhood of the current solution, even if improving solutions can be found on the early exploration of the neighbourhood.

The second method, called *better-updating* or *depth-first*, consists of replacing x during the local search whenever the current neighbour generated, y^i, is better than x. In this case, the subsequent "neighbour" y^{i+1} is obtained by applying the corresponding move to the *new* solution x, and hence does not belong to the initial neighbourhood. Better-updating is used in our implementation because it generally provides superior results, as the number of solutions "tried" in each local search is larger.

The TripleChange improvement procedure takes a solution and runs sequentially through three kinds of neighbourhoods – insert, swap and 1-change. For each neighbourhood, it searches for an improving solution in a better-updating way, trying a number of changes equal to the cardinality of the neighbourhood. Solutions in the tabu list are not accepted (except in a blockage situation). All solutions

are inserted in the tabu list after being accepted.

If using this procedure does not lead to a better solution, then we are on a local optimum of the three combined neighbourhoods, and we have to make a non-improving move.

5.2 RndChange improvements

In this method there is a parameter R which determines the number of tentative moves that is tried on the current solution. If during these tentatives an improving move is found, the method (immediately) returns it. Otherwise, we move to the best non-tabu solution found on the R iterations. For each of the R tentatives, a neighbourhood is drawn from the insert, swap and 1-change neighbourhoods, according to some probabilities of success. Then, the parameters required for executing a move on this neighbourhood are also randomly drawn. The implied solution is only accepted if it is not in the tabu list (except in a blockage situation). All solutions are inserted in the tabu list after being accepted.

6 Computational Results

We generated several benchmark instances, with three different dimensions. Tabu search takes considerably more CPU time than Local Search, however the resulting solutions are always better than those obtained by Local Search, since this method cannot escape local optima. MIP solutions are still better for small instances, but for larger instances Tabu Search obtains in less than 1 hour much better solutions than MIP running for 5 hours.

References

[1] C. van Eijl, *A Polyhedral Approach to the Discrete Lot-Sizing and Scheduling Problem,* PhD Thesis, Technische Universiteit Eindhoven (1996).

[2] R. Kuik, M.Salomon and L.N. van Wassenhove, *Batching decisions: structure and models,* European Journal of Operational Research 75 (1994) 243-263.

[3] A. Drexl, A. Kimms, *Lot sizing and scheduling - Survey and extensions,* European Journal of Operational Research 99 (1997) 221-235.

[4] Thomas Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley and Sons (1990) 427-446.

[5] Saïd Hanafi, *On the Convergence of Tabu Search* Journal of Heuristics 7 (2000) 47-58.