Compiling capacitated single-item lot-sizing problem in a CostMDD

Walid Khellaf, Romain Guillaume, Jacques Lamothe

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-10 (2022) 2024–2029

Compiling capacitated single-item lot-sizing problem in a CostMDD

Walid Khellaf * **Romain Guillaume** ** **Jacques Lamothe** ***

* *IRIT, university of Toulouse paul sabatier, 31062, France (e-mail: walid.khellaf@irit.fr).*

** *IRIT, University of Toulouse Jean Jaures, 31058, France (e-mail: romain.Guillaume@irit.fr)*

*** *Industrial Engineering Center, Mines Albi, Toulouse University, Albi, 81000, France, (e-mail: jacques.lamothe@mines-albi.fr)*

Abstract: This paper deals with capacitated single-item lot sizing problem (CLSP) in an interactive support system context. The interaction is done thanks to queries to the system: The decision-maker makes partial choices and asks about the consequences in terms of costs but also about possible inventory levels. Hence, three fundamental queries are investigated: Find production plans with a cost less than K; Find possible plans with conditioning on a pair of variables (production and inventory); Find the production plans that simultaneously satisfy the previous queries. A knowledge compilation approach is used and composed of two phases: offline and online. Offline, a top-down algorithm computes cost multivalued decision diagrams. While online, we show how cost multivalued decision diagrams can improve the reactivity of answers to queries.

The proposed approach is compared to classical constraint programming with CP Optimizer. Tests indicate that our algorithm is more efficient than CP re-solving in terms of computation time.

Keywords: Capacitated single-item lot-sizing, knowledge compilation, Multivalued decision diagrams, Queries.

1. INTRODUCTION

Beginning in the 1950s (Wagner and Whitin (1958), Manne (1958)), research on the lot sizing problem applied most of the methods of operations research. The goal is always to decide when and how much to produce (or order) in order to meet customer demand (firm or forecasted), period by period over a finite time horizon, in order to minimize the total expected cost (production costs and/or storage costs and/or setup costs, etc.) for one or more items while respecting general constraints. The resolution methods are generally classified A.Drexl (1997) into exact methods (dynamic programming Klein. (1971), branch-and-bound Chung et al. (1994)) and approximate methods (heuristics, meta-heuristics, Lagrangian relaxation J.R. Hardin (2005)).

The lot sizing (LS) problem is one of the most important problems in production planning, see the review in N.Brahimi (2006) with many versions depending on various characteristics A.Drexl (1997). Moreover, many specialized heuristics Degraeve (2007) or approaches based on multi-reference relaxation Manne (1958) have also been proposed. Here, is investigated a basic version, the single-product lot sizing problem (CLSP) Klein. (1971) with time-varying capacity. It is known to be NP-Hard Yanasse (1982) and Klein. (1971).

One of the main challenges of today's planning problem solving is interactivity Alexander N (2005): decision makers may want to work interactively with their planning systems in real time. For CLSP, this corresponds to checking the feasibility of a production plan with a given cost window, obtaining a production plan with conditioning on decision variables or with bounds on the cost. This difficulty of taking into account the additional set of queries and decision maker preferences can be achieved by knowledge compilation languages Henrik Reif Andersen (2010), Bart Selman (1996) and Cadoli (1997), or coupling decision diagrams and dynamic programming Hooker (2013). This differentiates our resolution approach from those reviewed in the CLSP literature.

The idea is to reformulate knowledge, offline, in a compact and efficient representation while keeping guarantees on the complexity of the online calculation of requests. This approach is used in several application fields of operational research and in artificial intelligence, especially in product configuration task Amilhastre J (2002) , Fargier H (2014), in the control of an autonomous system Alexandre N (2010).

This paper is organized as follows: Section 2, presents the capacitated single-item lot sizing and the associated queries. Section 3 and Section 4 presents the formal definitions of knowledge compilation, source, and target languages. Section 5 and Section 6 presents our proposal with two top-down algorithms for solving the offline and online problems. Section 7 presents a description of the experiments and the analysis of the first results obtained.

10.1016/j.ifacol.2022.10.005

Section 8 concludes the paper and gives some perspectives on the future work.

2. CAPACITATED LOT SIZING PROBLEM

Let's consider the traditional capacitated single item lot sizing . It aims to find a production schedule achieving an optimal trade-off between setup and inventory holding costs while complying with given capacity constraints and ensuring that demand for this product is satisfied. Two basic queries are identified : computing whole of the plans having the same cost, and imposing preferences on a quantity of inventory or production at some periods. The answers of those two queries informs the decision maker to address unforeseen events and thus better manage the resources and the production.

2.1 Mathematical formulation of CLSP

The following is a summary of the notations used in the representation of the data and of the decision variables.

Parameters:

T :	Number of periods t
d_t :	Demand at t
$\underline{X_t}$, $\overline{X_t}$:	Minimal, Maximal production quantities at t
I_0 :	Initial inventory
$\underline{I_t}, \overline{I_t}$:	Minimal, Maximal inventory at the end of t
p_t, s_t, h_t :	Production, Setup and Holding cost at t
f_t :	Cost function of X_t, Y_t, and I_t at t

Variables:

X_t :	Quantity produced at period t
I_t :	Inventory at the end of period t
Y_t :	Setup existence at period t

In order to be brief let $D_t^x = [\![\underline{X_t}, \overline{X_t}]\!]$ and $D_t^i = [\![\underline{I_t}, \overline{I_t}]\!]$.

A classical objective of CLSP is to minimize the sum of production, inventory holding, and preparation costs, which is expressed by (1). Constraints (c1.2) express the inventory balance. Constraints (c1.3) are the capacity constraints. Constraints (c1.4) and (c1.5) bound Production and Inventory. Constraints (c1.6) are the non-negativity conditions on the production quantities and the binary nature of the set up variables. However, the CLSP problem resolution usually produces only one optimal solution, if there is one.

$$Min \sum_{t=1}^{T} p_t X_t + s_t Y_t + h_t I_t = \sum_{t=1}^{T} f_t(X_t, Y_t, I_t) \quad (1)$$

$$(c1.2) \quad I_{t-1} + X_t = d_t + I_t \quad t \in [\![1, T]\!]$$
$$(c1.3) \quad X_t \leq \overline{X_t} * Y_t \quad t \in [\![1, T]\!]$$
$$(c1.4) \quad X_t \in D_t^x \quad t \in [\![1, T]\!]$$
$$(c1.5) \quad I_t \in D_t^i \quad t \in [\![1, T]\!]$$
$$(c1.6) \quad X_t, I_t \in \mathbb{N} \geq 0, Y_t \in \{0, 1\} \quad t \in [\![1, T]\!]$$

2.2 Queries

Although, once the CLSP is resolved, decision makers may want to understand the solution space and be able to query it, in real time, with some additional preferences. The research question here is: Is it more time efficient to put the problem 1 into a compiled form that is used to answer queries; or to resolve problem 1 to answer each query?

Let introduce two types of fundamental queries based of two types of preference restriction that a decision maker may have: on variables ou on costs.

Conditioning on the decision variables: The *conditioning queries* was introduced by Amilhastre J (2002). It is a partial assignment of decision variables. In CLSP, it corresponds to the resolution of the problem (1) with addition of constraints on the production or on the inventory quantity. In this paper a *conditioning query* is a vector Q of couples of conditioning:

$$\mathcal{Q} = (\mathcal{Q}_t) \quad (2)$$

with $\mathcal{Q}_t = \{Q_t^i\}_{i=1}^n$, with $(u_t|v_t)|\{t \in [\![1, T]\!], u_t \in D_t^x \cup \{*\}, v_t \in D_t^i \cup \{*\}\}$.

For instance $Q_t^i = (q_{X,t}^i|*)$ means that the production quantity of the period t is constrained to $q_{X,t}^i$ while the inventory quantity is not. $Q_t^i = (*|q_{I,t}^i)$ means that the inventory quantity is constrained but not the production quantity. And $Q_t^i = (q_{X,t}^i|q_{I,t}^i)$ means that both production and inventory quantities are constrained at t. Finally $\mathcal{Q}_t = \{(q_{X,t}^1|*), (q_{X,t}^2|*)\}$ means that the production quantity of the period t is constrained to be $q_{X,t}^1$ or $q_{X,t}^2$.

Conditioning on the cost function: The *cost conditioning query* constrains the maximal deviation from the optimal value (eq.3), sometimes called cost regular (Alessandro and Gilles P (2006)). Let H^* be the optimal cost of the problem (1) and Δ be the maximal deviation given by the decision maker, this query is formulated as follows:

$$\sum_{t=1}^{T} f_t(X_t, Y_t, I_t) \leq H^* + \Delta \quad (3)$$

3. KNOWLEDGE COMPILATION

The concept of knowledge compilation (KC) consists of solving once, offline, a problem (called a source language) that can be expressed by constraints Hooker (2013) or solution tuples. Then, compilation means that the set of feasible solutions is transformed into a compact graph format that allows to answer a series of online-queries in polynomial time (Bart Selman (1996), Cadoli (1997) and Darwiche A (2002)). This compilation concept is not new. For example, Alexandre N (2010) proposes a compilation of CSP into an interval automaton. Koriche et al. (2015) proposes a decomposed representation for a constraint network. In literature, two main types of methods allow compiling: Top-down methods Bergman D (2016) start from an empty set of solutions and add solutions in the representation, until having a representation of all the solutions of the problem. Conversely, Bottom-up methods Fargier H (2014) start from an unconstrained representation, add the constraints one by one to the graph. These graphical languages can take several forms of representation and also depend on the queries to be processed. They can be binary decision diagrams (BDD), or valued decision diagrams (VDD), or multi-values decision diagrams (MDD).

3.1 Source language for CLSP

Our source language is CLSP easily formulated as a constraint satisfaction problem (CSP) in (1). It is defined as a set of constraints involving a number of variables associated with a finite domain (c1.4 and c1.5). The objective is simply to find a set of values to assign to the variables so that all constraints are satisfied (c1.2 and c2.3).

4. TARGET LANGUAGE FOR CLSP

For the choice of compact graph format, called the target language, it is necessary to overcome the difficulty of allocating memory space for the compiled graph (called the knowledge map). Thus, it is interesting to adapt the representation to a partial knowledge of the nature of the requests Darwiche A (2002). The compiled form proposed in this paper (see Fig.1) will be in the family of Cost Multi-valued decision diagrams.
A Cost Multi valued decision diagram (Henrik Reif Andersen (2010), Alessandro and Gilles P (2006)) denoted M is a directed acyclic graph $M = (V, E, f)$ such that the function f associates to each edge $e \in E$, a couple $(X_t|I_t)$ and a real cost $f(X_t, I_t)$. In our proposed representation, the set of nodes of M are partitioned into $T + 1$ periods $V_1, ..V_{T+1}$. It initially contains a root node n_1, and one terminal nodes, the true terminal node 1. One node in V_t is associated to a value of the inventory I_t at t. The number of nodes in v_t is limited by the domain of I_t. It can have as many input edges as there are values in the domain ofX_t.

Thus, each edge $e \in E$ from $u \in V_{t-1}$ to $v \in V_t$ is labelled by a couple $((q_{X,t}|q_{I,t}), f(q_{X,t}, q_{I,t}))$. It means that from the state node u at $(t-1)$ to the state node v at t which is associated to the inventory value $q_{I,t}$, the production X_t is $q_{X,t}$, the inventory I_t is $q_{I,t}$ and f is the resulting cost for period t.

Each path from the root n_1 node to the node 1 is said to be valid. It represents a solution of the CLSP problem (1). The cost of a valid solution is the sum of cost of the edges of the path.
Let's add two attributes $(F^{\downarrow}(n_t), B^{\uparrow}(n_t))$ to each node $n_t \in V_t$, defined as follows:

- In Top-down manner:

$$F(n)^{\downarrow} = Min[F(u)^{\downarrow} + f(u \to n)|u \in V_t - 1]$$

- For bottom-up manner :

$$B(n)^{\uparrow} = Min[B(u)^{\uparrow} + f(n \to u)|u \in V_t + 1]$$

$F^{\downarrow}(n)$: represents shortest path (the minimum cost) from node (n_1) to node (n). $B^{\uparrow}(n)$: represents shortest path (the minimum cost) from node (n) to node (1)

This gives CostMDD graph denoted $CostM$ which takes into account the specificity of CLSP with respect to the two requests previously mentioned.

5. OFFLINE COMPILATION OF CLSP INTO COSTM

Our approach for compiling a constraint network into a cost Multi valued decision diagrams $CostM$ is a top-down method. The graph G in Algorithm 1 grows in a recursive way. At the beginning G has one layer V_1 with only one node $n1|I_0$. A layer of nodes is added in the graph G at each while loop. If it exists one realisable production plan, The line R1 returns the decision diagrams $CosM$ that represents all the admissible solutions. The Line R2 stops if there do not exist an admisible production plan i.e. any partition V_t is empty. At each call (period), are added the nodes corresponding to all feasible I and edges to all feasible pairs $(X|I)$ with the associated objective function f. we have indicated in the function $AddE2G$, that all the nodes (V_{T-1}) of the last period must be connected to the node $n_{T-1} = 1$ and we also calculate the attribute $F(n_t)$ for each node at the same time as the nodes are constructed. At the end of the resolutions corresponding to node (1), we initialize the attribute of this node to $B(1) = 0$, in Line R3, we add attributes $B(n_t)$ to the other nodes in a bottom-up manner from the graph $CostM$.

Algorithm 1: CLSP2CostM

Data: $G, t, \overline{\boldsymbol{X}}, \underline{\boldsymbol{X}}, \overline{\boldsymbol{I}}, \underline{\boldsymbol{I}}, T, d, I_0, p, s, h, f$
Result: $G = (V, E), E = (n_t, n_{t+1}, a, f(a))$
if $t = T$ **then**
  return G ; // R1
if $t \neq T \wedge V_t = \emptyset$ **then**
  return UNSAT ; // R2
$NodeAtPeriod =: \emptyset$; // Set of nodes at t
while $V_t \neq \emptyset$ **do**
  $n_{t-1} := V_t.pop(0)$; // Select node in t-1
  for $x \in \{\underline{X_t}, \overline{X_t}\}$ **do**
    $i = x + int(n_{t-1}.split("|")[1]) - d_t$
    if $i \in \{\underline{I_t}, \overline{I_t}\}$ **then**
      $n_t = "n" + str(t+1) + "|" + str(i); a = (x_t, i_t)$
      $F(n_t) = Min(F(n_t), F(n_{t-1}) + f(a)$
      $AddE2G(n_{t-1}, n_t, a, f(a), F(n_t))$
      if $n_t \notin NodeAtPeriod$ **then**
        $NodeAtPeriod.append(n_t)$; // Nnode
$V_{t+1} := NodeAtPeriod$; // Set of nodes at t+1
CLSP2CostM$(G, t+1, \overline{\boldsymbol{X}}, \underline{\boldsymbol{X}}, \overline{\boldsymbol{I}}, \underline{\boldsymbol{I}}, T, d, I_0, p, s, h, f)$;
// Backward
for $t \in T, ..., 1$ **do**
  for $n \in V_t$ **do**
    for $u \in$ *Output(n)* **do**
      $B(n) = Min(B(n), B(u) + f(n, u))$; // R3

The advantage of this algorithm is its incremental aspect. Finding the optimal solution is finding a shortest path from node n_1 to 1 in G. According to Lotfi and Yoon (1994) it can be done in pseudo-polynomial time $O(T * (\min\{D_t^x, D_t^i\})^2)$.

Example 1. Let's introduce a small CLSP instance with 4 periods ($T = 4$). The requirements are $d = [4, 2, 4, 8]$, the limited production capacity are 8 for all periods and the limited inventory are 1 for all periods. The production, inventory and setup cost are respectively: $p_t = [1, 2, 1, 3]$; $h_t = [1, 2, 3, 4]$; $s_t = [4, 3, 2, 1]$ for $t = 1, 2, 3, 4$.
Figure 1 presents the Multivalued decision diagrams $CostM$ by the CLSP compiled with the above data. Each node represents a feasible inventory in each period, for

example: starting from n1 to the first one whose initial stock is null, we can produce 4 and put nothing in stock with a cost of 8 or produce a quantity of 5 and put one in stock with a cost of 10. We execute the same operation until period T. The outgoing edges go to nodes having the same inventory quantity. However, there is a limit to the number of nodes in each V_t, which is bounded by the domain D_t^i. Finally there exists 12 paths (12 admissible solutions) from $n1$ to 1. And the shortest path has cost 46.

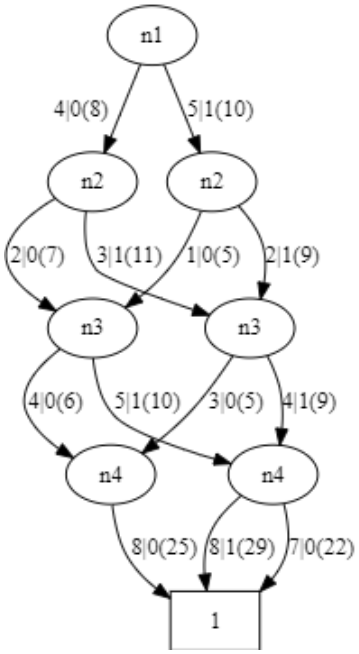

Fig. 1. Cost Multivalued decision diagrams (CostM) for the problem(1) with cost function.

6. ALGORITHM FOR ANSWERING QUERIES

6.1 Optimisation under conditioning

Let suppose that the CLSP is compiled by Algorithm 1 into CostM (see for instance figure 1) so all the feasible solutions are loaded in the $CostM$. Now, the decision maker gives his assignment list Q_t at each period t. In algorithm 2, for each edge of the graph at each period, the edges that are not consistent with assignment Q_t are deleted (in for t loop). As a consequence of deletion, two conditions must be verified in the while loop: if the number of edges outgoing from the node (n_t) is null, all the n_t input edges must be deleted. if the number of edges inputting node (n_t) is null, all the n_t outgoing edges must be deleted.

Algorithm 2: Conditioning queries

Data: $G = (V, E), \mathcal{Q}$
Result: $G = (V, E); E = (n_t, n_{t+1}, a, f(a))$
$LREdge =: \emptyset$; // Edge deleted by $\mathcal{Q}$
for $t \in [\![1, T]\!]$ **do**
  for $e \in E_t$ **do**
    if $a \neq Q_t$ **then**
      $LREdge.append(e)$
while $LREdge \neq \emptyset$ **do**
  $e(n_t, n_{t+1}, a) := LREdge.pop(0)$ $remove(e)$;
  if $Output(n_t) = 0$ **then**
    $LREdge.appendAll(n_{t-1} \rightarrow n_t)$
  if $Input(n_{t+1}) = 0$ **then**
    $LREdge.appendAll(n_{t+1} \rightarrow n_{t+2})$

6.2 Possible solutions under cost conditioning

This step of updating is necessary when the first request is applied ($LREge \neq 0$), the attributes ($F \wedge B$) must be recalculated and their information will be preserved in the nodes (V_t) of each period $\forall n_t \in V_t$ and $\forall t \in T$.

In R.K. Ahuja (1993), two cost attibutes for any node n of a $CostM$ Graph have been introduced: $F(n)^{\downarrow}$, $B(n)^{\uparrow}$. They can be calculated thanks to a single forward and backward operation in $CostM$ with a complexity of O(|V| + |E|) time and space of size $CostM$.

The minimum cost H^* can be found in the $F^{\downarrow}[1]$ or in the$B^{\uparrow}[n1]$. Now, the decision maker introduces a maximal deviation of the objective Δ (second type of query). Algorithm reduces the graph $CostM$ according to the cost window thanks to 2 conditions : - if in a node n, F(n) + B(n) ¡ H* + Δ, then node n can be deleted. - if for an edge e(u,v) from node u to node v, F(U) + f(u,v) +B(v) ¡ H* + Δ, then edge e can be deleted.

Algorithm 3: Conditioning on cost function

Data: $(G(V, E), H^*, \Delta)$
Result: $CostMQ =: MultiDAG(V, E, f)$
for $t \in 1, ..., T$ **do**
  for $n_t \in V_t$ **do**
    for $e \in n_t \rightarrow n_{t+1}$ **do**
      if $F(n_t) + f_{a,t} + B(n_{t+1}) \leq H^* + \Delta$ **then**
        $AddE2CostMQ(n_t, n_{t+1}, a, f(a))$
return $CostMQ : \overrightarrow{CostMQ}$

Example 2. The two decision maker queries are assumed to be defined as follows :
- q1: a packing list contains $\mathcal{Q} = \{Q_2 = (*|0)\}$:hence the decision maker want to have in period two an inventory quantity 0 at the end of period 2. - q2 : is a cost interval $\Delta = 2$ for which the decision maker tolerates expressing as $\sum_{t=1}^{T} f_t(X_t, Y_t, I_t) \leq H^* + \Delta; H^* = 46$.

The set of solutions by the intersection of the results of two queries can be visualised by the acyclic decision diagrams represented in figure 4. Each edge is drawn differently if it was eliminated because of query q1 (dashed) and deletion consequence res q1(dotted) or q2 (darker white), or remainder (full black).

In this example, there are three solutions satisfying the decision maker's requests are represents by the black path in Figure 1. $Sol_1 = (4|0, 2|0, 4|0, 8|0)$ and $Sol_2 = (5|1, 1|0, 4|0, 8|0)$ with cost 46, $Sol_3 = (5|1, 1|0, 5|1, 7|0)$ with cost 47.

This is one of the important properties of $Costmdd$ which plays a crucial part in decision making at the deciders level to formalize all the admissible decisions.

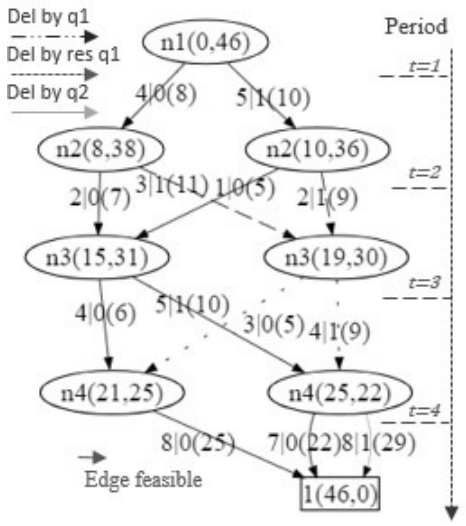

Fig. 2. CostMQ after both conditioning in CostM

7. EXPERIMENTS

To evaluate the effectiveness of our approach, i.e. compile the problem into *CostM* and then answer the queries, we compare it to:

(1) Exactly integrate the constraints in the initial problem and restart solving, noted: CP.

The experiments were performed on a computer equipped with an Intel(R) i7-8665u cpu at 2.1GHz with 16 Gb of RAM. The *CostM* code was implemented in python version 3.8.8 using NetworkX to write the G-graph and in Graphviz dot format for export. In order to compare the different results, we used a library from the ILOG CPLEX Academic Studio V.20.1 for the *CLSP* problem, and the implementation for *csp* is done on CP Optimizer 20.1.0.

To compare each mode in terms of time, we adapted the problem data set of (Lotfi and Yoon (1994)) to create random test data. We tested the problems over the following periods: $T \in \{6, 12, 24, 48\}$ and $\Delta \in \{0, 20, 40, 60, 80, 100\}$. The demand for period t $(d_t,)$ is uniformly distributed between 67 and 200. The size of the domain D_t^i is set to 50. As the minimal inventory is fixed at 50, inventories can vary in the interval [50; 100]. The minimal production quantity and initial inventory are set to 0. The values of the production capacity, maximum inventory, and costs are generated thanks to uniform distributions as defined in table 1. First, the offline solving. The following table

Table 1. Problem data

Quantity	Uniform	Cost	Uniform
Max production $\overline{X_t}$	(1,1.5)*α	Production p_t	(10,30)
Max inventory $\overline{I_t}$	(50,100)	Setup s_t	(1,1.5)*β
Min inventory $\underline{I_t}$	50	Holding h_t	(0.5,1.5)

α: Constant quantity=1200, β: Constant cost=1600

1 shows the CLSP problem solving time by the CP optimizer and our top-down approach to find a set of optimal solutions. we implemented our CLSP model and solved it with Cplex for the existence of the optimal solution for this data generated, and we generated an experimental design of 70 tests. It can be observed (see in table 2) that the time (CPU) of both methods are sensitive to the number of periods T, and the growth rate of our approach is lower. This method allows a huge time saving when the number of periods is large. It has a number of nodes $\sum_{t=1}^{T}(min\{D_t^x, D_t^i\}+2)$. Thus his limitation is on the size of the memory.

Table 2. Comparison of average offline resolution time (seconds) with $\Delta = 0$

Period	T=6	T=12	T=24	T=48
CLSP2CostM	0.28	1.5	2.01	4.1
CP Optimiser	4.12	7.34	13.78	18.02

The following figure 3 demonstrates how much time it takes to find plans with an equivalent cost to the different Δ parameters. We can see clearly the advantage of choosing a compiled *CostM* which stays constant (because the search is limited in the decision diagram) in time over all other options which increase at a rate that is approximately related to the cost. We also tested the time required

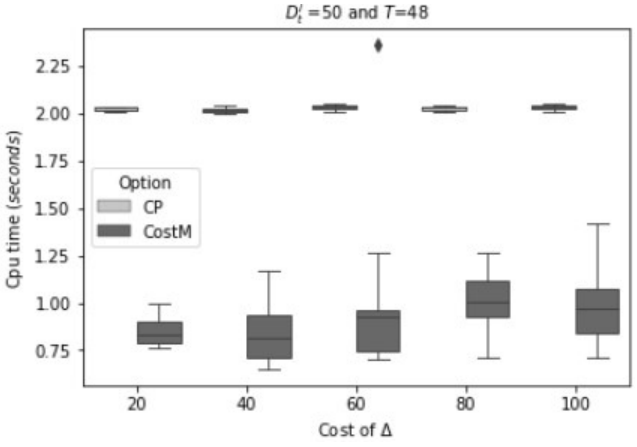

Fig. 3. Time required to resolve a CLSP problem online with conditioning on the cost function

when a user simply prefers to assign a very large number of potential decisions to a variable and to determine the set of feasibility's for their problem as rapidly.

we randomly generated without replacement a list ($\mathcal{Q}_t = \{Q_t^i\}_{i=1}^{K}$) containing a number of K values from the set of D_t^x. Q_t^k representing the allowed assignments to the variable production quantities at period t. We compare our CostM approach, with these two different solving strategies:

- S1: Compiling CLSP with queries in $CLSP2CostM$.
- S2: Compiling CLSP without the queries then applying algorithm 2 in CostM

The figure 4 shows the time versus the number of queries, especially the horizontal line where a change of strategies is "recommended". e.g: For more than 200 queries, the use of CostM is more advantageous in time compared to including these queries in the initial CLSP problem. So, in CLSP, containing a small number of queries needs fewer edges to build in the costM but otherwise, the deletion of edges is very fast.

Therefore, the results in figures (4 and 3) show that the interactive resolution based on the *CostM* reduces

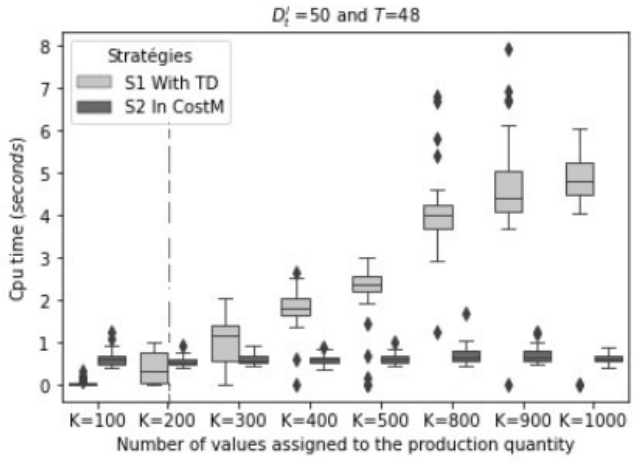

Fig. 4. Determination of the switching point of strategy

the time significantly when considering the two types of proposed queries.

8. CONCLUSION

This paper proposes a compilation data structure *CostM* for a lot sizing problem. It allows efficient management of the CLSP solution space once compiled. Moreover, it helps the decider by propagating the consequences of his choices i.e. values he sets to the decision variables or to the cost function.

We have presented the two types of efficient algorithms: a knowledge compiler that generates a knowledge graph representing the solution space; 2 querying algorithms that compute the impact of queries on the reduction of the knowledge graph. On a set of randomly generated data, the interest in computation time of this approach has been validated when compared to solvers resolution time (CP).

In our current and future work, we aim to focus mainly on two industrial and engineering axes: First, to use that approach for supporting the negociation between a factory and a distribution center; Also, to extend the structure of the compile *CostM* for considering multi-product lot sizing, and the addition of other parameters such as the backorder. Second, it is a question of implementing the full tool via these two algorithms and making it available for inputs in format *xcsp*.

ACKNOWLEDGEMENTS

The authors would like to thank the ANR for funding the CAASC project. They also thank the members of the AI Interdisciplinary AI Interdisciplinary Institute ANITI Grant no. ANR-19-PI3A-0004.

REFERENCES

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

Alessandro, Z. and Gilles P, M.M. (2006). Planning with soft regular constraints. *Workshop on Preferences and Soft Constraints in Planning*, 77–78.

Alexander N, R.F. (2005). Constraints and ai planning. *Intelligent Systems, IEEE*, 69–70.

Alexandre N, H.F. (2010). Knowledge compilation using interval automata and applications to planning. *ECAI*, 459–464.

Amilhastre J, F.H. (2002). Consistency restoration and explanations in dynamic csps-application to configuration. *Artificial Intelligence*, 199–234.

Bart Selman, H.K. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2), 193–224.

Bergman D, A.A.J. (2016). Decision diagrams for optimization. arti intelligence: Foundations, theory, and algorithms. *Springer*.

Cadoli, M.D. (1997). A survey on knowledge compilation. *AI Communications*, 137–150.

Chung, C.S., Flynn, J., and Lin, C.H.M. (1994). An effective algorithm for the capacitated single item lot size problem. *European Journal of Operational Research*, 75(2), 427–440.

Darwiche A, P.M. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17, 229–264.

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

Fargier H, M.P. (2014). Compacité pratique des diagrammes de décision valués normalisation, heuristiques et expérimentation. *Artificial Intelligence*, 571–592.

Henrik Reif Andersen, T.H. (2010). interactive cost configuration over decision diagrams. *Artificial Intelligence Research*, 37, 99–139.

Hooker, J.N. (2013). Decision diagrams and dynamic programming. *InInternational Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 94–110.

J.R. Hardin, G.N. (2005). Analysis of bounds for a capacitated single-item lot-sizing problem. *Computers and Operations Research*.

Klein., M.F. (1971). Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1), 12–20.

Koriche, F., Lagniez, J.M., and Marquis, P. (2015). Compiling constraint networks into multivalued decomposable decision graphs. *24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 332–338.

Lotfi, V. and Yoon, Y.S. (1994). An algorithm for the single-item capacitated lot-sizing problem with concave production and holding costs. *Operational Research Society*, 45(8), 934–941.

Manne, A. (1958). Programming of economic lot-sizes. *Management Science*, 4(2), 115–135.

N.Brahimi, S.D.P. (2006). Single item lot sizing problems. *European Journal of Operational Research*, 168, 1–16.

R.K. Ahuja, T. Magnanti, J.O. (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

Wagner, H.M. and Whitin, T.M. (1958). Dynamic version of the economic lot size model. *Management Science*, 5(12), 89–96.

Yanasse, G. (1982). Computational complexity of the capacitated lot size problem. *Management Science*, 28(10), 1174–1186.