KYBERNETIKA — VOLUME *48* (2012), NUMBER 2, PAGES 329–345

A FAST LAGRANGIAN HEURISTIC FOR LARGE-SCALE CAPACITATED LOT-SIZE PROBLEMS WITH RESTRICTED COST STRUCTURES

Kjetil K. Haugen, Guillaume Lanquepin-Chesnais and Asmund Olstad

In this paper, we demonstrate the computational consequences of making a simple assumption on production cost structures in capacitated lot-size problems. Our results indicate that our cost assumption of increased productivity over time has dramatic effects on the problem sizes which are solvable.

Our experiments indicate that problems with more than 1000 products in more than 1000 time periods may be solved within reasonable time. The Lagrangian decomposition algorithm we use does of course not guarantee optimality, but our results indicate surprisingly narrow gaps for such large-scale cases – in most cases significantly outperforming CPLEX.

We also demonstrate that general CLSP's can benefit greatly from applying our proposed heuristic.

Keywords: heuristics, capacitated lot-sizing, restricted cost structures

Classification: 65K05, 90B30, 68W99

1. INTRODUCTION

The capacitated lot-size problem (CLSP) has drawn significant research attention since Manne's [16] original MILP formulation more than 50 years ago. Several review articles [6, 13, 15] and [3] indicate the size, diversity and complexity of CLSP-research.

Karimi et al. [13] categorizes CLSP algorithmic research related to solution methods in various categories. Of particular interest for this work is the subcategory of Relaxation Heuristics. According to Karimi et al. [13], such heuristics usually produce better quality solutions, are more general and allow extensions to different problems.

In spite of extensive research, the size of problems practically feasible are limited. By practically feasible, we mean problems that can be solved in reasonable time within an acceptable deviation from optimum. As modern product variety may involve vast product counts, the reported sizes of problems practically feasible are still far from real world cases.

However, some articles do report large-scale CLSP-like problems solved. Diaby et al. [5] report problem sizes of up to 5000 products in 30 time periods. Comparable problem sizes are reported by Haugen et al. [11]. Common for these applications are reformulations of the CLSP introducing added problem flexibility. Diaby et al. [5] allow

overtime usage, while Haugen et al. [11, 10] introduce demand-affecting prices. Hence, both these two approaches solve other problems than CLSP.

Our aim in this article, is to investigate a different angle of attack in order to achieve solution speed for problem sizes closer to real world cases without relaxing the original CLSP formulation. Still, certain research facts must be considered; the strong NP-hardness of CLSP [2, 4]. Consequently, we need to apply some tricks.

In section 2, we introduce our CLSP formulation while section 3 discusses our algorithmic set-up as well as our cost assumptions. Section 4 contains our numerical experiments, while the article concludes and suggests possible further research in section 5.

2. OUR CLSP FORMULATION

At this point, a mathematical formulation of CLSP is appropriate. The Single-level (big bucket) CLSP may be formulated as:

$$\text{Minimise } Z = \sum_{t=1}^{T} \sum_{j=1}^{J} [s_{jt}\delta_{jt} + h_{jt}I_{jt} + c_{jt}x_{jt}] \tag{1}$$

$$\begin{aligned}
\text{Subject to} \quad & \textstyle\sum_{j=1}^{J} a_{jt}x_{jt} \leq R_t && \forall\, t && (2)\\
& x_{jt} + I_{j,t-1} - I_{jt} = d_{jt} && \forall\, jt && (3)\\
& 0 \leq x_{jt} \leq M_{jt}\delta_{jt} && \forall\, jt && (4)\\
& I_{jt} \geq 0, && \forall\, jt && (5)\\
& \delta_{jt} \in \{0,1\} && \forall\, jt && (6)\\
& j \in \{1,2,\ldots,J\} && && (7)\\
& t \in \{1,2,\ldots,T\} && && (8)
\end{aligned}$$

with decision variables:

x_{jt} : the amount of item j produced in period t
I_{jt} : amount of item j held in inventory between periods $t, t+1$
δ_{jt} : 1 if item j is produced in period t ; 0 otherwise

and parameters:

T : number of time periods
J : number of items
s_{jt} : setup cost for item$j$ in period t
h_{jt} : storage cost for item j between periods $t, t+1$
c_{jt} : unit production cost for item j in period t

a_{jt} : consumption of capacitated resource by item j in period t
R_t : amount of capacity resource available in period t
d_{jt} : given future demand for product j in time period t
M_{jt} : this is the smallest possible "Big M" needed to take care of binary logic for product j in period $t, M_{jt} = \sum_{s=t}^{T} d_{js}$,
I_0 : Initial inventory > 0.

The objective (1) contains total costs (set-up, storage and production). Constraints (2) are the capacity constraints and (3) inventory balancing, while (4), with the integrality requirement (6), enforces set-up when production is positive. Non-negativity constraints (5) are reasonable. (7) and (8) define ranges for indices.

3. OUR ALGORITHMIC SET-UP

3.1. Relaxation heuristics — Lagrangian relaxation

The trick mentioned in section 1 is related to assumptions on cost structures. If we can make reasonable (read practically relevant) assumptions on cost structures, easing the computational burden in certain sub problems, we can hope to achieve speed-ups compared to existing research.

In order to explain how and why our approach works, we need to explain the basics of Lagrangian relaxation. As indicated in section 1, Relaxation Heuristics is a central technique in CLSP-heuristic research. The technique works as follows: Firstly, the per period capacity constraint (2) is relaxed (added to the objective (1)). As a consequence, J decoupled sub problems – for instance solvable by the Wagner–Whitin algorithm [20][1] – emerges. By solving these problems, set-up structures for each product j is obtained, and the solution for the common problem will constitute a lower bound for the original problem. Secondly, this set-up structure is applied in order to fix corresponding binary (set-up) variables resulting in an LP sub problem, producing an upper bound on the solution of the original problem, which is solved to produce Lagrangian multipliers (shadow prices). We name the first problem set LB-problems, while the second problem set is named UB-problems[2]. These values are then fed back into the original relaxed problem as new multipliers. Now, the basic Lagrangian relaxation loop is constructed and improved solutions are obtained running back and forth between the UB- and the LB-problems.

The above described algorithmic set-up was used by both Thizy and Wassenhove [18] and Trigeiro [19] in their trend-setting articles. Hence, our choice of relaxing the capacity constraint (2) is in accordance with existing theory. This fact is also supported by Chen and Thizy [4] as well as research by Haugen et al. [10, 11].

The main difference between these approaches is perhaps related to Trigeiro's use of Lagrangian multiplier smoothing techniques. A similar set-up (including smoothing) was used by Haugen et al. [10, 11].

[1] We apply a slightly modified version of the original algorithm as reported by Wagelmans et al. [21]

[2] See appendices B and C for additional information on both formulation as well as solution strategies for these sub problems.

3.2. Cost assumptions

The specialized cost assumptions we apply are needed in order to solve the UB-problems more efficient. We apply a technique recently proposed and demonstrated by Haugen et al. [9]. As the technique is well described in [9], we will just discuss it briefly. In order to keep things simple, we investigate a single product case. The multi product case is discussed in detail in [9].

If all binary variables (δ_{jt}'s) in the CLSP-formulation (1) – (8) are fixed, the resulting LP may be formulated[3]:

$$\text{Minimise } Z = \sum_{t=1}^{\hat{T}} [h_t I_t + c_t x_t] \tag{9}$$

$$\begin{aligned} \text{Subject to} \qquad x_t &\leq R_t && \forall\, t && (10)\\ x_t + I_{t-1} - I_t &= d_t && \forall\, t && (11)\\ x_t &\geq 0 && \forall\, t && (12)\\ I_t &\geq 0 && \forall\, t. && (13) \end{aligned}$$

If we start by assuming constant production costs ($c_1 = c_2 = \ldots c_T = c$), it is straightforward to show (see Haugen et al. [9]) that the objective can be reformulated to:

$$\text{Minimise } Z = \sum_{t=1}^{T} h_t I_t. \tag{14}$$

Given this, it is obvious that the optimal solution to the LP must be the "Just-in-time" solution ($x_t^* = d_t \forall\, t$), given that this solution is feasible (i. e. not violating the constraint (10)). If the capacity constraint is violated say in time period τ_1, then we can arrive at an optimal solution by "shuffling"[4] production from this time period τ_1 to the nearest time period ($\tau_2 < \tau_1$) with spare capacity. Any other time period choice further away, will increase total inventory costs. Furthermore, it is likewise clear that given the following revised production cost assumption;

$$c_1 \geq c_2 \geq \ldots \geq c_T \tag{15}$$

the above argument is still valid.

This argument is easily extended to the multi-item case – see Haugen et al. [9], providing an extremely efficient specialized algorithm for the (LP) UB-problems.

If our proposed algorithm for CLSP turns out to be successful, which later sections indeed will reveal, the practical usability will rely on whether our added cost restrictions fits reality. Our cost assumption (15) is limited to production costs. Production costs are market determined. As such, it would be very surprising if a certain producer would be able to predict them.

[3]Note that $\hat{T}$ in equation (9) typically is different from T ($\hat{T} \leq T$) in equation (1) due to fixation of the binary variables.

[4]We name this algorithm the " bulldozer" algorithm for further reference. The choice of name is due to it's similarity to a bulldozer clearing snow by shuffling it to the closest possible storage location.

Production costs contain (primarily) wages and technology. Both factors are hard to predict. If some producer actually could predict changes from a given average value, it might be opportune to ask if the producer perhaps should engage in finance market speculative activities instead of running CLSP-problems to organize local production. In any case, one would typically observe decreasing production costs if empirical observations are performed. One could think about most modern products who at the introduction phase typically will have high per unit production costs. Even if wages, and most other production inputs have increasing costs, per unit product costs normally decrease due to productivity increase over a product's life-cycle. Let us try to sum up: If production costs are to change, they should (logically) change in a decreasing way, covered by our assumption (15). Alternatively, in the short run, it is hard to see producers being able to beat the market by forecasting labour and technology better.

3.3. An added approximation

As indicated by several authors (see e. g. [8, 12] or [11]), solving sub-problems to optimality in Lagrangian decomposition may prove inefficient. The main point in these techniques is to achieve direction as opposed to exactness. As such, we could try to improve speed by investigating certain approximations for our proposed sub problem solvers. The "bulldozer" algorithm discussed above is simply so fast, that added approximations seems unlikely to induce improvements.

3.4. Solving the LB-problem

The LB-problems however, may be interesting to investigate further in approximations or new heuristics. Even though the DP-based WW-algorithm [20, 21], is extremely efficient, it provides much of the computational burden in our proposed algorithmic set-up. As a consequence, we propose (in many ways similar to Kirca and Kokten [14]) to apply a very simple EOQ[5]-approximation [7] as an alternative to the exact[6] WW-solver. A more thorough description of this approximation is left for appendix A.

Our algorithmic set-up is summarized in Figure 1.

The LB-problems are either solved by a "normal" WW-algorithm [20, 21] or by our EOQ-approximation described in appendix A. The UB-problems are solved by the "Bulldozer"-algorithm described in subsection 3.2.

A final point should be mentioned. In order to ease the problem of achieving feasibility in the "Bulldozer"-stage, we have allowed a dynamic Lagrangian multiplier smoothing procedure. That is, we open up for using different smoothing parameters at different iteration steps in our Lagrangian relaxation heuristic.

4. NUMERICAL EXPERIMENTS

4.1. Computational set-up

All our numerical experiments are performed on a HP z400 equipped with the Intel Xeon W3520 processor and 8GB DDR3 RAM. The software platform contains gcc: 4.4.5,

[5]Economic Order Quantity

[6]Exactness means to optimality here.

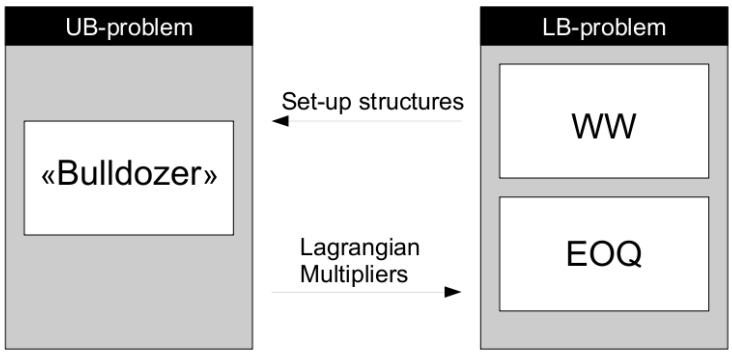

Fig. 1. The algorithmic set-up.

Python: 2.6.6, Pyrex: 0.9.8.5, Cplex: 12.2.0.0 (from python wrapper) and Cython: 0.14 (only used for the tests in appendix A). The experiments are mainly done in Ubuntu Linux 10.10 64bits Maverick, but certain operations were performed in Windows XP 32bits SP3 (running inside virtualbox 4.0.0).

All additional information on cases, run-times, graphics and so forth are (of course) obtainable from the authors upon request.

4.2. Trigeiro cases

We start out by examining our heuristics performance on some standard problems from literature, originally defined by Trigeiro et al [19] and collected by Wolsey and Belvaux [1]. We pick the six standard cases; `tr6-15`, `tr6-30`, `tr12-15`, `tr12-30`, `tr24-15` and `tr24-30`. The numbers in the case-names refers to the number of products and time periods respectively[7].

Table 1 sums up our results. (All cases are run with "WW" as the LB-problem-solver.)

Case	J	T	Z_h	Z_{CPLEX}	$CPU(s)$	Z^*	$CPU^*(s)$
`tr6-15`	6	15	39896	52800	0.0125	37092	0.1213
`tr6-30`	6	30	69899	86600	0.0499	60835	0.6587
`tr12-15`	12	15	78563	81640	0.6587	70922	0.8751
`tr12-30`	12	30	160746	189249	0.6039	129788	12.0592
`tr24-15`	24	15	143418	192000	0.0763	135970	0.9582
`tr24-30`	24	30	310977	434600	0.1433	287425	3.8429

Tab. 1. Results for the Trigeiro cases.

In Table 1, the three first columns denote case name, number of products (J) and number of time periods (T) respectively. The columns labelled Z_h and Z_{CPLEX} contains

[7] That is, the `tr6-15`-case contains 6 products in 15 time periods.

CLSP objective function values for our heuristic and CPLEX. These objective function values are obtained by running our heuristic and CPLEX (pairwise) for the same amount of CPU-time (in seconds) given in the column labelled $CPU(s)$. The final columns (Z^*) and ($CPU^*(s)$) contain optimal objective function values found by CPLEX as well as corresponding CPU-times.

We observe immediately, that our heuristic (by this measurement method) outperforms CPLEX for all cases as $Z_h \ll Z_{\text{CPLEX}}$.

Table 2, calculated based on the information in Table 1 provides a clearer image.

Case	$\left\lvert\frac{Z_h - Z_{\text{CPLEX}}}{Z_h}\right\rvert \cdot 100\,\%$	$\left\lvert\frac{Z^* - Z_h}{Z^*}\right\rvert \cdot 100\,\%$	$\left\lvert\frac{Z^* - Z_{\text{CPLEX}}}{Z^*}\right\rvert \cdot 100\,\%$
`tr6-15`	32.34 %	7.56 %	42.35 %
`tr6-30`	23.89 %	14.90 %	42.35 %
`tr12-15`	3.92 %	10.77 %	15.11 %
`tr12-30`	17.73 %	23.85 %	45.81 %
`tr24-15`	33.87 %	5.48 %	41.21 %
`tr24-30`	39.75 %	8.19 %	51.20 %
Average (%)	25.25 %	11.79 %	39.67 %

Tab. 2. Deviations (%) for the Trigeiro cases.

The first result-column ($\left\lvert\frac{Z_h - Z_{\text{CPLEX}}}{Z_h}\right\rvert \cdot 100\,\%$) in Table 2 shows that our heuristic on average produces around 25 % better solutions than CPLEX for the same amount of execution time. Furthermore, our heuristic is on average slightly below 12 % from the optimal value ($\left\lvert\frac{Z^* - Z_h}{Z^*}\right\rvert \cdot 100\,\%$), while CPLEX is around 40 % away from optimality ($\left\lvert\frac{Z^* - Z_{\text{CPLEX}}}{Z^*}\right\rvert \cdot 100\,\%$).

The above cases, originally considered hard, are of course easily solved on today's hard- and software. In fact, CPLEX proves optimality for the Trigeiro cases ranging from 0.12 CPU-seconds for the `tr6-15` case up to 12.06 CPU-seconds for the `tr12-30` case. As such, our algorithm does not introduce any "revolution" for these cases, apart form the obvious fact that it produces good quality solutions significantly faster than CPLEX.

4.3. Some medium sized examples and behaviour compared to CPLEX

In order to test our heuristic more seriously, we have made our own cases. These cases range from relatively small cases 10 × 10 (10 products in 10 time) periods up to quite large cases of 200 × 10, 10 × 200 and 50 × 50. The terminology $Y \times Z$ will be used in subsequent sections where Y is number of products and Z is number of time periods.

For all costs, holding costs (h_{jt}) are always set to 1 and production costs (c_{jt}) to 20. The set-up costs (s_{jt}) are randomly picked within the set $[100, 200, 300, 400, 500]$ but kept constant after the pick. Demand is a matrix (product × period) of pseudo-Gaussian (rounded to get integer and positive numbers). The capacity constraint (R_t) at time period t is the multiplication of a Gaussian (mean=ratio, var=0.01) generated number and the sum of the demand for the period.

All data are written in two formats; AMPL and MPS, the AMPL format is read by our heuristic and the MPS-file by CPLEX.

Before we move into some computationally more interesting cases, we present a simple analysis of our algorithm's behaviour related to problem tightness. Figure 2 shows our results.

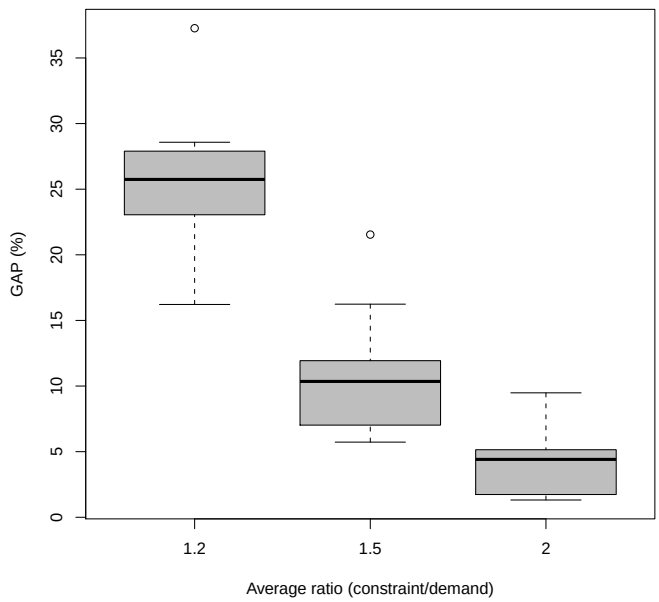

Fig. 2. Heuristic behaviour as a function of problem tightness.

The boxplot[8] in Figure 2 is constructed by running a total of 35 cases ranging from 10×10 by 5×20 up to 50×10. We have defined three different problem classes based on problem tightness visualized on the *Average ratio*-axis in the box plot. Each box holds a different tightness computed as $J \cdot \frac{\sum_{t=1}^{T} R_t}{\sum_{j=1}^{J} \sum_{t=1}^{T} d_{jt}}$. Consequently, the family of cases representing the left-most box (the value 1.2) are really tight problems, while the right-most box of 2.0 represents more loose problems.

As can be readily observed from Figure 2, our heuristic (not very surprisingly) behaves better for less tight problems. For instance, the average GAP(%) from the optimal value is found slightly above 25 %[9] for the tightest family, while the relatively loose family with an *Average ratio* of 2 on average has only around 5 % deviation from the optimal value.

This observation does not necessarily indicate that our heuristic performs badly on tight cases, but that it performs significantly better on less tight cases. The Trigeiro cases in Tables 1, 2 are in fact quite tight with an *Average ratio* of 1.32.

[8]We use standard boxplots, visualising probability distributions efficiently. The height of a box gives very visible information on variance of the distribution. A narrow box indicates small variance.

[9]The solid black line in the leftmost box.

The main body of medium sized problems in our analysis is reported in Figures 3, 4 and 5. These cases[10], substantially bigger, contain cases within 10×200, 200×10 and 50×50.

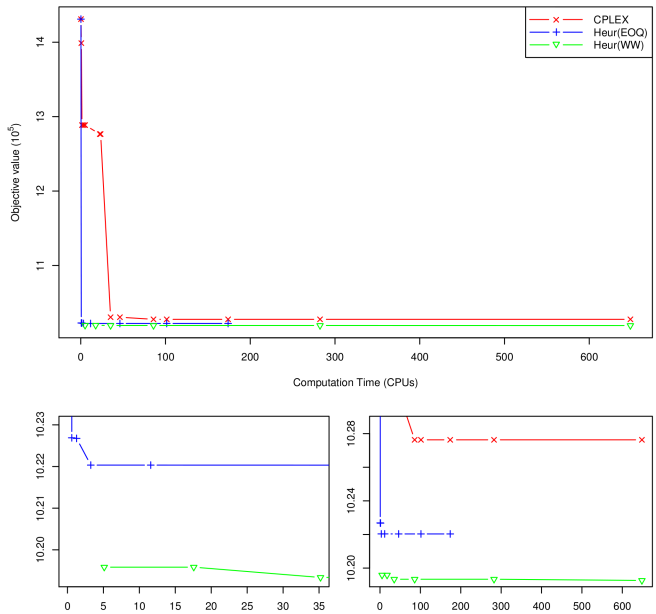

Fig. 3. Objective function value as a function of CPU-time for 10×200 cases.

All Figures 3, 4 and 5 contain CLSP objective function value as a function of CPU-time. Each figure contains 3 different solution instances; CPLEX, and our algorithm either with "WW" or EQO as LB-problem solvers. Furthermore, the two panes at the bottom are zoomed parts of the chart on top.

A closer inspection of Figures 3 and 4 reveals a similar type of pattern. The EOQ version (the blue curve) finds a relatively good solution extremely fast. Slightly slower, the "WW" version (the green curve) finds a better solution, while CPLEX (the red curve) struggles and do not show solutions better than hour heuristic within the observed interval of computational time.

However, the Branch and Bound Algorithm of CPLEX should of course in the end produce the optimal value, which is not guaranteed by our heuristic. Indeed, this is observed in Figure 5. Still, our heuristics produce relatively good solutions very fast compared to CPLEX.

[10] 15 generated cases in each category.

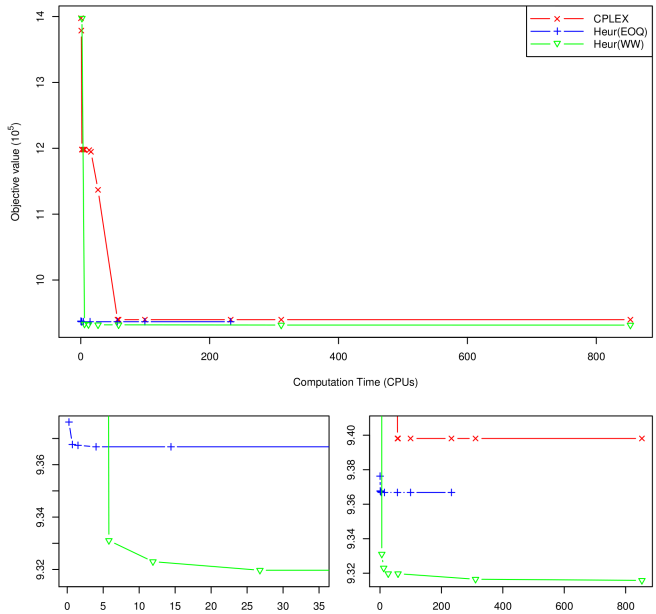

Fig. 4. Objective function value as a function of CPU-time for 50×50 cases.

4.4. Two large-scale cases

As the title of this article indicates, our main objective have been to investigate our heuristics' behaviour for really large-scale cases. Even though our previous cases indicate efficient behaviour, the ultimate test should include much larger problem instances. We have tested a 1000×1000 case as well as a 5000×5000 case. To give you some indication of size, the 1000×1000 case produces a MPS-file with a size around 1GB, while the bigger case of 5000×5000 delivers a 25GB MPS-file.

Not unexpectedly, CPLEX struggled with these cases. In fact, already for the 1000×1000 case, CPLEX crashed, after close to 4 (real-time) hours finding a singular solution with an objective function value of 580218000. This solution was found after 30 (real-time) minutes. As a consequence, we did not bother to test CPLEX for the 5000×5000 case. Actually, the real-time needed just to generate the MPS-file would be close to a week for this case.

However, our algorithms provided surprisingly good and fast results. Table 3 shows the output of our algorithm with EOQ as LB-problem solver.

As can be observed by Table 3, we achieve very good results. In less than 353 CPU-seconds an objective function value with around a 0.3 % GAP[11] is obtained.

Running the same case with "WW" as the LB-problem solver produces results as

[11] Strictly speaking, this GAP is an approximation as our exact LB-problem solver is substituted with the EOQ-approximation, still the upper bound is exact.

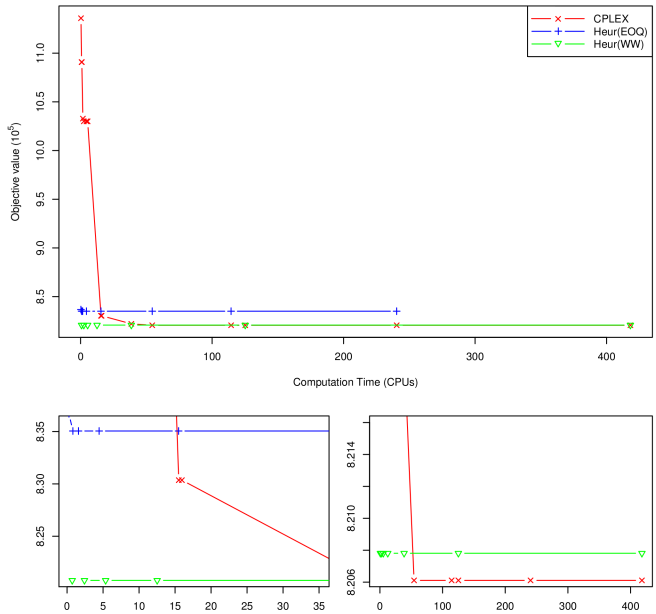

Fig. 5. Objective function value as a function of CPU-time for 200 × 10 cases.

shown in Table 4.

As could be expected, the exact ("WW") LB-problem solver produces better results than the EOQ-approximation. Still, the improvement is not dramatic – actually just around 0.3 %. However, this improvement is obtained through a significant increase in computing time. Actually, this improvement increases computing time close to 200 times ($\frac{69688.828}{352.699} \approx 198$).

To some extent, we can claim (for the case at hand at least) that 2 things are demonstrated: Our algorithms outperform standard software through CPLEX, but also and perhaps even more interesting, our simple EOQ approximation provides very good solutions very fast.

The 5000 × 5000 case was only run with the EOQ LB-problem solver. The reason ought to be obvious – a 200 times speed difference (see above) would turn 4.6 CPU hours

Iteration	Z	$CPU(s)$	GAP(%)
10	409839123	70.454	0.324 %
50	409833038	352.699	0.322 %

Tab. 3. Algorithmic performance with EOQ for the 1000 × 1000 case.

Iteration	Z	$CPU(s)$	GAP(%)
10	408516589	6957.707	0.0012 %
30	408516589	20745.197	0.0012 %
100	408516289	69688.828	0.0011 %

Tab. 4. Algorithmic performance with "WW" for the 1000×1000 case.

(16622.292 CPU-seconds) into more than a CPU-month. But, and this is important, our EOQ-based heuristic did find solutions in reasonable time. The results are given in Table 5 without further comments.

Iteration	Z	$CPU(s)$	GAP(%)
10	10449259234	1701.746	0.00338 %
30	10449259234	5081.390	0.00338 %
100	10449247810	16622.292	0.00327 %

Tab. 5. Algorithmic performance with EOQ for the 5000×5000 case.

4.5. Behaviour of our algorithm for cases violating assumption (15)

The main reason for the observed result quality is perhaps due to our choice of UB-problem solver. This solver was derived based on the assumption (15). However, there is nothing stopping us from testing our heuristic approaches on general CLSPs. After all, the arguments made in subsection 3.3 on not solving sub-problems to optimality are still valid. We did some tests (see below), consciously generating cost data breaking (15). Now, the "Bulldozer" serves as an approximation, but these tests did indeed indicate that solution quality was still at a good level.

The actual experiments involved a case with $J = 50$ and $T = 10$. The cost structure were randomly (Gaussian) generated securing that all cases violated our cost assumption. On average (10 different cases), our heuristic performed around 10 (10.4556) times faster than CPLEX. For the same execution time, our heuristic proved (on average) 17 % smaller gap than CPLEX.

As such, our algorithmic framework may be of interest as a method to solve large-scale general CLSP-problems.

5. CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

In this paper, we have demonstrated a possibility of solving really large-scale CLSP-cases under relatively minor restrictions on production costs. Recall that our assumption is limited only to this cost element – inventory and/or set-up costs are unrestricted. Additionally, as we see it, our assumption (15) should not be considered practically very restrictive. An assumption of increasing productivity over time does not (in our opinion) violate real-world expected empirical behaviour.

As opposed to other research reporting really large-scale cases, we do not reformulate the original CLSP. There may of course be very good reasons for doing so (see e. g. Haugen [10]), but we still feel that the original CLSP are of practical relevance.

There are several options to improve our algorithmic choice. For instance, choosing either "WW" or EOQ as a sub-problem solver for the LB-problems could be extended to applying a combination. An interesting strategy to investigate could be to apply EOQ initially, to acquire a relatively good solution fast, and then continue with "WW" in order to improve on this solution. We have not tested this option, though obviously feasible. Furthermore, a closer look into the possibility of more serious applications of dynamic smoothing may prove interesting. A classical feature of the type of heuristic we have applied is cycling. Previous research (see for instance [10] or [11]) indicates algorithmic sensitivity with regards to the either static or dynamic choices of smoothing parameter(s). An algorithmic scheme with the possibility of changing these parameters dynamically seems an interesting candidate to investigate further.

Finally, it seems fair to stress that we merely have demonstrated the potential of our algorithms in solving really large-scale CLSPs. Surely, it is necessary to formulate a much wider set of problem instances for testing in order to get a better feeling for behaviour of such cases. Still, we find our results both interesting as well as surprising when it comes to observed solution quality and execution speed.

A. A SHORT DESCRIPTION OF OUR EOQ APPROXIMATION

It is of course well know from classical literature (see for instance the classical text-book by Nahmias [17]) that the EOQ model may serve as an approximation in solving dynamic lot-size models. Such an approximation will improve if demand variability is moderate, and Wagner and Whitin themselves actually demonstrate the limiting behaviour of their WW-algorithm in the original paper [20].

Our approach can be described as follows:

1) Compute EOQ_j for all products $j \in \{1, \ldots, J\}$.

2) Make necessary adjustments on EOQ_j computed in 1) in order to spawn production costs.

3) $\forall\, j \in \{1, \ldots, J\}$ find set-up structures.

Point 1) above is straightforward. We compute average demand per product $\bar{d}_j = \frac{1}{T}\sum_{t=1}^{T} d_{jt}$ and compute $EOQ_j = \sqrt{\frac{2\bar{d}_j \bar{s}_j}{\bar{h}_j}}$, where $\bar{s}_j$ and $\bar{h}_j$ are average set-up and inventory costs respectively.

As our model contains production costs, point 2) indicates that we must adjust the above calculated values for EOQ_j by some factor reflecting this. Some trial and error lead to the adjustment factors for EOQ_j of the form $K\left(c_{jt} + a_{jt}\lambda_{jt}^{k}\right)$, where K is some norming constant (revealed through experiments), and λ_{jt}^{k} is the Lagrangian multiplier for product j in time period t at iteration step k.

Point 3) is then simply performed by picking the EOQ-values $\left(EOQ_j = \sqrt{\frac{2\bar{d}_j \bar{s}_j}{\bar{h}_j}}\right)$ and looping through the given demand (over t) until aggregate demand is larger than the given EOQ-value. This time period makes the next set-up period.

Table 6 highlights the quality of a such an algorithm, as it produces results quality-wise comparable to the Silver and Meal (SM) and Part Period Balancing (PPB) heuristics; more than two times faster. SM approximates the original lot-size problem by looking at average costs instead of total costs. EOQ looks at total costs, but alternatively approximates demand. PPB is a kind of intermediate approach between EOQ and SM.

The data in Table 6 are computed for 20 randomly generated data sets (one product in 20 time periods) where all costs are constant. Demand is uniformly generated on the interval $[0, 20]$.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	CPUs
SM	0	0.74	3.4	3.8	5	13	0.43
PPB	0	1.6	3.8	7	9.7	25	0.43
EOQ	0.53	1.2	2	4.9	5.1	23	0.16

Tab. 6. Quantiles, mean and median of relative errors (%) and Computation time (ms)-

B. DESCRIPTION OF LB-PROBLEMS

The LB-problems referred in subsection 3.1 may be formulated as:

$$\begin{aligned} \text{Min } Z \quad = \quad & \sum_{t=1}^{T}\sum_{j=1}^{J}[s_{jt}\delta_{jt} + h_{jt}I_{jt} + c_{jt}x_{jt}] \\ & + \sum_{t=1}^{T}\lambda_t\left(\sum_{j=1}^{J}a_{jt}x_{jt} - R_t\right) \\ \text{s.t.} \quad & \text{the constraints (3) to (8).} \end{aligned} \tag{16}$$

where all parameters and variables are defined in section 2. (16) can be reformulated as:

$$\begin{aligned} \text{Min } Z \quad = \quad & \sum_{t=1}^{T}\sum_{j=1}^{J}[s_{jt}\delta_{jt} + h_{jt}I_{jt} + \bar{c}_{jt}x_{jt}] - \sum_{t=1}^{T}\lambda_t R_t \\ \text{s.t.} \quad & \text{the constraints (3) to (8).} \end{aligned} \tag{17}$$

where $\bar{c}_{jt} = c_{jt} + \lambda_t a_{jt}$. Now, for given values of λ_t, the final part of (17) is a constant an can be removed from the objective. As a consequence, the remaining problem is identical to the original Wagner/Whitin [20] formulation.

The LB-problem is then solved either by a modern version of the original Wagner/Whitin Dynamic Programming algorithm [21] or by our EOQ-approximation described in appendix A.

C. DESCRIPTION OF UB-PROBLEMS

In this appendix we formulate and show our proposed algorithmic choices for The LB problems referred in subsection 3.1. These problems can be formulated (As Linear Programs) by equations (18) – (22) where variables and parameters are the same as defined in section 2 apart from $\hat{T}$ which is different from the original T. This difference occurs due to the problem structure where binary variables are fixed and $\hat{T}$ is hence smaller than or equal to T.

$$\text{Min } Z = \sum_{j=1}^{J} \sum_{t=1}^{\hat{T}} [h_{jt} I_{jt} + c_{jt} x_{jt}] \tag{18}$$

s.t.

$$\sum_{j=1}^{I} a_{jt} x_{jt} \leq R_t \quad \forall\, t \tag{19}$$

$$x_{jt} + I_{j,t-1} - I_{jt} = d_{jt} \quad \forall\, j, t \tag{20}$$

$$x_{jt} \geq 0 \quad \forall\, t \tag{21}$$

$$I_{jt} \geq 0 \quad \forall\, t. \tag{22}$$

An pseudo-code algorithm for the single item version of (18) – (22) is shown below:

0. **LET** $x_t^* = d_t, \forall\, t$
1. **IF** $x_t^* \leq R_t, \forall\, t$ **STOP** (x_t^* is optimal)
2. **IF** next period is $\hat{T} + 1$ **STOP**
3. **ELSE** find next period, τ where $x_t^* > R_t$ and produce a total of $x_t^* - R_t$ in previous periods $\tau-1, \tau-2, \ldots$ as close as possible to τ. (If impossible, problem is infeasible **STOP**)
4. **SET** $x_\tau^* = R_\tau$ and update $x_{\tau-1}^*, x_{\tau-2}^*, \ldots$ correspondingly
5. **GOTO** 2.

The above algorithm is extended to handle multiple items by simply choosing which product to start to produce and continue on the next product. The actual item rank is based on the ratios $\frac{c_{jt}}{a_{jt}}$ to secure cost minimization.

ACKNOWLEDGEMENT

Grants from The Norwegian Research Council (Strategisk Høgskoleprosjekt: Supply Chain Management (SCM) og optimeringsmodeller) are gratefully acknowledged.

(Received July 25, 2011)

REFERENCES

[1] G. Belvaux and L. A. Wolsey: LOTSIZELIB: A library of Models and Matrices for Lot-Sizing Problems. Internal Report, Universite Catholique de Louvain 1999.

[2] G. R. Bitran and H. H. Yanasse: Computational complexity of the capacitated lot size problem. Management Sci. *28* (1982), 1174–1186.

[3] L. Buschlkühl, F. Sahling, S. Helber, and H. Tempelmeier: Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. OR Spectrum *132* (2008), 2, 231–261.

[4] W. H. Chen and J. M. Thizy: Analysis of relaxation for the multi-item capacitated lot-sizing problem. Ann. Oper. Res. *26* (1990), 29–72.

[5] M. Diaby, H. C. Bahl, M. H. Karwan, and S. Zionts: A Lagrangean relaxation approach for very-large-scale capacitated lot-sizing. Management Sci. *38* (1992), 9, 1329–1340.

[6] C. Gicquel, M. Minoux, and Y. Dallery: Capacitated Lot Sizing Models: A Literature Review. Open Access Article hal-00255830, Hyper Articles en Ligne 2008.

[7] F. W. Harris: How many parts to make at once. Factory, the Magazine of Management *10* (1913), 2, 135–136.

[8] K. K. Haugen, A. Løkketangen, and D. Woodruff: Progressive Hedging as a meta-heuristic applied to stochastic lot-sizing. European J. Oper. Res. *132* (2001), 116–122.

[9] K. K Haugen, A. Olstad, K. Bakhrankova, and E. Van Eikenhorst: The single (and multi) item profit maximizing capacitated lot-size problem with fixed prices and no set-up. Kybernetika *47* (2010), 3, 415–422.

[10] K. K. Haugen, A. Olstad, and B. I. Pettersen: The profit maximizing capacitated lot-size (PCLSP) problem. European J. Oper. Res. *176* (2007), 165–176.

[11] K. K. Haugen, A. Olstad, and B. I. Pettersen: Solving large-scale profit maximization capacitated lot-size problems by heuristic methods. J. Math. Modelling Algorithms *6* (2007), 135–149.

[12] T. Helgasson and S. W. Wallace: Approximate scenario solutions in the progressive hedging algorithm. Ann. Oper. Res. *31* (1991), 425–444.

[13] B. Karimi, S. M. T. Fatemi Ghomi, and J. M. Wilson: The capacitated lot sizing problem: a review of models and algorithms. Omega *31* (2003), 365–378.

[14] O. Kirca and M. Kokten: A new heuristic approach for the multi-item lot sizing problem. European J. Oper. Res. *75* (1994), 2, 332–341.

[15] J. Maes, J. O. McClain, and L. N. Van Wassenhove: Multilevel capacitated lot sizing complexity and LP-based heuristics. European J. Oper. Res. *53* (1991), 2, 131–148.

[16] A. S. Manne: Programming of economic lot-sizes. Management Sci. *4* (1958), 2, 115–135.

[17] S. Nahmias: Production and Operations Analysis. Sixth edition. McGraw Hill, Boston 2009.

[18] J. M. Thizy and L. N. Van Wassenhove: Lagrangean relaxation for the multi-item capacitated lot-sizing problem: A heuristic implementation. IEE Trans. *17* (1985), 4, 308–313.

[19] W. W. Trigeiro, L. J. Thomas, and J. O. McClain: Capacitated lot sizing with setup times. Management Sci. *35* (1989), 3, 353–366.

[20] H. M. Wagner and T. M. Whitin: Dynamic version of the economic lot size model. Management Sci. *5* (1958), 3, 89–96.

[21] A. Wagelmans, S. Vanhoesel, and A. Kolen: Economic lot sizing – an $O(n \log n)$ algorithm that runs in linear time in the Wagner–Whitin case. Oper. Res. *40* (1992), 5145–5156.

Kjetil K. Haugen, Molde University College, Box 2110, 6402 Molde. Norway.
e-mail: Kjetil.Haugen@himolde.no

Guillaume Lanquepin-Chesnais, Molde University College, Box 2110, 6402 Molde. Norway.
e-mail: Guillaume.Lanquepin@hiMolde.no

Asmund Olstad, Molde University College, Box 2110, 6402 Molde. Norway.
e-mail: Asmund.Olstad@himolde.no