Parallel Genetic Algorithm for the Capacitated Lot-Sizing Problem

Knut Haase
Institut für Betriebswirtschaftslehre
University of Kiel, Germany
E-mail: haase@bwl.uni-kiel.de

Udo Kohlmorgen
Institut für Angewandte Informatik und
Formale Beschreibungsverfahren
University of Karlsruhe, Germany
E-mail: kohlmorgen@aifb.uni-karlsruhe.de

Abstract

A parallel genetic algorithm is presented to solve the well-known capacitated lot-sizing problem. The approach is implemented on a massively parallel single instruction multiple data architecture with 16384 4-bit processors. Based on a random keys representation a schedule is backward oriented obtained which enables us to apply a very simple capacity check.

1 Parallel Genetic Algorithm

Genetic algorithms are a general purpose optimization technique inspired by population genetics. The fields in which genetic algorithms are used range from operations research problems [17] and learning classifier systems[7], [11] to training neural networks [3]. For a detailed introduction to genetic algorithms see e.g. [9] or [15].

A genetic algorithm models the development of a population over a number of generations as it happens in nature. The understanding is that in nature the fitter an individual is the better is its chance to survive and the more and better offspring it will create. Therefore, the performance of the population may improve in every generation.

Generally speaking a genetic algorithm operates on a set of individuals, called population. Each individual represents a solution to a given problem. In our case an individual represents a solution to the capacitated lot-sizing problem. The individual is represented as a chromosome which consists of genes. Here each gene contains a real value. The performance (often referred to as 'fitness') of an individual is its phenotypic value with respect to an objective function which is to be optimized. For the capacitated lot-sizing problem an individual's performance is determined by the cost that the schedule produces. Genetic operators (e.g. mutation and crossover) are used to create new individuals. Due to the selection of good individuals as parents for the next generation the average performance of the population is expected to increase.

There exist various strategies for the selection process. A simple strategy for selecting parents is to choose individuals with a probability proportional to their performance.

Those individuals which are chosen as new parents are then subject to crossover and mutation. The crossover operator combines two parents and produces one or two new individuals. Therefore a genetic algorithm has the following overall structure:

```
genetic algorithm
randomly generate initial population
evaluate each individual of the population
REPEAT
    select parents
    use crossover to create offspring
    mutate offspring
    evaluate offspring
UNTIL termination criterion satisfied
```

Since genetic algorithms are inherently parallel we are using a fine-grained parallel computer MasPar MP 1216. 16k processors are placed on a two dimensional grid with toroidal connections. Figure 1 shows this array with the connections to the 8 neighbors for each processor. The toroidal connections are not drawn in this figure. We implemented the neighborhood model ([14] and [16]) on this computer. In this model each processor holds one individual of the population. In the selection process for each individual a mating partner is chosen from one of its 8 direct neighbors, only. So there is no need for global communication. Recombination is done by 2 point crossover, and mutation is standard.

This model avoids premature convergence [14].

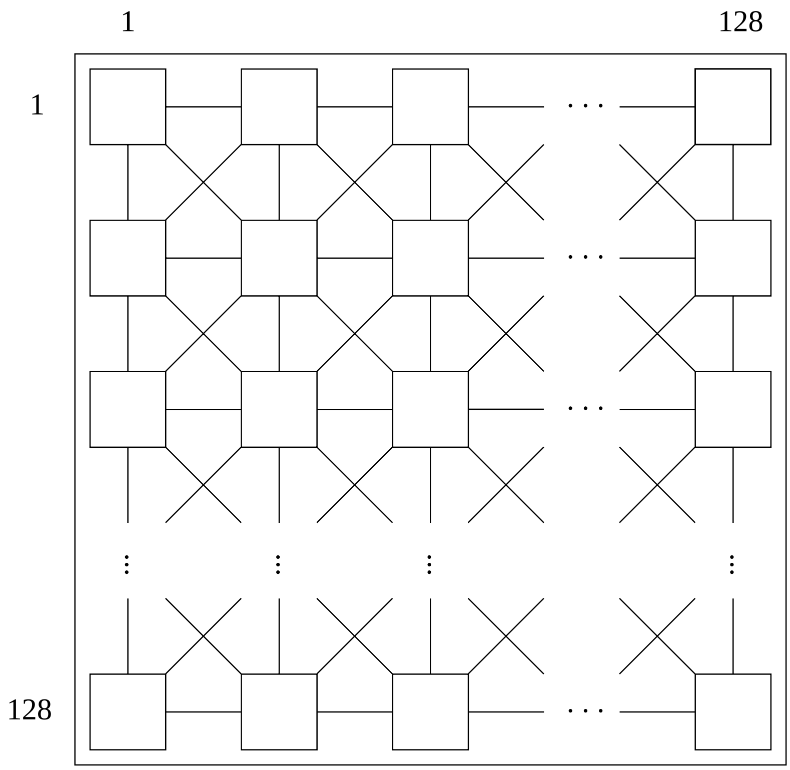

Figure 1: two dimensional 128 $\times$ 128 processor array

The selection process to determine the new parent1 chooses among the old parent1 and the offspring child1 and child2. The best individual of these three is chosen with a probability of 55%. With a probability of 15% for each individual those are chosen as parent1. Therefore the best individual has an overall probability of 70% to be selected as new parent1.

<u>parallel genetic algorithm (neighborhood model)</u>

randomly generate initial population such that each processor contains one individual (called parent1)
evaluate each individual of the population
REPEAT
  select an individual of the neighborhood as parent2
  use crossover to create offspring (child1 and child2) from parent1 and parent2
  mutate offspring
  evaluate offspring
  replace parent1 with one out of parent1, child1, and child2
UNTIL termination criterion satisfied

For a more detailed description of the parallel genetic algorithm see [4].

2 The Capacitated Lot-Sizing Problem (CLSP)

The capacitated lot-sizing problem (CLSP) is characterized as follows: A number of J different items is to be manufactured on one machine (corresponding to a single capacity constraint). The planning horizon is segmented into a finite number of T periods. In period $t \in \{1, ..., T\}$ the machine is available with C_t capacity units. Producing one unit of item j requires $p_j > 0$ capacity units. The demand for item j in period t, $d_{jt} \geq 0$, has to be satisfied without delay. Setting up the machine for item j causes setup cost $s_j > 0$. Setup costs occur for each lot produced in a period *(basic assumption)*. Holding cost $h_j \geq 0$ is incurred for the inventory of item j at the end of a period. The objective is to minimize the costs for setups and holding.

Defining the decision variables

I_{jt} the inventory of item j at the end of period t ($I_{j0} = 0 \ \forall j$)

q_{jt} the quantity (lot-size) of item j to be produced in period t

x_{jt} a binary variable indicating whether a setup occurs for item j in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$)

we can state the CLSP as follows:

$$Minimize \sum_{j=1}^{J} \sum_{t=1}^{T} (s_j x_{jt} + h_j I_{jt}) \tag{1}$$

subject to

$$
\begin{aligned}
I_{j,t-1} + q_{jt} - I_{jt} &= d_{jt} && \forall j, t && (2)\\
\sum_{j=1}^{J} p_j q_{jt} &\leq C_t && \forall t && (3)\\
C_t x_{jt} - p_j q_{jt} &\geq 0 && \forall j, t && (4)\\
I_{jt}, q_{jt} &\geq 0 && \forall j, t && (5)\\
x_{jt} &\in \{0, 1\} && \forall j, t && (6)
\end{aligned}
$$

The objective function (1) counts the costs for the setups and the holding of the items. (2) are the inventory balances. Constraints (3) make sure that the total production in each period does not exceed the capacity. For each lot (4) forces a setup, i.e. the corresponding binary setup variables must be one, thus increasing the sum of setup cost. The last two constraints (5) and (6) properly define the domains of the continuous and binary variables, respectively.

In the literature for the CLSP a multitude of heuristics (cf. [6], [12], [13], [19]) and exact methods (cf.[1], [8]) have been proposed.

Now, let us consider the following example.

Example 1: Let $J = 3$, $T = 4$, $(h_j) = (1\ 1\ 1)$, $(s_j) = (100\ 300\ 200)$, $(p_j) = (1\ 1\ 1)$, $(C_t) = (100\ 100\ 100\ 100)$, and

$$(d_{jt}) = \begin{pmatrix} 30 & 20 & 40 & 50 \\ 10 & 10 & 20 & 10 \\ 20 & 50 & 50 & 70 \end{pmatrix}$$

We determine an optimal solution of the CLSP with the standard solver LINDO [18]. The corresponding objective function value is $Z^* = 1610$ and the lot-sizes are as follows:

$$(q_{jt}) = \begin{pmatrix} 50 & 0 & 60 & 30 \\ 20 & 0 & 30 & 0 \\ 20 & 100 & 0 & 70 \end{pmatrix}$$

We see that in the optimal solution a splitting occurs for the demand $d_{14} = 50$, i.e. a fraction of 20 units is produced in period $t = 3$ which is included in the lot-size $q_{13} = 60$. Thus it is important to provide a demand splitting in approaches for solving the CLSP. In the following section we introduce a new heuristic which allows one (additional) demand splitting per period.

3 Genetic Representation of a CLSP Schedule

A CLSP solution is computed as follows: The CLSP schedule is generated backward oriented. We compute the lot-sizes in period $t = T$, we then go back to period $t = T - 1$, and so on, until the production decisions are made for period $t = 1$. In every backward step from a period t to period $t - 1$ a feasiblity check is performed. The selection of items to be produced in a period t depends on item and period specific random keys [2] .

Initially all q_{jt} ($j = 1, \ldots, J$, $t = 1, \ldots, T$) are set to 0. Consider now a period $t, 1 \leq t \leq T$, where we have already made production decisions for the periods $\tau = t + 1 \ldots T$ by fixing $q_{j\tau}$ for $j = 1 \ldots J$. Then the remaining cumulative demand for item j from period t to the horizon T which has to be satisfied in the periods $t, \ldots, 1$ is defined by

$$D_{jt} = \sum_{\tau=t}^{T} (d_{j\tau} - q_{j\tau})$$

The remaining total required capacity is specified by

$$TRC = \sum_{j=1}^{J} p_j D_{j1}$$

Furthermore, the still available capacity in period t will be computed as follows:

$$AC_t = C_t - \sum_{j=1}^{J} p_j q_{jt}$$

Outline of the scheduling algorithm

$TRC := \sum_{j,t} p_j d_{jt}$ $\forall j\, D_{jT} := d_{jT}$
FOR $t = T$ DOWNTO 1 DO
    $AC_t = C_t$
    $i = \lceil \psi_t J \rceil$
    $k = 0$
    WHILE $k \leq J - 1$ AND $AC_t > 0$
        $j := (i + k - 1) \bmod J + 1$
        IF $\alpha_{jt} \geq \theta_t$ AND $AC_t > p_j D_{jt}$ THEN
            $q_{jt} := D_{jt}$
            $D_{jt} := 0$
            $TRC := TRC - p_j q_{jt}$
            $AC_t := AC_t - p_j q_{jt}$
            $k = k + 1$
    WHILE $k \leq J - 1$ AND $CC_{t-1} < TRC$
        $j := (i + k - 1) \bmod J + 1$
        $q_{jt} := min\{D_{jt}, AC_t/p_j\}$
        $D_{jt} := D_{jt} - q_{jt}$
        $TRC := TRC - p_j q_{jt}$
        $AC_t := AC_t - p_j q_{jt}$
        $k = k + 1$
    FOR $j = 1$ TO J DO
        $D_{j,t-1} := D_{jt} + d_{j,t-1}$

Finally, we denote the cumulative capacity from period $\tau = 1$ to period $\tau = t$ by

$$CC_t = \sum_{\tau=1}^{t} C_\tau$$

Now consider the vector $v_t = (\alpha_{1t}, \ldots, \alpha_{Jt}, \theta_t, \psi_t) \in (0,1)^{J+2}$ where θ_t is a threshold value, ψ_t is used to determine the first item which is scheduled in period t, and $\alpha_{1t}, \ldots, \alpha_{Jt}$ are preference values of the items $j = 1, \ldots, J$. If the preference value $\alpha_{jt} \geq \theta_t$ and $AC_t \geq p_j D_{jt}$ then item j will be scheduled in period t with $q_{jt} = D_{jt}$. Thus, due to the capacity restriction, an item j with $\alpha_{jt} \geq \theta_t$ may not be scheduled in period t. Moreover, which items are scheduled

Table 1: Genetic representation

α_{11}	α_{21}	α_{31}	θ_1	ψ_1	α_{12}	α_{22}	α_{32}	θ_2	ψ_2	α_{13}	α_{23}	α_{33}	θ_3	ψ_3	α_{14}	α_{24}	α_{34}	θ_4	ψ_4
.2	.3	.4	.8	.6	.7	.8	.4	.1	.7	.5	.6	.3	.4	.3	.9	.2	.8	.7	.4

Table 2: Computational report

t	i	j	(d_{1t}, d_{2t}, d_{3t})	AC_t	q_{jt}	CC_{t-1}	TRC
4	2		(50,10,70)	100		300	380
		3		30	70		310
		1		0	30		280
3	1		(60,30,50)	100		200	
		1		40	60		220
		2		10	30		190
2	3		(20,10,100)	100		100	
		3		0	100		90
1	2		(50,20,20)	100		0	
		2		80	20		70
		3		60	20		50
		1		10	50		0

in a period depends also on the sequence in which we try to schedule the items. We consider the items in the following sequence

$$SEQ_t = (i,\ i+1, \ldots,\ J,\ 1, \ldots,\ i-1)$$

where $i = \lceil \psi_t J \rceil$.

Note, if all items j with $\alpha_{jt} \geq \theta_t$ are scheduled in period t and $TRC > CC_{t-1}$ then we have to schedule more items in period t until $TRC \leq CC_{t-1}$ (capacity check, cf. [10]). This will be done according to the sequence SEQ_t. Furthermore, if demand splitting is required due to capacity restrictions, it will only be performed in this second part of the algorithm. A more detailed description of the algorithm is given in the outline of the scheduling algorithm (see above).

Table 1 provides one genetic representation for the optimal solution of Example 1. The corresponding computation of the optimal solution is reported in Table 2.

4 Computational Results

The genetic representation which we use to derive a solution corresponds to the random keys described above. Each random key is assigned to one gen. The number of gens totals $T \times (J+2)$. This corresponds to a floating point representation. Our parallel genetic algorithm optimizes the parameters to compute solutions for the CLSP. We employed the well-known 120 benchmark-instances from [5] where T as well as J range from 8 to 50. Our computational study shows that the results obtained by the parallel genetic algorithm has the same solution quality as the state of the art algorithm from [12], which outperforms the heuristics of [6] and [19]. The detailed results are shown in Table 3. Z^* denotes the best result obtained by the three algorithms. The results indicate that our parallel genetic algorithms is superior for problems with 50 items, 8 periods and slightly better for problems with 8 items, 50 periods. For problem with 20 items, 20 periods the algorithms from [12] gets better results. But our results in this category are on the average only 1.44% higher than the ones obtained by [12]. Table 4 shows the number of problems for which each algorithm found the best result of all three algorithms.

Table 3: Computational results

	Dixon-Silver	Kirca-Kökten	parallel GA
50 items, 8 periods	1.29	0.65	0.17
20 items, 20 periods	7.55	0.06	1.50
8 items, 50 periods	9.57	0.99	0.76
total average	6.14	0.57	0.81

average % deviation from the best solution found by DS, KK or PGA

Overall our studies show that a parallel genetic algorithm is capable of solving the capacitated lot-sizing problem as good as or better than the best special

Table 4: Number of best results

	Dixon-Silver	Kirca-Kökten	parallel GA
50 items, 8 periods	4	12	24
20 items, 20 periods	0	37	3
8 items, 50 periods	0	19	21
total number	4	68	48

heuristics known so far. The computation of a schedule from the genetic representation is very easy and does not require too much knowledge about the capacitated lot-sizing problem itself. The genetic representaion we use has some advantages. First, optimizing the parameters α_{jt}, θ_t, and ψ_t instead of the actual lot-sizes q_{jt} avoids the implementation of a special crossover strategy. We can apply the normal one-point, two-point, multi-point, and uniform crossover strategies and we always get feasible solutions. Second by not generating non feasible solutions, we avoid enlarging the search space. A setback is that we could not prove that the optimal solution is always a member of our new search space. But as it can be seen, the algorithm still found very good solutions. Another disadvantage of our parallel genetic algorithm is the time required to compute the solutions. On the SIMD machine MasPar MP 1216 with 16k processors it takes about ten minutes before the genetic algorithm terminates, while other algorithms need about one second on a PC to compute their heuristic solution. A huge advantage of our parallel genetic algorithm is it's flexibility. We have shown in [4] that our parallel genetic algorithm is easily adapted to a slightly different problem, and the solutions are even better compared to other algorithms whereas specialized algorithms do not adapt very well.

References

[1] I. Barany, T.J. van Roy, and L.A. Wolsey. Strong formulations for multi-item capacitated lot-sizing. *Management Science*, 30:1255–1261, 1984.

[2] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154-160, 1994.

[3] J. Branke. Evolutionary Algorithms for Neural Network Design and Training. In J.T. Alander, editor, *First Nordic Workshop on Genetic Algorithms*, Vaasa, Finnland, 1995.

[4] J. Branke, U. Kohlmorgen, H. Schmeck, and H. Veith. Steuerung einer Heuristik zur Losgrößenplanung unter Kapazitätsrestriktionen mit Hilfe eines parallelen genetischen Algorithmus. In J. Kuhl and V. Nissen, editors, *Tagungsband zum Workshop Evolutionäre Algorithmen in Management-Anwendungen*, Göttingen, 1995.

[5] D. Cattrysse, J. Maes, and L.N. van Wassenhove. Set partitioning and column generation heuristics for capacitated lotsizing. *European Journal of Operational Research*, 46:38–47, 1990.

[6] P.S. Dixon and E.A. Silver. A heuristic solution procedure for multi-item single-level, limited capacity, lot-sizing problem. *Journal of Operations Management*, 23–39, 1981.

[7] P.W. Frey and D.J. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6,2:161–182, 1991.

[8] L.F. Gelders, J. Maes, and L.N. van Wassenhove. A branch and bound algorithm for the multi-item single level capacitated dynamic lot-sizing problem. In S. Axsäter and et al., editors, *Multi-stage production planning and inventory control*, pages 92–108. Springer Berlin, 1986.

[9] D.E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1992.

[10] K. Haase. Capacitated lot-sizing with sequence dependent setup costs. Technical Report 340, Institut für Betriebswirtschaftslehre, University of Kiel, Germany, 1994. *OR Spektrum (to appear).*

[11] J.H. Holland. Adaptation in natural and artificial systems. Ann Arbor: The University of Michigan Press, 1975.

[12] Ö. Kirca and M. Kökten. A new heuristic approach for the mult-item dynamic lot sizing problem. *European Journal of Operational Research*, 75:332–341, 1994.

[13] M.R. Lambrecht and H. Vanderveken. Heuristic procedures for the single operations, multi-item loading problem. *AIIE Transactions*, 11:319–326, 1979.

[14] B. Manderick and P. Spiessens. Fine-Grained Parallel Genetic Algorithms. In J.D. Schaffer, editor, *Third International Conference on Genetic Algorithms* , pages 428–433. Morgan Kaufmann, San Mateo, 1989.

[15] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, 1994.

[16] H. Mühlenbein. Parallel Genetic Algorithms in Combinatorial Optimization. *Computer Science and Operations Research*, pages 441–453, Pergamon Press 1992.

[17] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of genetic algortihms*, pages 316–337, Morgan Kaufmann, 1991.

[18] L. Schrage. *Linear integer and quadratic programming with LINDO*, third edition, 1986.

[19] J.M. Thizy and L.N. Van Wassenhove. Lagrangean relaxation for the multi-item capacitated lot-sizing problem: A heuristic implementation. *IIE Transactions*, 17:308–313, 1985.