Katarzyna GDOWSKA[*], Roger KSIĄŻEK[*]

THE STRUCTURE OF THE GENETIC ALGORITHM OF LOT-SIZING AND SCHEDULING PROBLEM FORMULATED AS CAPACITATED LOT SIZING PROBLEM

Abstract

In this paper the structure of the genetic algorithm utilised for solving an integer programming model of lot-sizing and scheduling problem is introduced. The genetic algorithm presented in the paper was employed for solving a lot-sizing and scheduling problem formulated as Capacitated Lot Sizing Problem. The method of chromosome encoding, utilised crossover operators and mutation operators employed in this genetic algorithm are presented and explained, moreover, implemented modifications are indicated.

Keywords: genetic algorithm, CLSP, lot-sizing and scheduling, integer programming

1. INTRODUCTION

The objective of the set of lot-sizing and scheduling problems is to determine minimal costs as their solution. These costs are connected to inventoriable costs (high inventory level, high costs) and costs of lot organising (small-size lots, frequent starts, high costs). In existing formulations of the problems of this type a solution is being searched in the space between these contradictory goals.

A list of selected works where lot-sizing and scheduling problem is presented is to be found, amongst others, in following works: (Karimi et al., 2003) [5], (Quadt, Kuhn, 2008) [9].

The mathematical model for Capacitated Lot Sizing Problem (CLSP) is considered to be the very first and fundamental formulation of lot-sizing and scheduling problem. The most frequent situation of CLPS utilisation is a problem with long planning horizons. In such a problem changeover costs and changeover time are charged in any case when a machine's status is ready-to-manufacture. The set of parameters and variables of the CLSP formulation is to be found in Table 1.

A PLCM for CLSP may be formulated according to mathematical description presented in Table 2.

2. GENETIC ALGORITHM

In the paper (Książek, 2011) [7] results of a computational experiment are presented; the experiment consisted of solving a particular problem for the same set of data at first with an implemented genetic algorithm and then solving the PLCM formulation of the problem with GLPK solver (GNU Linear Programming Kit). The genetic algorithm was implemented circuit the language JAVA SE 6. On the other hand, GLPK is a free programme kit for solving integer-programming models.

[*] AGH University of Science and Technology, Faculty of Management, Department of Operation Research and Information Technology

Table 1. Parameters and variables of CLSP formulation

T	=	$\{1,\ldots,T\}$	–	Set of periods, where T is the number of periods
N	=	$\{1,\ldots,N\}$	–	Set of products, where N is the number of products
C_t			–	Manufacturing capacity, the length of period t
ST_j	$\leq$	C_t	–	Changeover time to product j
SC_j			–	Changeover cost to product j
p_j			–	Unit manufacturing time for product j
h_j			–	Unit inventoriable cost for product j
d_{jt}			–	Demand for product j in period t
x_{jt}			–	Production of product j in period t
y_{jt}	=	1	–	If in period t a machine is ready to manufacture product j
		0	–	others
I_{jt}			–	Inventory of product j in the end of period t

Table 2. Mathematical model for CLSP

$$\min \sum_{t\in T}\sum_{j\in N}(h_{jt}I_{jt} + SC_j y_{jt}) \tag{1}$$

$$I_{j,t-1} + x_{jt} - d_{jt} = I_{jt} \quad t\epsilon T, j\epsilon N \tag{2}$$

$$p_j x_{jt} + ST_j y_{jt} \leq C_t y_{jt} \quad t\epsilon T, j\epsilon N \tag{3}$$

$$\sum_{j\in N}(p_j x_{jt} + ST_j y_{jt}) \leq C_t \quad t\epsilon T \tag{4}$$

$$x_{jt}, I_{jt} \geq 0 \quad t\epsilon T, j\epsilon N \tag{5}$$

$$y_{jt} \in \{0,1\} \quad t\epsilon T, j\epsilon N \tag{6}$$

Computational experiments revealed that utilisation of the genetic algorithm for lot-sizing and scheduling problems seems to be reasonable when it comes to huge problems where finding a solution with integer programming methods is extremely time consuming. It is possible that a properly constructed and adjusted genetic algorithm may be efficient in finding out far better results than the ones achieved with integer programming methods; as far as the results received in the same amount of time are concerned.

THE ALGORITHM

At the beginning a feasible solution for a given CLSP problem is determined with a simple algorithm of filling up the production of x_{jt} connected to demand for the product d_{jt}, when moving in the planning horizon from the last planning period towards the beginning period. The condition (4) is taken into consideration and it guarantees that the total loading coming out of total production in period t will not exceed the limit for this period C_t.

Subsequently, an initial population of chromosomes is constructed based on the found beginning solution. Each chromosome provides information on the volume of production x_{jt}, so that a single chromosome is a particular feasible solution of CLSP problem.

Afterwards the initial population is subjected to following operations: *selection*, *crossover* and *mutation*. Exclusively during crossover and mutation operations it is allowed to keep within chromosomes an unfeasible solution which breaking conditions of the model.

At the last stage of the full cycle a population's life the chromosomes are repaired in order to eliminate solutions which do not fulfil conditions of the model. In order to repair chromosomes a chromosome fitness function and comparison function are utilised. The latter

improve the solution contained in a chromosome with respect to the objective function's value. For this purpose three functions improving a chromosome which does not satisfy expectations and additional two functions improving a chromosome with regard to the objective function are utilised.

REPRESENTATION

In CLSP model three variables (Table 1) are to be indicated and the outcome of the model depends on them. Moreover, the variables are connected with each other by mathematical relations (Table 2) that cause each variable determine other variables' values.

When a problem has its feasible solution, values of one variable predominate over values of the other two variables. The chromosome representing the solution contains information about variable x_{jt}, i.e. production of product j in period t. Other variables I_{jt} and y_{jt} are computed basing on fixed values of variable x_{jt}. Each chromosomes ch_n contains the string of numbers:

$$ch_n = \{(x_{1,1}),(x_{1,t+1}),...,(x_{1,T}),(x_{j+1,1})(x_{j+1,t+1}),...,(x_{j+1,T}),...,(x_{N,T})\} \quad (7)$$

Each value of variable x_{jt} is written inside the chromosome with a binary number. It result in obtaining a chromosome, which contains the string of two values "1" and "0". The number of bytes b that are necessary to encode the value of a single variable of production volume x_{jt} was estimated basing on the number of bytes that are necessary to encode maximal value of manufacturing capacity in the particular period C_t.

CROSSOVER OPERATOR AND MUTATION OPERATOR

A standard one-point crossover operator [2] was utilised in the structure of the genetic algorithm employed for this problem. Utilisation of k-point crossover operator [2] for this genetic algorithm was taken into consideration at initial stages of constructing this algorithm. However, this operator did not have any significant contribution to improvement of the received results. Therefore less complicated, 1-point crossover operator was chosen. Crossover operation conducted according to this method requires fulfilling of the procedure presented in the Scheme 1.

Scheme 1. Procedure of chromosome crossover operation, (Gwiazda, 2007) [2].

1. Select a pair of parent chromosomes $A^{(t)}$ and $B^{(t)}$ from the set of parents,
2. Create a pair of child chromosomes $C^{(t+1)}$ and $D^{(t+1)}$ in the following way:
3. Select randomly a crossover point $cp \in \{1, \ldots, n\text{-}1\}$,
4. for $i = 1$ to cp do
5. $c_i^{(t+1)} = a_i^t$
6. $d_i^{(t+1)} = b_i^t$
7. end do
8. for $i = cp + 1$ to n do
9. $c_i^{(t+1)} = b_i^t$
10. $d_i^{(t+1)} = a_i^t$
11. end do

The mutation operation is conducted with a standard mutation operator [3]. Particular values of the variable x_{jt} are selected according to the probability of mutation then they are replaced with random values which can range from 0 till the maximal number of products that

we are able to manufacture in this period. The mutated value of x_{jt} and the rest of planned for the period t production of other products are not included.

FUNCTIONS UTILISED FOR IMPROVING A CHROMOSOME WHICH DOES NOT SATISFY EXPECTATIONS

Functions of this group guarantee that a chromosome which they were performed over will contain a solution feasible with regard to conditions of CLSP for planning and lot-sizing problem.

The first function for improving a chromosome which does not satisfy condition constraints of the problem is a function that eliminates production C^{+}_{t} exceeding manufacturing capacity C_t for period t:

$$C_t^+ = \begin{cases} \sum_{i \in N} x_i - C_t & \sum_{i \in N} x_i > C_t \\ 0 & \sum_{i \in N} x_i \le C_t \end{cases} \tag{8}$$

For each period t a product j is randomised, afterwards value of production x_{jt} is set to be equal 0. Subsequent products j are being drawn until $C^{+}_{t} = 0$, so that the load does not exceed the manufacturing capacity C_t in a selected period t. Certainly, the utilised function may result in the fact that in some periods t demand will not be satisfied. Such a situation is being improved with use of other functions.

The second function utilised for improving the solution contained in a chromosome is a function which eliminates unnecessary inventory. If the inventory volume I_{jT} of product j in the last planning period T is greater then 0, it means that production of product j planned for the planning horizon exceeds demand for the production volume which is an equal inventory for the last period I_{jT}. In order to eliminate the surplus of manufactured products we reduce production x_{jt} for a particular product j beginning with the last period T and we direct towards the period number 1. The way we perform it guarantees that the condition (2) is not broken, which means that demand d_j for product j has to be satisfied.

The third function performs in the opposite way to the ones just presented. It replenishes production of product j in particular periods with missing products in order to eliminate negative inventory. The function performs similarly to the method utilised by the second function for satisfying the condition (2) of the model for CLSP, which means that demand d_j for product j has to be satisfied.

FUNCTIONS UTILISED FOR IMPROVING A CHROMOSOME WITH REGARD TO THE OBJECTIVE FUNCTION

Thanks to these functions it is possible to receive in a considerably short time results that are similar to the basis solution computed with a genetic algorithm, but the value of the objective function is better.

The first function utilised for improving the solution of the problem was a function consolidating production parties. As an illustration let us analyse a situation: in the period t we manufacture 10 pieces of product j and in the period t+1 – 30 pieces. In utilised model we bear the expenses of two changeovers. If in the period t we have free manufacturing capacities C_t so when we transfer from the period $t + 1$ planned production of 30 pieces of product j, we are able to avoid bearing the expenses of one changeover. In a situation when we decided to transfer production from the period t to the period $t + 1$, the solution might have become significantly worse. Therefore such operations are not taken into consideration since there is a

possibility that unsatisfied demand appears in the period *t* as a result of reduction of production in this period. When we transmit production to the previous period the only cost we generate are the inventoriable costs of pieces of product *j* from the period *t* till the period *t* + 1. Hence, in a situation when we still have free manufacturing powers and inventoriable costs for product *j* we will bear are lower than the changeover cost SC_j we consolidate production parties accordingly to the method presented above.

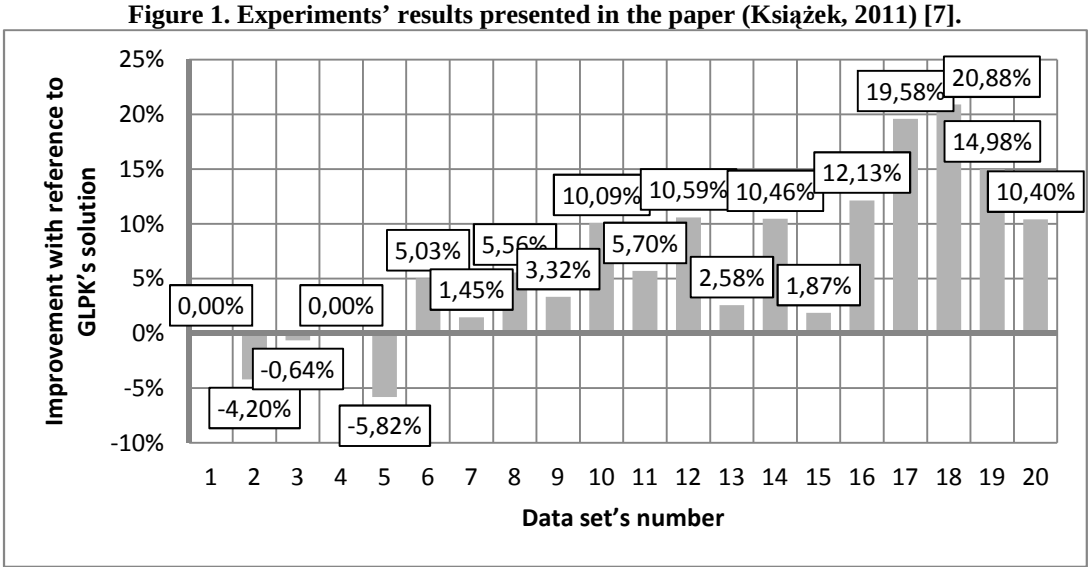
The first function utilised for improving the solution of the problem is similar to the one presented right above. In this situation we attempt to eliminate unnecessary inventorial costs h_j connected with the product *j*. As an illustration lets us analyse a situation which is similar to the previous example: we produce 10 pieces of the product *j* in the period *t* and in the period *t* + 1 – 30 pieces. If at the end of the period *t* in the 10 pieces of the product *j* are present in a warehouse, what means that demand for the product *j* has been fully satisfied, we should make sure whether there is a possibility to manufacture these 10 pieces in a subsequent period so that we will be able to avoid inventoriable costs. If in the period *t* + 1 the manufacturing capacity C_{t+1} make it possible to manufacture additional 10 pieces of the product *j*, therefore we transfer a production party of this product to the period *t* + 1 and we are able to avoid inventoriable costs for the product *j* in the period *t* + 1.

3. CONCLUDING REMARKS

Results received with a genetic algorithm constructed according to presented principles were demonstrated in the paper (Książek, 2011) [7].

It is reasonable to utilise a genetic algorithm in solving big lot-sizing and scheduling problems, since solving then with integer-programming methods is time consummating (computations were performed with the solver GLPK [GNU Linear Programming Kit]). Computational experiments presented in the paper (Książek, 2011) [7] prove that a properly adjusted genetic algorithm is able to find effectively better results than the ones obtained with the integer programming method in the same amount of time – see Figure 1: Experiments' results presented in the paper (Książek, 2011) [7]).

Figure 1. Experiments' results presented in the paper (Książek, 2011) [7].



It is possible to receive considerably well results when parameters of the genetic algorithm are properly selected. It also benefited from two groups of functions that were utilised in the structure of the algorithm: the functions utilised for improving a chromosome which does not satisfy expectations and the functions utilised for improving a chromosome with regard to the objective function. Without the mentioned functions the genetic algorithm was not able to be successful when compared to integer programming. It should be emphasised that better results were received just thanks to utilisation of these functions, especially the functions of the second group, since they organise the set of algorithm's searches and direct the algorithm towards searching amongst the best solutions for a given population.

The genetic algorithm presented in this paper and results received with it, demonstrated that this method may be useful for CLSP for lot-sizing and scheduling for a production party of a greater size. A small disadvantage of computing results with the genetic algorithm may certainly result from a complicated algorithm's structure and significant influence of its parameters on the quality of received solutions.

REFERENCES

[1] Dellaert N., Jeunet J., Jonard N., *A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs*. "International Journal of Production Economics", No 68, 2000, p. 241–257.
[2] Gwiazda T. D., *Algorytmy genetyczne*. *Kompendium*. *Tom I*. *Operator krzyżowania dla problemów numerycznych*. Wydawnictwo Naukowe PWN, Warszawa 2007.
[3] Gwiazda T. D., *Algorytmy genetyczne*. *Kompendium*. *Tom II*. *Operator mutacji dla problemów numerycznych*. Wydawnictwo Naukowe PWN, Warszawa 2007.
[4] Jinxing Xie, Jiefang Dong, *Heuristic genetic algorithm for general* C*apacitated lot-sizing problems*. "Computer & Mathematics with Applications" No 44, 2002, p. 263–276.
[5] Karimi B., Fatemi Ghomi S., Wilson J., *The capacitated lot sizing problem: a review of models and algorithms*. "OMEGA – The International Journal of Management Science" No 31, 2003, p. 365–378.
[6] Kimss A, *A genetic algorithm for multi-level, multi-machine lot sizing and scheduling.* "Computers & Operations Research", No 26, 1999, p. 829–848.
[7] Książek R., *Przedstawianie wyników działania algorytmu genetycznego dla zadania CLSP planowania wielkości i szeregowania partii produkcyjnej*. „Automatyka", No 15, Issue 2, Wydawnictwo AGH, Kraków 2011.
[8] Michalewicz Z., *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictwo Naukowo-Techniczne, Warszawa 1996.
[9] Quadt D., Kuhn H., C*apacitated lot-sizing with extentions*: *a review*. "Operations Research" No 6, 2008, p. 61–83.

BUDOWA ALGORYTMU GENETYCZNEGO DLA ZADANIA CLSP PLANOWANIA WIELKOŚCI I SZEREGOWANIA PARTII PRODUKCYJNEJ

Streszczenie

Poniższa praca przedstawia budowę algorytmu genetycznego zastosowanego do rozwiązania zadania programowania całkowitoliczbowego dla problemu planowania wielkości i szeregowania partii produkcyjnej. Opisany algorytm genetyczny posłużył do rozwiązania problemu CLSP planowania wielkości i szeregowania partii produkcyjnej. W pracy przedstawiono i wyjaśniono sposób kodowania chromosomów, użyte operatory krzyżowania i mutacji oraz wprowadzone modyfikacje.

Słowa kluczowe: algorytm genetyczny, model CLSP, planowanie wielkości i szeregowanie partii produkcyjnej