www.czasopisma.pan.pl www.journals.pan.pl

ARCHIVES
of
FOUNDRY ENGINEERING

DOI: 10.2478/afe-2013-0065

Published quarterly as the organ of the Foundry Commission of the Polish Academy of Sciences

ISSN (2299-2944)
Volume 13
Issue 3/2013

84 – 87

Production Scheduling for the Furnace - Casting Line System

A. Stawowy *, J. Duda
AGH University of Science and Technology, Faculty of Management, Gramatyka 10, 30-067 Krakow, Poland
*Corresponding author. E-mail address: astawowy@zarz.agh.edu.pl

Received 28.05.2013; accepted in revised form 31.05.2013

Abstract

The problem considered in the paper is motivated by production planning in a foundry equipped with the furnace and casting line, which provides a variety of castings in various grades of cast iron/steel for a large number of customers. The quantity of molten metal does not exceed the capacity of the furnace, the load is a particular type of metal from which the products are made in the automatic casting lines. The goal is to create the order of the melted metal loads to prevent delays in delivery of goods to customers. This problem is generally considered as a lot-sizing and scheduling problem. The paper describes two computational intelligence algorithms for simultaneous grouping and scheduling tasks and presents the results achieved by these algorithms for example test problems.

Keywords: Application of information technology to the foundry industry, Production planning, Scheduling

1. Introduction

Scheduling is an important management activity within a plant. Finding a good feasible schedule by which costs and lead times can be reduced, is often a very complex and difficult task. Complex and comprehensive character of the modeled objects requires development of flexible tools which would be useful for solving the considered problem.

There are only a few studies reported on models and algorithms for production planning and scheduling in foundries. The current state-of-the art in planning and scheduling research in foundry production systems can be found in [1]. Assuming that a production bottleneck is the furnace, a mixed-integer programming (MIP) models are usually proposed to determine the lot size of the items and the required alloys to be produced during each period of the finite planning horizon that is subdivided into smaller periods.

Four approaches are used as the solving tools:

1. discrete event simulation,
2. specialized heuristics, mostly computational intelligence (CI) algorithms,
3. commercial solvers for problems of small size,
4. commercial solvers combined with heuristics reducing space of feasible solutions.

The aim of this paper is to present the effective CI heuristics for production planning and scheduling in the single furnace-single casting line system. Section 2 provides a MIP model for foundry scheduling problem. In Section 3, the details of proposed heuristics are given. The computational experiments are described in Section 4, and the conclusions are drawn in Section 5.

2. Lot-sizing and scheduling model

The MIP model presented in this section is an extension of Araujo et al. lot sizing and scheduling model for automated foundry [2]. We use the following notation:
Indices
i=1,…,I - produced items; k=1,…,K - produced alloys
t=1,…,T - working days; n=1,…,N - sub-periods
Parameters
d_{it} - demand for item i in day t; w_i - weight of item i

$a_i^k = 1$, if item i is produced from alloy k, otherwise 0
st_k - setup cost for alloy k; C - loading capacity of the furnace
h_{it}^-, h_{it}^+ - penalty for delaying (–) and storing (+) production of item i in day t
Variables
I_{it}^-, I_{it}^+ - number of items i delayed (–) and stored (+) at the end of day t
$z_n^k = 1$, if there is a setup (resulting from a change) of alloy k in sub-period n, otherwise 0
$y_n^k = 1$, if alloy k is produced in n in sub-period, otherwise 0
x_{in} - number of items i produced in sub-period n

Production planning problem is defined as follows:

$$\text{Minimize}\quad \sum_{i=1}^{I}\sum_{t=1}^{T}(h_{it}^{-}I_{it}^{-}+h_{it}^{+}I_{it}^{+})+\sum_{k=1}^{K}\sum_{n=1}^{N}(st_k z_n^k) \qquad (1)$$

subject to:

$$I_{i,t-1}^{+}-I_{i,t-1}^{-}+\sum_{n=1}^{N}\sum_{k=1}^{K}x_{in}a_i^k-I_{it}^{+}+I_{it}^{-}\ge d_{it},\quad i=1,\dots,I,\, t=1,\dots,T \qquad (2)$$

$$\sum_{i=1}^{I} w_i x_{in} a_i^k + st_k z_n^k \le C y_n^k,\quad k=1,\dots,K,\, n=1,\dots,N \qquad (3)$$

$$z_n^k \ge y_n^k - y_{n-1}^k,\quad k=1,\dots,K,\, n=1,\dots,N \qquad (4)$$

$$\sum_{k=1}^{K} y_n^k = 1,\quad n=1,\dots,N \qquad (5)$$

$$I_{it}^{-},I_{it}^{+},x_{it}\ge 0,\quad I_{it}^{-},I_{it}^{+},x_{it}\in\Im,\quad I_{i0}^{-},I_{i0}^{+}=0,\ i=1,\dots,I \qquad (6)$$

The goal (1) is to find a plan that minimizes the sum of the costs of delayed production, storage costs of finished goods and the setup cost if the alloy is changed during furnace load.

Equation (2) balances inventories, delays and the volume of production of each item in each period. Constraint (3) ensures that the furnace capacity is not exceeded in single load. Constraint (4) sets variable z_n^k to 1, if there is a change in alloys in the subsequent periods, while constraint (5) ensures that only one alloy is produced in each sub-period.

3. Solution heuristics

Preliminary experiments performed by the authors indicated that the lot sizing problem presented in the previous section cannot be efficiently solved with simple heuristics that operate on a single solution (like tabu search or simulated annealing), as the number of variables is significant (few thousands for real size problems). Therefore two population heuristics have been used: genetic algorithm and differential evolution.

3.1. Genetic algorithm

In mathematical programming, and indirect, that has a non-standard structure, that usually needs to be decoded before the evaluation of objective function. Indirect representation, however, may better reflect the specifics of the problem, e.g. some constraints maybe included in its structure or it can allow for faster calculation of the objective function.

That is why a special representation of solution is used in the proposed genetic algorithm. This representation comprise three kind of vectors (chromosomes): at least one vector ***x*** representing the quantity of items that are produced in a given subperiod, the same number of vectors ***o*** representing the orders' numbers of the produced items, and one vector ***a*** representing alloy type that is produced in this subperiod. The number of vectors ***x*** and ***o*** in a chromosome is arbitrary set and limits the quantity of different items that can be produced in one subperiod. An exemplary solution with 10 subperiods, 10 items, 2 alloy types and maximum 3 changes allowed, written in the proposed representation is shown in Fig. 1. For example in the first subperiod 9 items for order 3 are produced, 50 items for order 4 are produced, and 33 items for order 2 are produced. All items are produced from alloy type 1.

i	1	2	3	4	5	6	7	8	9	10
$x1_i$	9	97	6	20	32	49	30	89	10	34
$x2_i$	50	3	66	28	64	28	62	16	43	73
$x3_i$	33	35	61	81	15	41	13	36	4	27
$o1_i$	3	8	5	6	1	9	1	9	3	7
$o2_i$	4	6	3	8	2	10	3	8	1	10
$o3_i$	2	9	2	10	4	7	5	7	4	6
a_i	1	2	1	2	1	2	1	2	1	2

Fig. 1. Solution representation used in proposed GA

Such representation allows for a significant reduction of the search space, and simultaneously ensures that only one type of alloy is produced during the single subperiod.

The representation has also this advantage that standard crossovers, like one-point, can be directly applied without any modification. All vectors are exchanged in the same positions, so the alloy for orders that are planned in a given period still match these orders. Contrary to the crossover, some mutation operators may cause that alloy of a given order does not match alloy planned in a given subperiod. The authors defined three different mutation operators. First mutation operating on items simply adds or subtracts 1.0 for a randomly chosen element of vector ***x***. This mutation does not disturb alloy of orders. Second mutation operates on orders and with a given probability it can change the order number in a randomly chosen element of vector ***o*** to another order number, that is produced from the same alloy. Finally, third mutation operates on alloy and can change alloy type in a randomly chosen subperiod (i.e. element of vector ***a***). The orders in this subperiod have to be changed for the ones that match new alloy type. Application of three different types of mutations allow for very precise exploitation of the solution space. Solutions to the recombination stage are taken on the basis of standard binary tournament (from two randomly chosen solutions, the one with better objective functions becomes the winner). After a series of experiments the crossover rate was set to 0.5, the first mutation rate to 0.2, and for remaining two mutations 0.02. Population size was set to 50 solutions. The pseudo-code of the algorithm is presented in Fig. 2.

```
Initialize population P with random values
Evaluate population P and print the best solution
while terminal_condition not met
  Select solutions for recombination with binary tournament
  Perform one-point crossover with probability 0.5
  Perform mutation1 with probability 0.2
  Perform mutation2 with probability 0.02
  Perform mutation3 with probability 0.02
  Evaluate population P and print the best solution
while end
```

Fig. 2. Outline of genetic algorithm used in experiments

3.2. Differential evolution

Differential evolution (DE) was developed in order to solve primarily continuous optimization problems, but due to the fact that it does not operate on gradients it can be applied to virtually any type of optimization problems. However, applications of DE to the lot-sizing problem described in the world literature are very rare. Lieckens and Vandaele proposed an efficient DE for solving an Advanced Resource Planning (ARP) model [4]. The most recently, Xu et al. proposed hybrid GA and DE algorithm for solving combined scheduling and lot-sizing problem [5].

Similarly to genetic algorithms DE operates on the population of solutions, but its mechanism relies mainly on the strategy of choosing solutions for the mutation stage. Many strategies were defined for DE, the authors decided to use the strategy denoted as DE/rand-To-best/1/bin/ [6] which means that the mutated solution is created on the basis of the best solution found so far and two other randomly chosen solutions, and then binary crossover is used (see Fig. 3).

Binary crossover simply replaces some of genes in original solutions with the values from the mutant solution with the Cr probability.

```
Initialize population P with random values
Evaluate population P and print the best solution
while terminal_condition not met
  for each solution p in P
     Select p1, p2 and p3 from P at random
     Generate mutant pm = s * (p_best - p1) + s * (p2 – p3)
     Perform binary crossover with p and pm with prob. Cr
     If f(p')> f(p) then Replace original solution p in P
  end for
while end
```

Fig. 3. Outline of differential evolution used in experiments

Contrary to the proposed genetic algorithm, differential evolution uses direct representation of a production schedule corresponding to the lot sizing model presented in Section 2. This means that the solution is a matrix of items and subperiods. Ssuch representation can easily lead to the situation that items planned for production in a given subperiod should be made from different alloy type, which means infeasible solution. In order to avoid such situation, a repair algorithm has been proposed. From the items that are planned in a given subperiod the one with the highest value (i.e. weight) is fixed to the plan, and the remaining items are removed from the plan if they are made from a different alloy type than the chosen one. Following parameters values were chosen in the experiments: crossover probability Cr = 0.9 and scale parameter s = 0.8.

4. Computational experiments

4.1. Test problems

Computational experiments were conducted on the basis of the test problems proposed in [2]. The characteristic of these problems is covered in Table 1.

Table 1.
Test problems characteristics

Parameter	Value
number of items (I)	50
number of days (T)	5
number of subperiods (N)	10
number of alloys (K)	10
demand (d_{it})	[10, 60]
weight of item (w_i)	[1, 30]
setup for alloy (st_k)	[5, 10]
setup penalty of alloy (s_k)	low: 5*st_k, high: 50*st_k
tightness of furnace capacity (Cap)	C/0.6, C/0.8, C/1.0, C/1.2

Ten instances of the problem were generated. The basis furnace capacity C was generated using the following formula corresponding to the total sum of the weights of ordered items:

$$C = \frac{\sum_{i=1}^{N}\sum_{t=1}^{T} d_{it} w_i + \sum_{k=1}^{K} st_k}{50} \qquad (7)$$

Each instance were computed for four variants of furnace capacity tightness Cap – from very tight C/0.6 to the very loose C/1.2. Also two variants of setup penalty for alloy type change between two subperiods were analysed: with relatively low value 5*stk and with ten times higher penalty value 50*stk.

4.2. Results of the experiments

Each problem instance was first run in CPLEX from IBM Optimization Studio 12.5 in order to determine lower bound for this instance. CPLEX algorithm and two heuristic algorithms (GA and DE) were run for 3 minutes for each problem instance and the combination of capacity tightness Cap and setup penalty sk. This means 80 different problems were computed. The calculations for each heuristic were repeated 10 times. Table 2 contains average values of a relative increase over the lower bound for a solution achieved by CPLEX solver and the solutions generated by both heuristics, and the standard deviation from the mean for the experiments with low penalty of alloy setup sk = 5.

Genetic algorithm occurred to be the best solver for low and normal value of the furnace capacity tightness. Differential evolution was better than CPLEX only for the loosest value of Cap, while for the tightest value of Cap both heuristics were significantly worse than CPLEX.

Table 2.
Results of the experiments for low penalty of alloy setup

Capacity tightness		CPLEX	GA	DE
C/1.2	***average***	**9.42**	**0.32**	**7.50**
	std.dev.	0.78	0.11	1.64
C/1.0	***average***	**27.20**	**1.20**	**32.68**
	std.dev.	2.37	1.11	8.68
C/0.8	***average***	**65.84**	**15.24**	**121.25**
	std.dev.	16.64	10.89	27.36
C/0.6	***average***	**175.66**	**263.86**	**626.65**
	std.dev.	117.29	123.72	316.94

However even the results provided by CPLEX were almost 2 times distant from the theoretical lower bound.

Table 3 presents the results achieved for the test problems with high penalty of alloy setup sk = 50. It means that the algorithm should prefer the solutions with a limited number of alloy type changes between two adjacent subperiods.

Table 3.
Results of the experiments for high penalty of alloy setup

Capacity tightness		CPLEX	GA	DE
C/1.2	***average***	**7.45**	**0.70**	**11.01**
	std.dev.	1.14	0.42	2.36
C/1.0	***average***	**22.34**	**1.02**	**31.93**
	std.dev.	2.87	0.37	6.78
C/0.8	***average***	**60.76**	**37.58**	**134.19**
	std.dev.	5.01	13.91	43.99
C/0.6	***average***	**49.57**	**96.91**	**247.98**
	std.dev.	11.14	9.46	39.95

Again, for a much higher value of the penalty of alloy setup, genetic algorithm achieved the best results, except for the tightest furnace capacity. This time DE was definitely the worst solver.

Presented experiments show that further improvements need to be implemented for both heuristics in order to make them competitive tools to the CPLEX solver, or even better as it was in the most cases in the experiments with low penalty of alloy setup.

5. Conclusions

In this paper, the computational intelligence algorithms are described for the lot-sizing and scheduling problem in single furnace-single casting line environment. The genetic algorithm proposed by us can achieve better results than CPLEX, and it can potentially handle more complex problems which can be expressed in any form (including if-then rules, external functions) that allows to assess the quality of solutions.

Although the CI algorithms work well for the problem under consideration, it should be noted that the lot-sizing problem does not fully describe the characteristic of the production planning in a foundry. LS model is primarily aimed at balancing resources without sufficiently taking into account the time requirements (clients' due dates). Therefore, the model requires an extension or reconstruction to reflect all the constraints and dependencies associated with production and marketing requirements.

References

[1] Stawowy, A. & Duda, J. (2012). Models and algorithms for production planning and scheduling in foundries – current state and development perspectives. *Archives of Foundry Engineering*. 12(2), 69-74. DOI: 10.2478/v10266-012-0039-4.

[2] de Araujo, S.A., Arenales, M.N. & Clark, A.R. (2008). Lot sizing and furnace scheduling in small foundries. *Computers & Operations Research*. 35(3), 916-932. DOI: 10.1016/ j.cor.2006.05.010.

[3] Goren, H., Tunali, S. & Jans, R. (2010). A review of applications of genetic algorithms in lot sizing. J*ournal of Intelligent Manufacturing*. 21(4), 575-590. DOI: 10.1007/s10845-008-0205-2.

[4] Lieckens, K. & Vandaele ,N. (2011). Differential evolution to solve the lot size problem. *ERN: Production; Cost; Capital & Total Factor Productivity, Value Theory*. DOI: 10.2139/ssrn.1968316.

[5] Xu, X., Li, L., Fan, L., Zhang, J., Xuhua, Y. & Wang, W. (2013). Hybrid Discrete Differential Evolution Algorithm for Lot Splitting with Capacity Constraints in Flexible Job Scheduling. *Mathematical Problems in Engineering*. DOI:10.1155/2013/986218.

[6] Feoktistov, V. (2006). *Differential Evolution. In Search of Solutions*. Berlin Heidelberg: Springer Verlag.