Scientific Research and Essays Vol. 7(41), pp. 3515-3527, 27 October, 2012
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE11.675
ISSN 1992-2248 ©2012 Academic Journals

Full Length Research Paper

An imperialist competitive algorithm mixed model assembly line sequencing problem on just in time system

Asqar Hemmati*, Mojtaba Hemmati and Mohammad Ahmadifard

Department of Industrial Engineering, Islamic Azad University, Abhar Branch, Abhar City, Zanjan, Iran.

Accepted 11 June, 2012

Mixed-model lines are used to produce several kinds of models in small lots without carrying large inventories. The production sequence for the mixed-model sequencing problem depends on the goals of the production facility. In order to enjoy the useful application of these lines, it is vital to devise a schedule for assembling the different products to be determined. Based on the NP-hardness of the problem, this present paper introduces an imperialist competitive algorithm (ICA) in three phases so as to solve a just-in-time (JIT) sequencing problem where the diversity of production rates to be optimized. Performance of the ICA was compared against two other search heuristics genetic algorithm (GA) and simulated annealing (SA) in small, medium and large problems. To compare presented algorithm with previous ones, an extensive computational study on 3 sets of benchmark problems has been conducted. Experimental results showed that our algorithm outperforms the previous algorithms, in respect of comparison metric.

Key words: Mixed-model sequencing, just-in-time, imperialist competitive algorithm, genetic algorithm, simulated annealing.

INTRODUCTION

Mixed-model assembly line (MMAL) is a type of production line which is able to produce several small-sized production lots and is highly responsive to sudden demand fluctuations without necessitating holding large stock of in process inventories. Nowadays, most manufacturing firms employ this type of line because of the increasing varieties of products in their attempts to quickly respond to diversified customer demands. Improvement of new technologies, competitiveness, diversification of products, and large customer demands has got practitioners to use different methods of production development.

MMAL sequencing is a problem of determining a sequence of the product models, in which a lot of emphasis is put on enhancing the line utilization. The effective utilization of a mixed-model line requires that managers pay attention two major problems (Miltenburg and Sinnamon, 1989):

1. The assignment of tasks to workstations (that is, the line balancing problem) and
2. The models sequencing on the line (that is, mixed model sequencing problem).

In this work, we concentrate on the mixed-model sequencing problem assuming that line balancing has already been done. It has used an average cycle time for assembling the products. When sequencing products on JIT mixed-model assembly line, the decision maker (DM) tries to achieve two objectives: parts usage smoothing and workload smoothing at the different workstations. Monden (1983) explained them as follows: The first objective aims at minimizing variation in parts utilization at the different workstations of the assembly line. If long-time consuming models are successively sequenced at a particular workstation, then a line-stoppage could occur.

*Corresponding author. E-mail: asqar.hemmati@gmail.com.

Therefore the second objective aims at minimizing variation of the workload associated with each workstation.

In this paper also, we have first objectives. The usage rate is a measure of the company's ability to keep the schedule level, or evenly intermixed - keeping the raw materials for the different products arriving to the system at as constant a rate as possible. Because JIT systems are concerned with having the right parts at the right place at the right time, sequencing must be done so that the raw materials are introduced into the system at a fairly uniform rate (McMullen and Frazier, 2000). Miltenburg (1989) has presented a metric to measure this usage rate, which was adopted for this research. This metric can be thought of as a surrogate for a firm's ability to produce several items at a uniform rate.

The structure of this paper is as follows: First, this paper reviews the previous studies, and presents a mathematical formulation of the mixed-model assembly line. Thereafter, the discrete particle swarm optimization is proposed. The comparison of algorithms is then explained. The experimental results are given and various test problems are provided. Finally, we present our conclusions.

LITERATURE REVIEW

The sequencing determination mixed-model assembly line problem was introduced in 1960. The researches activities are divided in two main parts. Those are the researches before and after JIT production system. In the first part, the sequence determination problem is defined based on the line output criteria, and the second part the most researches are based on pert production output criteria. The sequence will vary depending upon the possible goals of the company.

Monden (1983) defined two goals for the sequencing problems: (1) smoothing the workload (total operation times at each workstation on the assembly line) and, (2) keeping a constant rate of usage of every part used by the line. Toyota corporation developed goal chasing I (GC-I) and II (GC-II) methods to solve these problems. GC-I minimizes the one stage and assumes that the length of the unique workstation is equal to zero. GC-II solves GC-I under a special assumption regarding the product structure. Goal 1 recognizes that all products do not have the same operation time at each station on the line. If products with relatively longer operation times are successively scheduled, delays will eventually occur and line stoppages is the result.

Okamura and Yamshina (1979) presented a heuristic procedure for sequencing products (with different operation times) on a MMAL. Their objective was to minimize line stoppages. Other works on this problem include those by Macaskill (1973) and Thomopoulos (1967). Miltenburg and Sinnamon (1989) developed a non-linear programming for the second goal mentioned previously. The time complexity function of the proposed program was exponential; therefore, he developed and solved the problem by applying two heuristic procedures.

Miltenburg (1989) solved the same problem with a dynamic programming algorithm. Inman and Bulfin (1991) solved the problem proposed by Miltenburg and Sinnamon (1989) by converting it to a mathematically different approach. Other objectives have also been considered by a number of researchers. Sequencing mixed-model assembly lines have also been studied as a multi-objective problem.

Bard et al. (1994) developed a model involving two objectives minimizing the overall line length and keeping a constant rate of part usage. They solved the problem by using the weighted sum and proposed a tabu search (TS) method. Hyun et al. (1998) addressed three objectives minimizing total utility work, keeping a constant rate of part usage, and minimizing total setup cost. This problem was solved by proposing a new genetic evaluation and selection mechanism.

Aigbedo and Monden (1997) developed a parametric procedure to jointly consider the variations in finished-goods production, part usage, assembly line workload, and subassembly line workload. Zeramdini et al. (2000) proposed a two-step approach to address the bi-criteria sequencing considering part usage and workload smoothing. After the initial sequence is determined by a two-stage heuristic method for part usage consideration, it is divided into a number of subsequences equal to the supply frequency. Each subsequence is then resequenced for workload smoothing. An evaluation method based on part usage of each subsequence was also proposed.

McMullen (1998) considered two objectives minimizing number of setups and keeping a constant rate of part usage, and solved this problem by a TS method. McMullen and Frazier (2000) developed a simulated annealing (SA) method for the model used by McMullen (1998) and compared it against the TS method.

McMullen (2001a, b, c) also solved the same problem by using genetic algorithms (GAs), kohonen self-organizing map (SOM) and ant colony optimization, respectively, and compared their performance with SA and TS methods. Mansouri (2005) proposed Multi-Objective Genetic Algorithm (MOGA) to solve the problem proposed by McMullen (1998).

MATHEMATICAL FORMULATION OF PARTS USAGE OBJECTIVE (PU)

Considering the previous discussions on the goals of this study, the following mathematical formula was used. However, the concept of a minimum part set (MPS) which is a vector representing a product mix (Hyun et al., 1998; Zaramdini, 2003) was used in this paper.

Notations

In the development of model, we used the following notation:

α: Number of variety of products. β: Number of variety of parts. DT: Total production quantity of all product varieties. d_i: Production quantity of each product variety *i* $(d_1, d_2, ..., d_\alpha)$: is called the Product Composition Structure (PCS). b_{ij}: Number of units of part variety j required per unit of product variety i. $: b_{ij} = 1$ If product i uses part j and $b_{ij} = 0$ if product i does not use part j. N_j: Total quantity of part j required by all products. $X_{j,k}$: Total quantity of part j to be utilized for assembling the products of determined sequence from the first to the k*th* stage. $P_{j,k}$: Cumulative quantity of product *i* at stage *k.*

Optimizing the PU objective means the selection of a sequence $\pi = \{P_{i,k}\}$ which minimizes the following objective function:

$$f\ (x) = F_{PU} = \sum_{k=1}^{DT} \sum_{j=1}^{\beta} \left(\frac{k \times N_j}{DT} - X_{j,k} \right)^2 \tag{2}$$

Subject to:

$$\sum_{i=1}^{\alpha} P_{i,k} = k \qquad k = 1, ..., DT \tag{3}$$

$$0 \le P_{i,k} - P_{i,k-1} \le 1 \quad i = 1, ..., \alpha \quad k = 1, ..., DT \tag{4}$$

$$P_{i,k} \ is\, a\, non-negative\, integer\ \forall i\ and\ k \tag{5}$$

$$P_{i,0} = 0\ \ \forall\ \ i \tag{6}$$

$$X_{j,k} = \sum_{i=1}^{\alpha} (P_{i,k} \times b_{i,j}) \qquad \forall\ j\ and\ k \tag{7}$$

$$X_{j,DT} = N_j \quad and \quad X_{j,0} = 0 \qquad \forall\ j \tag{8}$$

$$N_j = \sum_{i=1}^{\alpha} (d_i \times b_{ij}) \qquad j = 1, ..., \beta \tag{9}$$

The objective function in the foregoing seeks to minimize the variation in parts consumption. The first constraint ensures that exactly k units are scheduled in periods 1 through k, while the last two constraints ensure, for each model, that either one units is scheduled in a given period or else it is not schedule at all.

Combinatorial complexity

Finding production sequences with desirable levels of both number of setups and production rates variation is NP-hard as pointed out by McMullen (2001b). Total number of sequences for a mixed-model sequencing problem having a product can be computed using the general formula to compute the number of permutations of a multi-set (Walpole and Myers, 1985) as follows:

$$Total\ \ sequences = \frac{\left(\sum_{i=1}^{a} d_i \right)!}{\prod_{i=1}^{a} (d_i)!} \tag{10}$$

As the problem increases in size, the number of feasible solutions increases in an exponential fashion, thereby attainment of optimal solutions becomes impractical for large problems. Problems with a large number of possible solutions usually cannot be solved to optimality within a reasonable amount of time.

THE PROPOSED IMPERIALIST COMPETITIVE ALGORITHM (ICA)

ICA is proposed by Esmaeil et al. (2008). They showed the algorithms capability in dealing with different types of optimization problems (Atashpaz et al., 2008). We use this algorithm in a MMAL problem. Similar to other evolutionary algorithms, this algorithm starts with an initial population of solution which is named country. Some of the best countries in the population are chosen to be the 'imperialists' and the rest are the 'colonies' of these imperialists. All the colonies of initial population are distributed among the imperialists based on their power.

A set of one imperialist and its colonies is called an 'empire'. The power of an empire which is equivalent to the fitness value in a genetic algorithm (GA) is inversely proportional to its cost. After distribution of all colonies among imperialists, these colonies start moving towards their relevant imperialist country. The total power of an empire relates to both the power of the imperialist country and the power of its colonies. This fact will be modeled by defining the total power of an empire by adding the percentage of the mean power of colonies to their imperialists. Then the imperialistic competition begins among all the empires. Any empire which is not strong enough to compete and cannot increase its power (or at least prevent decreasing it) will be eliminated.

The imperialistic competition will lead slightly to an increase in the power of powerful empires and a decrease in the power of weaker ones. Weak empires will

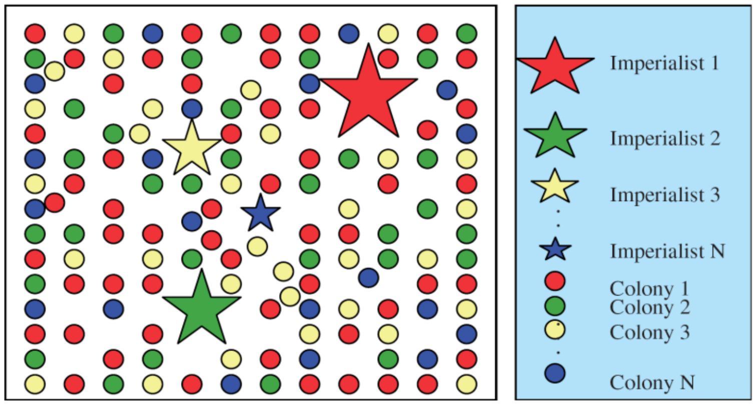

Figure 1. Generating the initial empires: The more colonies an imperialist possess, the bigger its relevant ✸mark.

lose their power and finally they will collapse. The movement of colonies towards their relevant imperialists through the competition among empires and also the collapse mechanism will hopefully cause all the countries to converge to a state in which there is just one empire in the world and all the other countries are colonies of that empire. In this ideal new world, colonies have the same position and power as the imperialist. The implementation of this algorithm in MMAL is as follows:

Begin ICA

(1) Initialize the empires.
(2) Move the colonies toward their relevant imperialist (assimilating).
(3) If there is a colony in an empire which has lower cost than that of imperialist, exchange the positions of that colony and the imperialist.
(4) Compute the total cost of all empires (related to the power of both imperialist and its colonies).
(5) Pick the weakest colony from the weakest empire and give it to the empire that has the most likelihood to possess it (imperialistic competition).
(6) Change some weakest colonies with new ones randomly (revolution).
(7) Eliminate the powerless empires.
(8) If stopping criteria met, stop, if not go to step 2.

End ICA

Generating initial empires

The main purpose of optimization is to find an optimal solution; each solution in this algorithm is shown as a country (similar to chromosome in GA In an n-dimensional optimization problem, a country is a $1 \times n$ array (the array of country represents a sequence of models of product). Initial population is generated as follows. Initial population generation is the first step in the proposed ICA. In this study each solution (country) as a sequence of each array of the products in a production cycle of MPS demands that its length is equal to DT. For example consider 5 productions A, B, C, D and E ($\alpha = 5$), So is demand for each items according: $d_A = 6, d_B = 3, d_C = 1, d_D = 1$ and d_E =1. As a result, total demand is equal to DT = 6 + 3 + 1 + 1 + 1 = 12. For example, a sequence BABDAEAACAB a feasible solution can be for this problem.

Optimization of an algorithm starts with generating initial population (countries) of size pop. N_{imp} Of the most powerful countries are selected to be imperialists. The remaining $N_{col}$of the population will be the colonies each belongs to imperialists. Thus we have two types of countries; imperialist and colony. To form the initial empires, we distribute the colonies among imperialists based on their power. To distribute the colonies among imperialists, we define the normalized cost of an imperialist as $N_{PU} = max_i\{PU_i\} - PU_n$; where $PU_n$is the cost of nth imperialist and N_{PU} is its normalized cost. The normalized power of each imperialist is shown as follows:

$$p_n = \left| \frac{N_{PUn}}{\sum_{i=1}^{N_{imp}} N_{PUi}} \right|$$

On one side, the normalized power of imperialist shows the number of colonies that should be possessed by that imperialist. Thus, the initial number of colonies of an empire will be $NCol_n = round\{p_n, N_{col}\}$. Where $NCol_n$is the initial number of colonies of *n*th empire. For distributing of colonies among the imperialist we randomly choose $NCol_n$of the colonies and give them to it. The imperialist and its colonies will form *n*th empire. Figure 1 shows the initial population of each empire. As

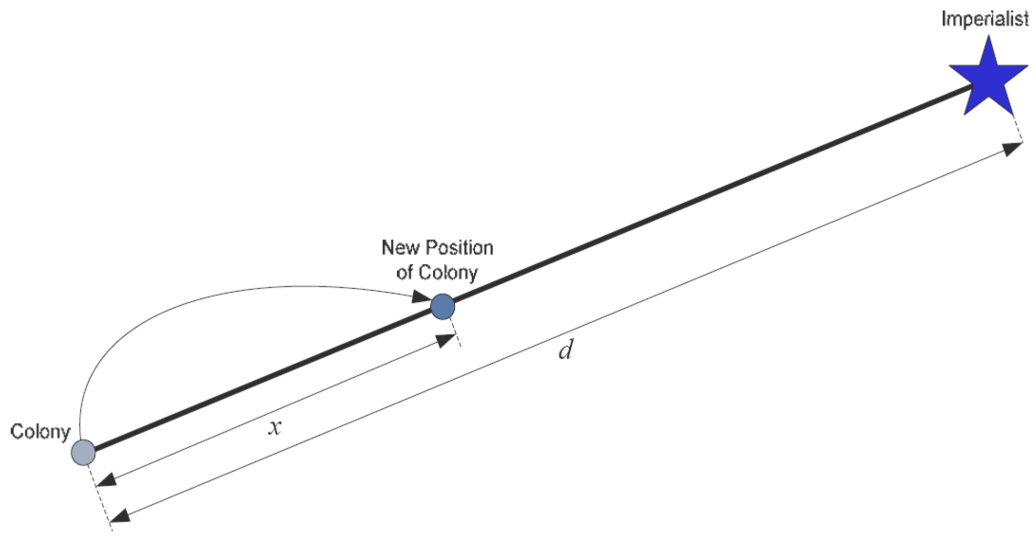

Figure 2. Moving colonies toward their relevant imperialist.

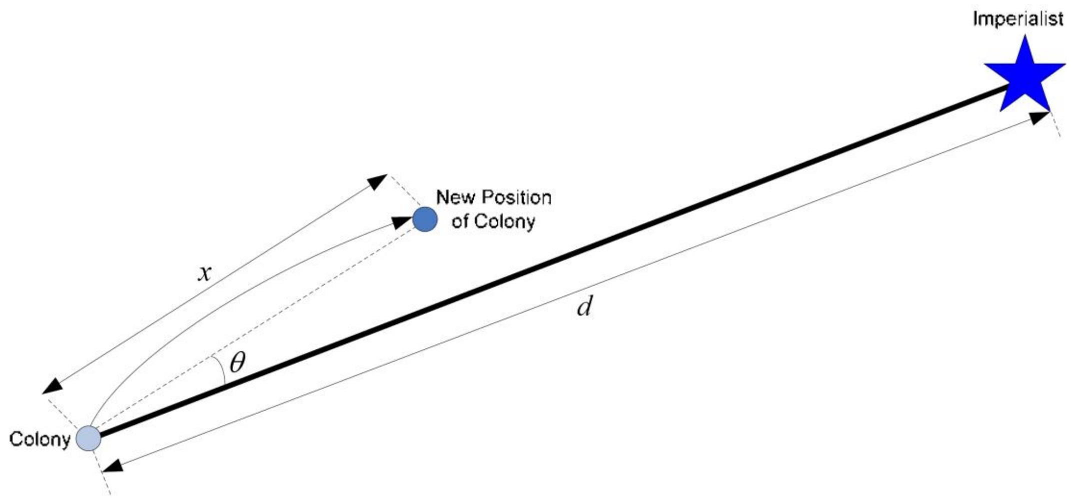

Figure 3. Moving colonies toward their relevant imperialist in a randomly deviated direction.

shown in Figure 1, bigger empires have a greater number of colonies while weaker ones have less. In Figure 1, imperialist 1 has formed the most powerful empire and has the greatest number of colonies.

Moving the colonies of an empire toward the imperialist (assimilating)

Imperialists countries started to improve their colonies. This fact has been modeled by moving all the colonies toward the imperialist. This movement is shown in Figure 2 in which the colony moves toward the imperialist by x units. The new position of colony is shown in a darker color. The direction of the movement is the vector from colony to imperialist. In this figure, x is a random variable with uniform (or any proper) distribution. Then for x we have $x \sim U(0, \beta \times d)$; where β is a number greater than 1 and d is the distance between colony and imperialist. A β > 1 causes the colonies to get closer to the imperialist state from both sides.

To search different points around the imperialist we add a random amount of deviation to the direction of movement. Figure 3 shows the new direction. In this figure θ is a random number with uniform (or any proper) distribution. Then $\theta \sim U(-\gamma, \gamma)$; where γ is a parameter that adjusts the deviation from the original direction. Nevertheless the values of β and γ are arbitrary, in most of our implementation a value of about 2 for β and about π/4 (Rad) for γ have resulted in good convergence of

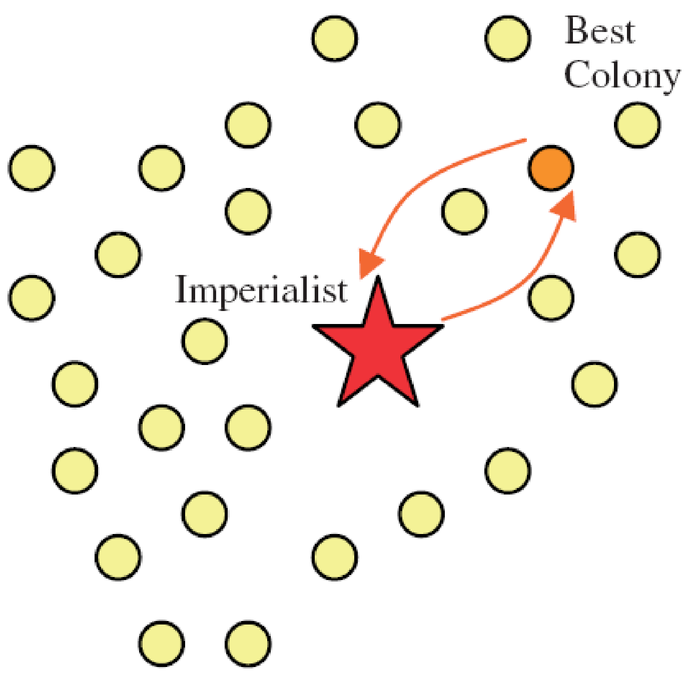

Figure 4. Exchanging the positions of a colony and the imperialist.

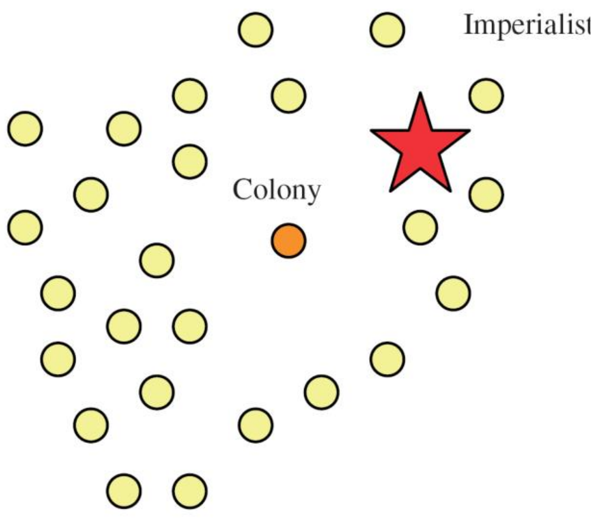

Figure 5. The entire empire after position exchange.

countries to the global minimum.

Exchanging positions of the imperialist and a colony

Owing to movement towards the imperialist, a colony may reach a position with lower cost than imperialist. In such a condition, the position of imperialist and colony are changed. After that, the algorithm will continue by the imperialist in a new position and then colonies start moving toward this position. Figure 4 depicts the position exchange between a colony and the imperialist. In Figures 4 and 5 the best colony of the empire is shown in a darker colour. This colony has a lower cost than that of the imperialist. Figure 5 shows the whole empire after exchanging the position of the imperialist and that colony.

Total power of an empire

Total power of an empire is mainly affected by the power of the imperialist country, though the power of the colonies of an empire has an effect, albeit negligible, on the total power of that empire. Therefore, the equation of total cost is:

$$TPU_n = PU(imperialist_n) + \xi mean\{PU(colonies\ of\ empire_n)\}$$

Where TPU_n the total is cost of the *n*th empire and ξ is a positive number which is considered to be less than 1. A little value for ξ causes the total power of the empire to be determined by just the imperialist and increasing it will increase the role of the colonies in determining the total power of an empire.

Imperialistic competition

As mentioned previously, all empires try to possess the other empires' colonies and control them. Through this imperialistic competition the power of weaker empires will decrease and as a result the power of more powerful ones will increase. We model this competition by just picking one of the weakest colonies of the weakest empires and making a competition among all empires to possess this colony.

Figure 6 illustrates the modeled imperialistic competition. Based on their total power, in this competition, each of the empires will have a likelihood of taking possession of the mentioned colonies. In other words, these colonies will not be possessed by the most powerful empires; however, these empires will be more likely to possess them.

To start the competition, first, the possession probability of each empire should be found based on its total power. The normalized total cost is simply obtained by $N_{TPUn} = max_i\{TPU_i\} - TPU_n$; where TPU_n and N_{TPUn} are respectively total cost and normalized total cost of nth empire. Having the normalized total cost, the possession probability of each empire is given by:

$$P_{p_n} = \left| \frac{N_{TPUn}}{\sum_{i=1}^{N_{imp}} N_{TPUi}} \right|$$

To divide the mentioned colonies among empires based on the possession probability of them, we form the vector

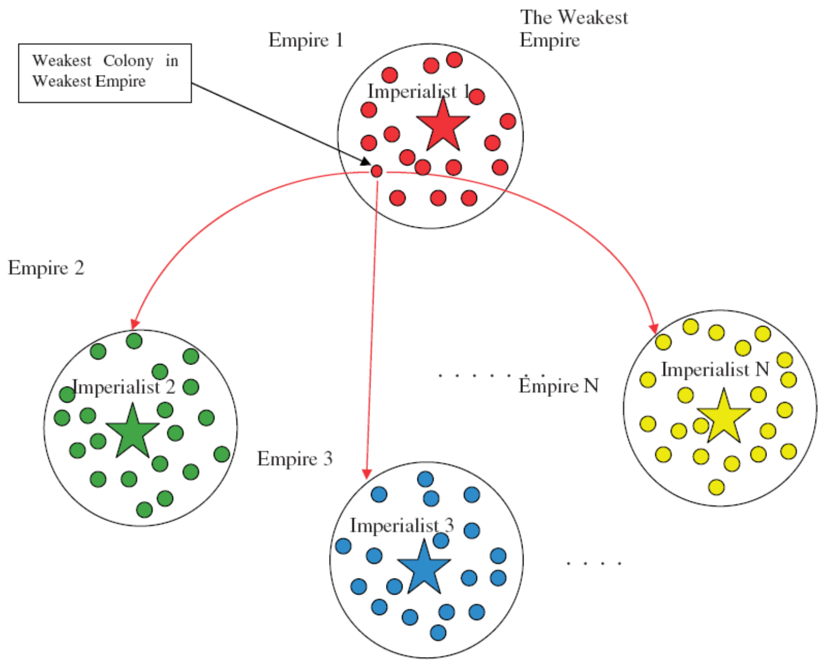

Figure 6. Imperialistic competition. The more powerful an empire is, the more likely it will possess the weakest colony of the weakest empire.

P as $P = [P_{p1}, P_{p2}, \dots, P_{p N_{imp}}]$. Then we create a vector R with the same size as P whose elements are uniformly distributed random numbers.

$$R = [r_1, \dots, r_{N_{imp}}] \,; r_1, \dots, r_{N_{imp}} \sim U(0,1)$$

Vector D is obtained by subtracting R from P.

$$D = P - R = \left[D_1, \dots, D_{1N_{imp}}\right] = [P_{p1} - r_1, \dots, P_{p N_{imp}} - r_{N_{imp}}]$$

Referring to vector D, we will hand the mentioned colonies into an empire whose relevant index in D is maximum.

Revolution

In each iteration we select some of the weakest colonies and replace them with new ones, randomly. The replacement rate is named as the revolution rate.

Eliminating the powerless empires

Powerless empires will collapse in the imperialistic competition and their colonies will be distributed among other empires. In modeling collapse mechanism, different factors can be defined for considering an empire powerless. In this paper, we assume an empire collapses when it loses all of its colonies.

Stopping criteria

The algorithm continues until no iteration is remaining or just one empire exists in the world.

COMPARISON OF ALGORITHMS

Performance of the ICA was compared against two other search heuristics GA and SA, and each structure is as follows:

Genetic algorithm (GA)

Genetic algorithms start with a population of solutions, whereas most stochastic search methods start with a single solution. An initial population is formed randomly or by means of a heuristic algorithm. Solutions are encoded in a form, which are called chromosomes. Each chromosome shows a complete solution to a problem. They are each assigned a fitness score that represents the ability of chromosomes to compete for mating and staying alive. Parents are picked up to mate according to their fitness values.

The fitter chromosomes produce more offspring than the less fit chromosomes. The solution set is then imposed to crossover, mutation and inversion. These stochastic operators are required for diversifying the solution pool and especially getting better solutions. Since the size of the population should be maintained

Table 1. Average relative percentage deviation ($\overline{RPI}$) and average CPU time for algorithms.

Problem size		Comparative algorithms					
		GA		SA		ICA	
		RPI	CPU time	RPI	CPU time	RPI	CPU time
Small problem	PS1	0.0000	0.22	0.0000	0.27	0.0000	0.26
	PS2	0.0000	1.34	0.0000	1.22	0.0000	1.43
	PS3	0.0000	2.46	0.0000	2.05	0.0000	2.14
	PS4	0.4628	3.58	0.4103	3.66	0.3421	4.04
	PS5	0.5921	4.7	0.5345	5.23	0.5042	4.12
Medium problems	PM1	0.0000	15.41	0.0000	13.33	0.0000	15.44
	PM2	1.0000	30.22	1.0000	27.14	1.0000	30.42
	PM3	0.4431	44.36	0.2381	39.67	0.2642	38.93
	PM4	0.3325	35.19	0.3005	38.47	0.2842	33.89
	PM5	0.4538	55.32	0.1325	46.87	0.1634	50.13
Large problems	PL1	0.4921	373.44	0.3845	342.52	0.3367	312.36
	PL2	0.3842	412.65	0.2067	405.19	0.2005	420.87
	PL3	0.5823	574.18	0.4942	550.28	0.3895	985.93
	PL4	0.3848	778.76	0.2884	693.19	0.2274	669.45
	PL5	0.8491	894.55	0.6422	817.88	0.5942	785.67
Average		0.3984533	215.09	0.3087933	199.13133	0.2870933	223.672

statically, some weak individuals in the population die, and better solutions thrive to stay alive. The cycle continues until a certain number of iterations are executed or once the population converges. The solution procedure is summarized in the pseudo-code in Table 1.

Algorithm 2: The main procedure of genetic algorithm (GA)

```
1. Begin,
2. Choose initial population,
3. Repeat,
4. Evaluate the individual fit nesses function of a certain proportion of the population,
5. Select pairs of best-ranking individuals to reproduce,
6. Apply crossover operator,
7. Apply mutation operator,
8. Apply inversion operator,
9. Until terminating condition,
10. End.
```

Operator's genetic algorithm

There are numerous crossover and mutation methods. In this work order crossover (*OX*), inversion (*INV*) operators (Michalewicz, 1992) and mutation are used.

Order crossover: To illustrate how it works, consider the following two parent sequences:

Parent 1: A A A A **|** A A B B B **|** B C C D D.

Parent 2: D A B A **|** B C B A A **|** B C A D A.

Brackets designate the portion of the sequences that will remain intact and become part of the offspring in the crossover process. The location of the brackets is determined at random, however the left bracket must be to the right of the first character in a sequence and the right bracket must be to the left of the last character in the sequence. A new parent is then created by moving all the characters appearing after the right bracket of the original parent at the beginning of the sequence. The results of this are shown as follows:

$\text{Parent }1'$: B C <u>C D | D</u> A A <u>A A | A A</u> B <u>B B</u> (C D D A A A A B B).

$\text{Parent }2'$: B <u>C</u> A <u>D</u> | A <u>D A</u> B <u>A</u> | B <u>C B A A</u> (C D D A A C B A A).

From this new sequence, characters that match the characters between the brackets of the other original parent are removed. For instance, from $\text{Parent }1'$, the first two A's, the first two B's, and the first C are removed because $\text{Parent }2'$ has the sequence BCBAA between its brackets. The sequence between the brackets of the original parent and this shortened list as shown in previous parentheses from the other parent is then used to construct an offspring. For instance, the sequence AABBB is taken from Parent 1 and the shortened list from $\text{Parent }2'$ is added starting at the right bracket and wrapping around to the beginning of the sequence. Using this technique, the following offspring are created (McMullen, 2001a).

Offspring 1: C B A A | A A B B B | C D D A A.
Offspring 2: A A B B | B C B A A | C D D A A.

Inversion: Inversion is a un-array operator that generates offspring from a single parent. It first chooses two random cut points in a parent. The elements between the cut points are then reversed. An example of the inversion operator is presented as follows:

Before inversion**:** C B A | B A B C | C A.
After inversion: C B A | C B A B | C A.

Mutation: The mutation operator changes one or some of the genes in a single parent randomly. This operator has been used to increase the diversity. In this paper, swapping mutation was used. Consider the following sequence:

Before mutation: B A A $\underline{A}$ C A B A B $\underline{B}$ C D D D

The two gray elements are randomly selected unique elements that are targeted for swapping. After swapping, or mutation, the sequence is as follows:

After mutation: B A A $\underline{B}$ C A B A B $\underline{A}$ C D D D

Simulated annealing (SA)

The SA technique proposed by Kirkpatrick et al. (1983) is an iterative, stochastic, neighborhood-based search method motivated from an analogy between the simulation of the annealing of solids and the strategy of solving combinatorial optimization problems. SA has been widely applied to solve combinatorial optimization problems (Yao, 1995). It is inspired by the physical process of heating a substance and then cooling it slowly, until a strong crystalline structure is obtained. This process is simulated by lowering an initial temperature by slow stages until the system reaches to an equilibrium point, and no more changes occur. In this paper similar to Behnamian et al. (2009), algorithmic framework of SA is described:

Algorithm 3: The main procedure of simulated annealing (SA).

```
1. Input: an instance $x$ of a combinatorial optimization problem
2. $S \leftarrow$ Generate Initial solution ()
3. $k \leftarrow 0$
4. $Tk \leftarrow$ Set Initial temperature ()
5. While termination conditions not met do
6. $S' \leftarrow$ Pick neighbor at random (N(S))
7. If $f(S') \le f(S)$ then
8. $S \leftarrow S'$;
9. Else
10. Accept $S'$ as new solution with probability p ($T_k$, $S'$, $S$)
11. End if
12. Adapt temperature ($T_k$)
13. End while
14. $S_{best} \leftarrow S$
15. Output: $S_{best}$, "candidate" to optimal solution for $x$.
```

The main idea of this technique is to start from some initial solution, π′, and successively move among neighboring solutions until the stopping condition is satisfied. At each iteration, *iter*, a random solution, π′, is selected from the neighborhood of actual solution (πiter) and it replaces with a probability.

$$P(t_{iter}, \pi_{iter}, \pi') = \min\left\{1, \exp(-\frac{f(\pi') - f(\pi_{iter})}{t_{iter}}\right\} \qquad (5)$$

Where t_{iter} is a parameter called the temperature at iteration *iter*. The temperature decreases during the search process according to the cooling scheme. The performance of SA depends on the following parameters, which have to be precisely selected: initial temperature, cooling scheme and final temperature. The following presents the implementation of SA algorithm. For detailed description of the SA method, the reader is referred to the literature of Aarts and Lenstra (2003), Kirkpatrick et al. (1983) and Tian et al. (1999).

Initial temperature

The initial temperature is selected on the basis of $K = nm + 1$ solutions $\pi_0; \pi_1; ...; \pi_k$, where π_j is randomly selected from the neighborhood of π_{j-1} and π_0 is an initial solution. The initial temperature is defined as:

$$t_0 = \varphi_1 \frac{\delta}{mn}, \qquad (6)$$

Where $\delta = \max_{j=1,...,k}\{f(\pi_j) - f(\pi_{j-1})\}$.

Cooling scheme

The temperature changes in every iteration according to the logarithmic cooling scheme:

$$t_{j+1} = \frac{t_j}{1 + \lambda t_j}, \qquad (7)$$

Where parameter λ is defined as:

$$\lambda = \frac{t_0 - t_f}{f_{t_{f}t_{t_0}}}, \qquad (8)$$

And f is 200. The final temperature t_f is determined from the following expression (Janiak et al., 2007):

$$t_f = \varphi_2 \frac{f(\pi_0)}{mn}. \qquad (12)$$

COMPUTATIONAL EXPERIMENTS

Data and test problems

In order to compare these algorithms against together, the problems used in McMullen (2001a) and Zaramdini (2003) were selected. These include 3 problem sets, each set consist of 5 problems, as presented in Appendix Table 1. They cover a diverse set of mixed-model sequencing problems, from the smallest problem with 11,880 solutions to the largest one with 6.334×10^{103} possible solutions.

Appendix Table 1 refers to the Bill of materials (b_{ij}) where for example, product A needs one unit of each part type: b, c, d and f. But for the assembly of product A, the part types a, e, g and h are not needed. Note that the small problem of the products type A to E, medium problems of the products type A to J and large problems of the products type A to O of Appendix Table 1 is used. In here, each problem 20 times by the algorithm performed, and the best solution for each algorithm in each step we have compared. These algorithms have been compiled in Matlab 7.0 and all numerical examples were tested on a PC with Intel Pentium 4, 1.67 GHz processor, 256 memory and windows XP professional operating system.

Parameters settings

The performance of the meta-heuristic algorithms is a direct relationship with parameters setting in a way that wrong choice of the parameters of the fully operational algorithms can bring about malfunctioning. There are several parameters that may influence the performance of the algorithms. For example, the larger population size may find better solution quality but cost higher computational expense.

It should be noted that changing these parameters may result in different outcomes than those achieved in this research. The numerous parameters of an ICA algorithm can be adapted to maximize the convergence on each problem. However our experimental approach was to select values which present a good trade-off in order to have a problem-free implementation. We have applied parameters tuning only for the Zeta (ξ), revolution rate, Revolution rate, Number of imperialists and Number of (countries, iterations) were set to 0.05, 0.4, 9 and (300; 1000) respectively.

In GA, the population size was tenfold the total number of units for all products, crossover, mutation and inversion probabilities were taken as 0.8, 0.1 and 0.1 respectively. Selection strategic for the mate selection, by tournament selection with a size of 2 to carry out the order crossover. Again the tournament selection with a size of 2 was used for constructing the population for the next generation. In addition, GA employed the insert operator as a mutation scheme. Initial temperature parameter φ_1 used uniform (0.5, 1) and final temperature parameter φ_2 used uniform (0, 0.1) in SA algorithm.

Experimental results

In this section, we are going to compare the proposed ICA, GA and SA for part usage problem in MMAL. All algorithms are coded in Matlab7.0 software and run on a PC with Intel Pentium 4, 1.67 GHz processor, 256 memory and windows XP professional operating system. After computation of objective line stoppage of each algorithm for its instances, the best solution obtained for each instance (which is named Min_{sol}) by any of the three algorithms is calculated. We use relative percentage index (RPI) as performance measure to compare the methods, because RPI fulfills some drawbacks of relative percentage deviation (RPD) in case of the line stoppage objective. When each algorithm has been obtained for its instances, the best and worst solutions obtained for each instance (which are named Min_{sol} and $Worat_{sol}$, respectively) by any of the three algorithms are calculated. RPI is obtained by given formula as follows:

$$RPI = \frac{A\lg_{sol} - Min_{sol}}{Worst_{sol} - Min_{sol}} \qquad (12)$$

Where Alg_{sol} is value objective function obtained for a given algorithm and instance. RPI takes value between 0 and 1. Clearly, lower values of RPI are preferred. Achieved results of the experiments for each comparing algorithm are presented in Table 1. For exclusion of the by-effects caused by randomized results, we execute each of the exemplary problems set for five times and the average of the attained results has founded the basis of the comparisons. Notice that these algorithms are run in equal times and first time in which the algorithm solutions do not change are comparison criterion we considered.

As you observe in Table 1 the out coming results of the average RPI and the average operation time of the algorithm is much better comparing the other methods. In Table 1, for showing the efficiency of algorithms in against together the statistical analysis is used, that is

Table 2. Pair-wise comparisons of results from RPI algorithms DPSOL, DPSO, GA and SA against each other.

Algorithm	Test	Results of *P-value*
ICA : GA	-0.88	0.810
ICA : SA	-1.08	0.503
GA : SA	1.33	0.135*

* Means the paired *t*_test is significant, that is, $P - Value < \alpha$

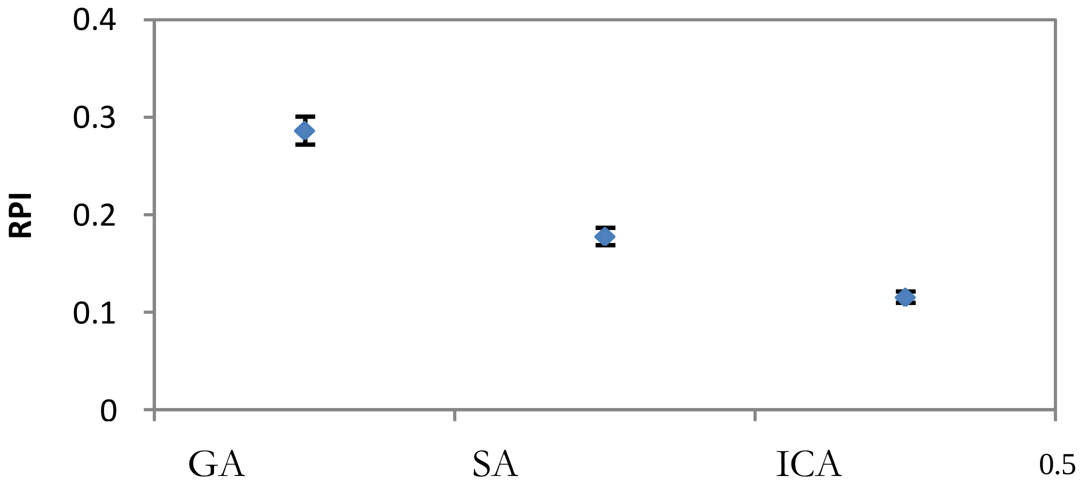

Figure 7. Plot of $\overline{RPI}$ for the type of algorithm factor.

pair-wise comparisons were made at significance level α = 0.01 employing one-tail *t*-test, since basically each problem is independent of the other problems. In fact the comparison is about the average RPI in the achieved results of the algorithms. In this hypothesis testing, alternative hypothesis is $H_1 : D > 0$ versus $H_\circ : D \leq 0$ whereas D is different between the averages RPI of comparative algorithms. Therefore, assume zero can be rejected if and only if:

$$t = \frac{\overline{D}}{S_D / \sqrt{n}} > t_{\alpha;n-1} \tag{11}$$

t and test of hypothesis results from the confidence level of 95% for average RPI is given in Table 2. As you see in Table 2 with the confidence level of 95% the average RPI in the achieved results of the ICA is much better than the other ones in the presented problem, and also the average RPI for SA results are less than GA. Figure 7 shows the diagram for the average value LSD (confidence interval 95%) for various algorithms.

Following Figure 7 clearly asserts the claim that the proposed ICA is superior over the other comparing algorithms. From another aspect in order to check the effect of the performance of other comparing algorithms, in this section basis for comparison is the size of problem.

CONCLUSIONS AND SUGGESTION

According to increasing variety of products and industries movement toward more production based on customer's order, we will face with increasing the number of various models; also, in increase the capacity of production is one of the assembly industries' targets. So, the usage of meta-heuristic methods in order to define the sequence of production related to common methods will be more noticeable.

Since the assembly time, the combination and quantity of needed parts for each kind of model are usually different; minimizing part usage in different workstation can be a vital factor in determining the process sequence of the products. This is a kind of NP-Hard problem and accessing the integer's linear programming is so difficult. Because of, this paper presented an imperialist competitive algorithm. This goal is one of the most important goals in MMAL, so that the usage rate is a measure of the company's ability to keep the schedule level, or evenly intermixed - keeping the raw materials for the different products arriving to the system at as constant a rate as possible.

In order to prove the efficiency of the proposed algorithm, three distinct kinds of methods have been used in the produced literature: innovative searching method of the GA and hybrid SA in the set of small, medium and large problems. The results indicated that the ICA excels comparing to the other comparing algorithms. We can offer some suggestions for the development of the present study. Introducing the problem in the dynamic and probability situations, applying changes in the problem assumption, using different operators in order to increase possibility of more searching of algorithm and leveling the convergence, applying new constraints on problem considering the existing constraints and employing the innovative methods such as the neural network, and colony combination of both.

REFERENCES

Aarts E, Lenstra JK (2003). Local Search in combinatorial optimization. Princeton University Press.

Aigbedo H, Monden Y (1997). A parametric procedure for multicriterion scheduling for just-in-time assembly lines. Int. J. Prod. Res. 35(9):2543–2564.

Atashpaz-Gargari E, Hashemzadeh F, Lucas C (2008). Designing MIMO PIID controller using colonial competitive algorithm: Applied to distillation column process. IEEE Congress on Evolutionary Computation (CEC 2008), No. 4631052, 1929–1934.

Bard JF, Shtub A, Joshi SB (1994). Sequencing mixed-model assembly lines to level parts usage and minimize line length. Int. J. Prod. Res. 32(10):2431–2454.

Behnamian J, Zandieh M, Fatemi GSMT (2009). Due window scheduling with sequence-dependent setup on parallel machines using three hybrid metaheuristic algorithms. Int. J. Adv. Manuf. Technol. 44(7-8):795-808.

Esmaeil AG, Farzad H, Ramin R, Caro L (2008). E“Colonial competitive algorithm: A novel approach for PID controller design in MIMO distillation column process”. Int. J. Intell. Comput. Cybernet. (IJICC) 1(3):337-355.

Hyun CJ, Kim YK, Kim Y (1998). A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. Comput. Oper. Res. 25(7-8):675–690.

Inman RR, Bulfin RL (1991). Sequencing JIT mixed-model assembly lines. Manage. Sci. 37(7):901–904.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). Optimization by simulated annealing. Science 220(4598):671–680.

Macaskill JLC (1973). Computer simulation for mixed-model production lines. Manage. Sci. 20(3):341–348.

Mansouri SA (2005). A multi-objective genetic algorithm for mixed-model sequencing on JIT assembly Lines. Eur. J. Oper. Res. 167(3):696-716.

McMullen PR (1998). JIT sequencing for mixed-model assembly lines with setups using Tabu search. Prod. Plan. Control 9(5):504-510.

McMullen PR (2001a). An efficient frontier approach to addressing JIT sequencing problems with setups via search heuristics. Comput. Ind. Eng. 41(3):335-353.

McMullen PR (2001b). A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem. Int. J. Prod. Econ. 72(1):59-71.

McMullen PR (2001c). An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. Artif. Intell. Eng. 15(3):309-317.

McMullen PR, Frazier GV (2000). A simulated annealing approach to mixed-model sequencing with multiple objectives on a JIT line. IIE Trans. 32(8):679-686.

Michalewicz Z (1992). Genetic algorithms + data structures = Evolution programs. Berlin: Springer.

Miltenburg J (1989). Level schedules for mixed-model assembly lines in just-in-time production systems. Manage. Sci. 35(2):192-207.

Miltenburg J, Sinnamon G (1989). Scheduling mixed model multi-level just-in-time production systems. Int. J. Prod. Res. 27(9):1487–1509.

Monden Y (1983). Toyota production system (2nd ed.). Norcross, GA: Institute of Industrial Engineers.

Okamura K, Yamashina H (1979). A Heuristic Algorithm for the Assembly Line Model Mix Sequencing Problem to Minimize the Cost of Stopping the Conveyor. Int. J. Prod. Res. 17(3):233-247.

Thomopoulos NT (1967). Line Balancing Sequencing for Mixed Model Assembly. Manage. Sci. 14(2):59-75.

Tian P, Ma J, Zhang DM (1999). Application of the simulated annealing algorithm to the combinatorial optimization problem with permutation property: An investigation of generation mechanism. Eur. J. Oper. Res. 118(1):81–94.

Walpole RE, Myers RH (1985). Probability and Statistics for Engineers and Scientists, Third ed. Macmillan, New York.

Yao X (1995). A new simulated annealing algorithm. Int. J. Comput. Math. 56(3-4):161–168.

Zaramdini W (2003). A Study of Just-In-Time Sequencing Procedures for Mixed-Model Assembly Lines Based on Genetic Algorithms. The Fifth Metaheuristics International Conference, Kyoto, Japan, August 25–28.

Zeramdini W, Aigbedo H, Monden Y (2000). “Bicriteria sequencing for just-in-time mixed-model assembly lines”. Int. J. Prod. Res. 38(15):3451–3470.

APPENDIX

Table 1. Problem sets used in the experiments.

Problem		DT	I	MPS	Solutions
Small problems	PS1	12	5	(8,1,1,1,1)	11,880
	PS2	12	5	(4,3,2,2,1)	831,600
	PS3	12	5	(3,3,2,2,2)	1,663,200
	PS4	15	5	(4,3,3,3,2)	126,126,000
	PS5	15	5	(3,3,3,3,3)	168,168,000
Medium problems	PM1	20	10	(7 5 1 1 1 1 1 1 1 1)	4.023×10^{12}
	PM2	20	10	(6,5,2,1,1,1,1,1,1,1)	1.408×10^{13}
	PM3	20	10	(5,5,3,1,1,1,1,1,1,1)	2.816×10^{13}
	PM4	20	10	(4 4 4 2 1 1 1 1 1 1)	2.112×10^{15}
	PM5	20	10	(2,2,2,2,2,2,2,2,2,2)	2.376×10^{15}
Large problems	PL1	100	15	(30,30,15,10,5,1,1,1,1,1,1,1,1,1,1)	2.329×10^{72}
	PL2	100	15	(25,25,20,15,5,1,1,1,1,1,1,1,1,1,1)	1.016×10^{72}
	PL3	100	15	(20,20,15,15,10,6,6,1,1,1,1,1,1,1,1)	4.901×10^{84}
	PL4	100	15	(15,15,15,10,10,10,10,5,4,1,1,1,1,1,1)	8.357×10^{91}
	PL5	100	15	(7,7,7,7,7,7,7,7,7,7,6,6,6,6,6)	6.334×10^{103}

Bill of assembly time configuration

Product type	Part type							
	a	b	c	d	e	f	g	h
A	0	1	1	1	0	1	0	0
B	0	1	0	0	0	0	0	0
C	0	0	0	1	0	0	0	1
D	1	0	1	0	0	0	0	1
E	1	0	0	0	1	0	1	0
F	0	1	0	0	0	0	1	0
G	0	0	1	0	0	1	0	0
H	0	0	0	1	0	0	1	0
I	1	0	0	0	1	1	0	0
J	0	0	0	0	1	0	0	1
K	1	0	0	1	0	0	0	1
L	1	0	1	0	0	1	0	1
M	1	0	1	0	1	0	1	0
N	0	1	0	1	0	1	1	0
O	0	1	1	0	0	1	0	0

The number in the table denotes the demand for that particular product.