Evolutionary Approaches to the Linear Machine Layout Problem

Salman Mohagheghi*, George Georgoulas, Chrystostomos Stylios***, Peter Groumpos******

**ABB Inc[1], US Corporate Research Center, Raleigh, NC 27606 USA (salman.m@ieee.org)*
***Dept. of Computer Applications in Finance and Management, TEI of Ionian Islands, Lefkas, Greece (georgoul@teiion.gr)*
**** Dept. of Informatics and Communications Technology, TEI of Epirus, Kostakioi, Artas, Greece (stylios@teiep.gr)*
*****Lab. of Automation and Robotics, Dept. of Electrical and Computer Eng. Patras, Greece (groumpos@ece.upatras.gr)*

Abstract: Flexible Manufacturing Systems (FMSs) cope with multi-product, usually small sized production. In this research work we investigate the use of evolutionary methods to solve the linear or single-row layout problem, which is the most commonly implemented layout in FMSs. Three different approaches, based on Ant Colony Optimization, Genetic Algorithms and Particle Swarm Optimization are tested. The experimental results show that a near optimal solution can be found for all three methods, exploiting only a small portion of the feasible solution space, pinpointing once more the merit of using evolutionary algorithms to tackle difficult combinatorial problems.

Keywords: Flexible Manufacturing Systems, Ant Colony Optimization, Genetic Algorithms, Particle Swarm Optimization.

1. INTRODUCTION

Flexible Manufacturing System (FMS) is a system in which a set of machines and a flexible material-handling system - usually automated guided vehicles and granty robots- are integrated using a central computer. FMS is different from the classical machining systems due to higher degree of automation and smaller number of machines. (Kusiak, 1990).

But the FMS layout design is even more crucial than in conventional manufacturing. While a variety of methods that implement complex networks and layouts are available, the linear or single-row layout is the most commonly implemented layout due to its simplicity. Different criteria have been used in order to select the “optimal” arrangement of a number of machines in a linear production line. All of them result in a combinatorial optimization problem.

Two classes of algorithms are available for the solution of combinatorial optimization problems: exact and approximate algorithms. *Exact algorithms* try to find the truly optimal solutions. Despite their recent success, for many *NP*-hard problems, their applicability is often limited to rather small instances. *Approximate algorithms* trade optimality for efficiency. Their main advantage is that, in practice, they often find reasonably good solutions in a very short time. Algorithms of this type are loosely called *heuristics.*

In this research work, we propose the use of evolutionary algorithms, which are part of the larger class of heuristic methods, to solve the linear machine layout problem. Three different members of the evolutionary family, namely the Ant Colony Optimization (ACO), the Genetic Algorithms (GAs) and the Particle Swarm Optimization (PSO) are tested achieving very promising results.

The rest of this paper is organized as follows: Section 2 presents the formulation of the Linear Machine Layout problem. In Section 3 the necessary material for each one of the three approaches is briefly summarized. In Section 4, the application results of these evolutionary algorithms to a specific setup are presented and Section 5 concludes the paper with some comments and remarks for future research.

2. LINEAR MACHINE LAYOUT PROBLEM

There are many shapes of linear layouts, such as straight line, circular loop, U-shape and serpentine line (Haragu and Kusiak, 1998) (Figure 1). The configuration of the production line depends heavily on the material-handling system. Apart from its configuration, the production line is characterized by the flow of material as unidirectional or bidirectional. In the latter, four different types of flow movement can occur (Figure 2) (Ponnambalam and Ramkumar, 2001): a) Repeated operations, b) In-sequence operations, c) Bypassing operations and d) Backtracking operations.

The most desirable flow is the in-sequence operation due to its unidirectional movement. Backward flow is the least desirable since it causes additional costs and complicates the flow more than the forenamed flow movements. The ideal scenario would include only in-sequence moves. In practice, however, bypassing and backtracking of jobs as they pass down the line is inevitable. The designer of a single-row layout has to find the optimal arrangement of machines for such a production line. The optimality depends on the criteria and the restrictions that are posed. There are four criteria,

[1] During this work S. Mohagheghi was with the Georgia Institute of Technology, Atlanta, GA, USA.

which a designer could take into account (Ponnambalam and Ramkumar, 2001): a) minimization of the number of backtracking movements, b) minimization of total backtracking flow distance, c) maximization of in-sequence movements, and d) minimization of the flow distance. Obviously different criteria/objectives will lead to different optimal settings.

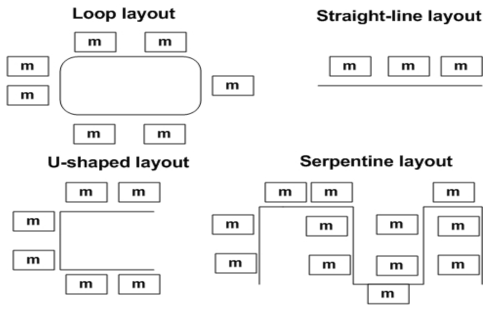

Fig.1. Alternative configurations of single-row layouts

Carrie (1975) was the first to study the linear layout problem and several approaches have been proposed since then (Aneke and Carrie, 1986; Lee, 1991; Kouvelis and Chiang, 1992; Sarker *et al.*, 1994; You-Dong, 1997; Ponnambalam and Ramkumar, 2001). In this paper, we investigate the solution to the "minimal backward-flow" model, i.e. minimization of total backtracking flow distance, which is presented in more detail in the following section.

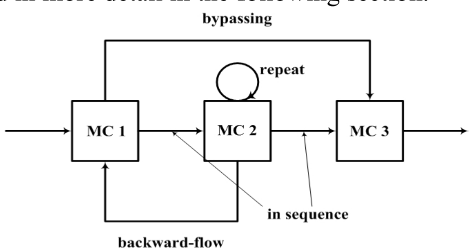

Fig.2. Four different flow movements in a linear layout

2.1 Minimal backward-flow model.

This model is developed by trying to minimize the total amount of backward flows for a production cell, as it is indicated by its name. For this model, we make the following assumptions (Sarker et al. 1994): a) Just one machine of each type is allowed in the line (no duplications of machines are allowed), b) The cost of material flows is proportional to the number of parts and the distance of flows, and c) Each machine is considered as a point and the distance between machines is "1" (unit distance).

The distance between the initial input point of parts and the first machine is also considered equal to 1. In Figure 3 the adopted conventions are depicted. Therefore, the problem can be described as follows:

Having M machines and n items of parts to be produced and for each item a corresponding demand d_j ($j = 1, 2, \ldots n$), place the machines in such an order so as to minimize the backward flows. As aforementioned, the quantity that has to be minimized is the total number of backtracking steps. Thus, for this problem, following the notation of (Sarker, *et al.*, 1994), we seek to minimize:

$$TB = \sum_{j=1}^{M}\sum_{i=1}^{M} r_{ij} b_{ij} = \mathbf{R}\cdot\mathbf{B} \qquad (1)$$

where M is the number of machines, $\mathbf{R} = \left[r_{ij}\right]_{M\times M}$ is the requirement matrix, $\mathbf{B} = \left[b_{ij}\right]_{M\times M}$ is the backtrack matrix, r_{ij} is the number of total moves from machine i immediately to machine j and b_{ij} is the number of backtrack steps from machine i to machine j and $(\cdot)$ defines element by element multiplication. For a more rigorous analysis the interested reader can refer to (Sarker, *et al.*, 1994).

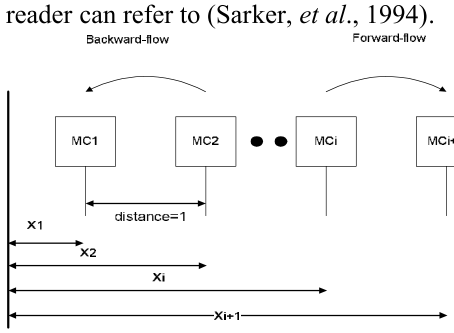

Fig. 3. Linear machine layout.

For example, suppose that we have 3 machines and we want to produce 2 items (10 parts of the 1st item and 15 parts of the 2nd item). In order to produce those parts, we have to use the 3 machines in the following order:

Item 1 1-2-3-2-3-1 (10 parts)
Item 2 3-2-1-3-2-3-1-2 (15 parts)
and the machine layout is 1-2-3.

In order to calculate the total cost, we construct the matrices **R**, **B** we multiply them in an element by element manner and we sum them according to equation 1.

$$\mathbf{B} = From \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \quad To \\ \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix} \end{array} \qquad \mathbf{R} = From \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \quad To \\ \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 25 & 15 \\ 15 & 0 & 35 \\ 25 & 40 & 0 \end{bmatrix} \end{array}$$

Therefore the total backtracking cost is: $TB = 1\times 15 + 2\times 25 + 1\times 40 = 105$.

It is obvious that the requirement matrix remains constant, whereas the backtrack matrix changes according to the arrangement of the machines. This function is both complex and difficult to estimate before all the machines are in place.

3. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms have been successfully used to tackle difficult real-life problems. Almost all of them are based on the use of a population of candidate solutions which are given different names depending on the specific algorithm (i.e. chromosomes, particles, ants, etc) and a number of mechanisms that allow the individual solutions to exchange information trying to guide the search towards more promising regions. The oldest and most prominent member of the evolutionary family is the GA paradigm proposed by Holland (Holland 1975). While the GA is loosely based on the genetics the other two paradigms investigated in this paper are based on the complex behaviour emerging from the interaction of single individuals of social species. Therefore the ACO was inspired by the behaviour of real ants (Dorigo *et al.*, 1991) and the PSO by flight of birds in a flock (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995).

3.1 Ant Colony Optimization (ACO)

Ant algorithms were introduced in early 1990's (Dorigo *et al.*, 1991) and they were combined with a multi-agent approach for difficult combinatorial optimization problems.

ACO is a stochastic search method based on the indirect communication of a colony of artificial ants, mediated by artificial pheromone trails. The pheromone trails in ACO serve as distributed numerical information used by the ants to probabilistically construct solutions for the specific problem. The modification of the pheromone trails by the ants during the algorithm's execution in order to reflect their search experience along with an evaporation mechanism is one of the two mechanisms helping the ants to find promising areas in the search space. The other one involves the use of heuristic information to grade the available states that the ant can move during each time step.

The employment of the ACO algorithm for this particular problem is performed by considering that the artificial ants are moving from one machine to another building a path, which is a candidate solution for this problem. That is, starting from a machine, the ant proceeds by selecting the machine to be put next in the layout. The selection of the next machine has to be done based on a trade-off between the pheromone (τ), which is deposited on the arcs connecting the current machine with the other machines and a heuristic (*n*) function, which measures locally the quality of the machine that can be added to the current partial solution.

Adopting the Ant Colony System (ACS) approach (Dorigo and Gambardella, 1997) we construct a candidate solution using the following "tour" formulation.

3.1.1 "Tour" construction

When the *k*-th ant is located at machine *i*, it chooses to move to machine *j*, according to the so-called *pseudorandom proportional rule,* given by:

$$j = \begin{cases} \max_{u \notin J_i^k} \left\{ [\tau_{iu}(t)]^{\alpha} \cdot [\eta_{iu}(t)]^{\beta} \right\}, if \quad q \le q_0 \\ J \qquad\qquad\qquad\qquad , \quad if \quad q > q_0 \end{cases} \tag{2}$$

where q_0 is a parameter that it is selected by the user in the interval [0,1], and *J* is a machine that has not been used so far and is selected with probability:

$$p_{iJ}^k(t) = \frac{[\tau_{iJ}(t)]^{\alpha} \cdot [\eta_{iJ}(t)]^{\beta}}{\sum_{l \notin J_i^k} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}(t)]^{\beta}}, \quad if \quad J \notin J_i^k \tag{3}$$

Among the other candidates, J_i^k denotes the tabu list (forbidden list), which contains all the machines that have been so far used including the current machine i, η_{ij} is a heuristic value that is available *a priori*, τ_{ij} is the pheromone trail, α and β are two parameters, which determine the relative influence of the pheromone trail and the heuristic information. In other words, with probability q_0 the ant makes the best possible move as indicated by the learned pheromone trails and the heuristic information (in this case, the ant is exploiting the learned knowledge), while with probability (1-q_0), it performs a biased exploration of the arcs. Tuning the parameter q_0 allows modulation of the degree of exploration and the choice of whether to concentrate the search of the system around the best-so-far solution or to explore other tours. As it is obvious the choice depends heavily on the quantities $\eta_{ij}(t)$ and $\tau_{ij}(t)$.

3.1.2 *Pheromone*

The quantity $\tau_{ij}(t)$ corresponds to the directional "arc" connecting machine *i* to machine *j*. This quantity changes after the completion of any search for all ants. At the first step of the algorithm all pheromone trails are initiated to a value τ_0. For a detailed representation of the pheromone settings as well as the mechanisms employed to modify the pheromone trails see (Papadimitrou *et al.,* 2006).

3.1.3 Heuristic information (Visibility)

The heuristic function (which some times is referred as visibility, taking its name from its original use in the context of the Travelling Salesman Problem (TSP)) is a quantity, which in our problem (a static problem) doesn't change over iterations ("iteration-invariant").,Therefore a more proper notation is: $\eta_{ij} = 1/d_{ij}$, where d_{ij} in the original implementation of the TSP simply denotes the distance between town *i* and *j*. In our case, d_{ij} corresponds to a "local" cost in accordance with the total cost function that has to be minimized. This means that d_{ij} measures the immediate cost that we have to pay by placing machine *j* after machine *i.* This is calculated by summing all the backflows created by this arrangement within a unity distance, i.e. by restricting our search to only adjacent steps in the production phase. In other words, d_{ij} is equal to the element r_{ji} of the requirement matrix.

3.2 Genetic Algorithms (GA)

The idea behind GAs is to emulate what nature does; in other words, GAs try to model genetic recombination, mutation and selection. They are a class of general purpose search methods balancing exploration and exploitation of the search space. They exploit the use of a population of chromosomes (candidate solutions) and an evolution process running on the population pushing them to search through the solution space in an effective manner. A GA has the following five components (Michalewicz 1996): a) A genetic representation for potential solutions to the problem,, b) A way to create an initial population of potential solutions, c) An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness", d) Genetic operators that alter the composition of children (usually crossover and mutation), and e) Values for various parameters that the genetic algorithm uses.

GAs were initially developed to solve real valued problems using binary representation. Thus, the genetic operators (crossover and mutation) were explicitly tailored for binary strings and later revised to account for real valued representations. However when dealing with permutation problems, both the representation and the genetic operators need to be selected with extra caution (Michalewicz, 1996).

For the problem at hand, in analogy to the TSP the most natural representation is the "path representation". For example two candidate solutions for a 9-machine problem would be represented as

(1 2 3 4 5 6 7 8 9)

(4 5 2 1 8 7 6 9 3)

It is obvious that the standard crossover and mutation operators cannot be applied since we have to make sure that each "location" is represented only once in each chromosome. Different crossover operations have been proposed as well as different mutation operations that guarantee that feasible offsprings will be created (Michalewicz, 1996).

For this work we employed the cycle crossover CX (Michalewicz, 1996). The cycle crossover preserves the absolute position of the elements (cities, machines, etc.) in the parent sequence and this might be beneficial since in our case – unlike the TSP – the absolute position does matter. For the 2 chromosomes listed above the CX would start at the left and choose the first machine from the first parent to produce the first offspring (o_1)

o_1: (1 x x x x x x x x)

Since we want every machine to be taken from one of its parents the next machine to be considered is machine 4 (just bellow the selected machine 1) leading to

o_1: (1 x x 4 x x x x x)

This, in turn, implies machine 8 (cited bellow machine 4)

o_1: (1 x x 4 x x 8 x)

Following this rule we select machines 3 and 2

o_1: (1 2 3 4 x x 8 x)

The selection of machine 2 means that we should choose machine 1 from the first string which is not possible since machine 1 has already been selected as the first machine. Thus we have completed a *cycle*. The remaining of the positions are filled from the second string (parent):

o_1: (1 2 3 4 7 6 9 8 5)

Similarly

o_2: (4 1 2 8 5 6 7 3 9)

Following the same sequence of thoughts the mutation operator also has to be redefined. In our case, we randomly select a chromosome and exchange the integers between 2 randomly selected places.

3.3 Particle Swarm Optimization (PSO)

PSO is a population based stochastic optimization technique developed by Kennedy and Eberhart (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995) inspired by the social behavior of animals such as flocking or fish schooling. Since its introduction, PSO has found many applications in solving optimization problems in real number spaces (Valle *et al*. 2008).

A potential solution to a minimization (or maximization) problem is represented by a particle having coordinates x_{id} and rate of change v_{id} in the D-dimensional space. In its original formulation, the updates of the particles are accomplished according to the following equations:

$$v_{id}(t+1) = v_{id}(t) + n_1 r_1 [p_{id} - x_{id}(t)] + n_2 r_2 [p_{nd} - x_{id}(t)] \quad (4)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (5)$$

where $v_{id}(t)$ is the current velocity of the ith particle, $x_{id}(t)$ is the current position of the ith particle, p_{id} is the particle's locations at which the best fitness has been achieved so far and p_{nd} is the best particle among the neighbours at which the best fitness has been achieved so far; r_1 and r_2 are two independently generated random numbers—–uniformly distributed in [0, 1]- and n_1, n_2 are learning factors. Since its introduction variations of the above formulation have been proposed (Valle *et al*. 2008), improving the performance of the continuous version of the PSO.

Apart from the well known continuous PSO, there is also a discrete binary version of the PSO algorithm (Kennedy and Eberhart, 1997). In the binary version, the formula for the velocity remains unchanged, except for the fact that the particle positions p_{nd}, p_{id}, $x_{id}(t)$ can only take binary values. So the probability of an individual to take a value of 1 can be modeled as (Valle *et al*. 2008):

$$P(x_{id} = 1) = f(x_{id}(t-1), v_{id}(t-1), p_{id}, p_{nd}) \quad (6)$$

In this model, the probability that the i^{th} individual chooses 1 for the d^{th} bit in the string, is a function of the previous state of the bit and the velocity which is the measure of the individual's tendency to choose 1 or 0. The probability above also implicitly depends on p_{id} and p_{nd}. Mathematically, v_{id} determines a threshold in the probability function, and therefore should be bounded in the range of [0,1]. This threshold can be modeled using the sigmoid function:

$$S(v_{id}(t)) = \frac{1}{1+\exp(-v_{id}(t))} \tag{7}$$

Using (7), the state of the d^{th} position in the string for the i^{th} individual at time t can be expressed as:

$$\text{if } \xi_{id} < S(v_{id}(t)) \text{ then } x_{id}(t) = 1, \text{ else } x_{id}(t) = 0 \tag{8}$$

Where ξ_{id} is a random number with a uniform distribution in the range of [0,1]. This procedure is repeatedly iterated over each dimension and for all individuals, testing if the current value x_{id} results in a better evaluation than p_{id}, in which case its value will be stored as the best individual state.

Clearly, the sociocognitive concepts of particle swarm are included in the function for v_{id} (according to (4)), which indicates that the tendency of each individual towards success is adjusted according to its own experience as well as that of its neighborhood. In all equations, some considerations have to be made in order to adjust the limits of the parameters.

In a more general case, when integer solutions (not necessarily 0 or 1) are needed, the optimal solution can be determined by rounding off the real optimum values to the nearest integer. Basic PSO equations, developed for a real number space, are used to determine the new position for each particle. Once $x_i(t)\in\Re^n$ is determined, its value in the d^{th} dimension is rounded to the nearest integer value using the following equation:

$$\begin{aligned} X_{id}(t) &= [x_{id}(t)], \quad d: 1 \rightarrow n \\ x_{id}(t) &\in \Re \textit{ and } X_{id}(t) \in Z \end{aligned} \tag{9}$$

The results presented by some researchers using integer PSO indicate that the performance of the method is not affected when the real values of the particles are truncated (Valle *et al*. 2008). Moreover, integer PSO has a high success rate in solving integer programming problems even when other methods, such as Branch and Bound, fail.

4. RESULTS

In order to test the utility of the evolutionary methods, we examine a well known FMS problem, where all the involved quantities are generated randomly and are summarized in the following Table 1.

For this setup the optimal arrangement of the machines is: 7, 8, 3, 2, 6, 9, 5, 1, 4 with a total number of backtracking steps equal to 2923 (total cost). The total number of possible solutions is 9!= 362880. On the other hand, the worst case scenario would be to arrange the machines in the following order 2, 1, 4, 7, 5, 6, 9, 8, 3 with a total cost of 4980.

Table 1: Eight Processes, with their corresponding routes and demands

Process #	Route information	# of parts
1	[1 6 8 9 3 5 7 4 3 8 6 8 2 3]	8
2	[2 6 1 8 9 5 2 1 6 2 9 5 6 8 4 3 4]	22
3	[8 7 3 4 1 8 5 6 2 3 1]	33
4	[3 4 6 9 2 1 7 2 8 1]	12
5	[8 7 3 1 4 1 5 6 2 9 3 1]	14
6	[9 8 7 2 3 4 5 1 5 7 6 2 3 1]	23
7	[3 7 9 4 9 2 5 1 7 8 2 8 6 3 2]	39
8	[3 7 2 4 6 2 9 1 9 5 8 3 4]	28

4.1. Ant Colony

Our "colony" consisted of 9 ants (equal to the number of the machines following the approach to solve the TSP) and in each experiment the algorithm was let to run for 1000 iterations. Because the algorithm is stochastic in nature, we repeated the experiments 10 times and calculated the average performance. Figure 4 depicts the evolution of the best solution for each one of the 10 trials along with their average.

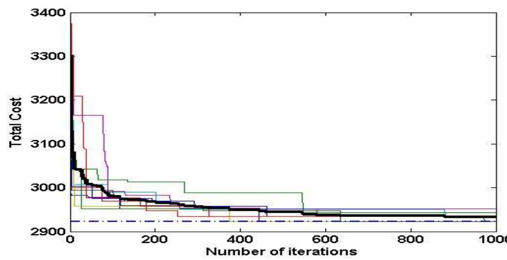

Fig. 4. The evolutions of C^{bs} for 10 different runs of the experiment. The thick line corresponds to the average of those 10 trials, while the dashed line on the bottom marks the global best value.

The average cost achieved was 2932 and the algorithm 4 times out of 10 also found the global optimal solution (2923).

4.2. GA

In order to have an equal number of function evaluations the GA consisted of 9 chromosomes and was let to run for 1000 generations. The experimental setup was also executed 10 times end the results are depicted in Figure 4. The average cost achieved was 2928 and the algorithm 9 times out of 10 found the global optimal solution.

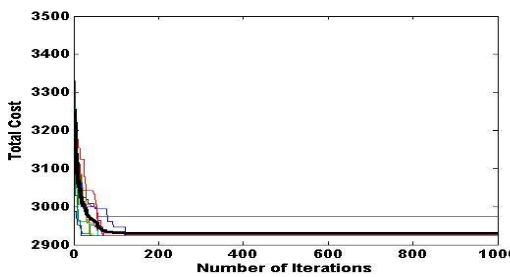

Fig. 5. The evolutions of C^{bs} for 10 different runs of the experiment. The thick line corresponds to the average of those 10 trials.

4.3. PSO

The particle swarm implementation consisted of 50 particles since the original trials with 9 particles didn't perform satisfactory. Each particle has a length of 9, with each entry indicating the position of each machine in the overall layout. In order to prevent deriving the same location for different machines, particles with one or more pairs of equal entries are highly penalized. Similar to the previous cases, the PSO was tested 10 times and each time it let to run for 1000 iterations. One out of ten times the algorithm converged to the global minimum of 2923, with the average of the 10 runs being 3008. Figure 6 illustrates the total cost for the various runs of the PSO algorithm.

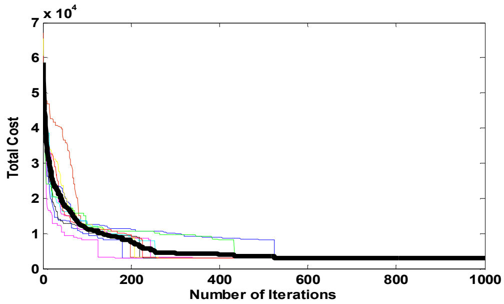

Fig. 6. The evolutions of C^{bs} for 10 different runs of the experiment. The thick line corresponds to the average of those 10 trials.

5. CONCLUSIONS

In this work, we investigated the solution of a simplified linear layout problem based on evolutionary approaches. The results are encouraging indicating that heuristic approaches can be used for the design of modern manufacturing systems, and with further investigation could be probably used for more complicated problems (general machinery layout problems). It was shown, that by exploring only a small number of candidate solutions, we were able to find a good (near-optimal) solution without even optimising the parameter settings of the algorithm.

Among the three algorithms employed, the GA performed the best, slightly outperforming the ACO approach. The PSO method didn't perform as good (however without failing to find near-optimal solutions) probably due to the nature of the problem which is not a simple permutation problem since the choice of the first positioned machine is also important. In future work the authors will also examine, validate and compare the performance of some PSO variants and structures, for instance experimenting with different neighborhood structures for defining the global optimum (Valle *et al*. 2008).

To summarize, it is apparent that these previously applied evolutionary optimization methodologies seem to be viable solutions for linear layout problem. But further experimentation is needed before reaching a conclusion about the superiority of one method over the others and its applicability for general purposes.

REFERENCES

Aneke, N. A. G., and Carrie, A. S. (1986). A design technique for the layout of Multi-product flow lines. *International Journal of Production Research*, volume 24, pp. 471-481.

Carrie, A. S. (1975). Layout of Multi-product lines. *International Journal of Production Research*, volume 13, pp. 541-575.

Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the traveling salesman problem. *Biosystems*, volume 43, no 2, pp. 73-81.

Dorigo, M., Maniezzo, V. and Colorni A. (1991). Positive feedback as a search strategy. *Technical report 91-016*, Dipartimento di Electronica, Politecnico di Milano, Milan.

Heragu, S. S. and A. Kusiak (1998). Machine layout problem in flexible manufacturing systems. *Operations Research*, volume 36, no 2, pp. 258-268

Holland, J. H. (1975), Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI.

Kennedy, J., and Eberhart, R. C. (1995), Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks*, Piscataway, NJ, pp 1942-1948.

Kennedy, J. and Eberhart, R. C. (1997), A discrete binary version of the particle swarm algorithm, *Proceedings of the Conference on Systems, Man, and Cybernetics*, Piscataway, NJ, pp 4104-4109

Kouvelis, P. and Chiang, W. C. (1992). A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *Int. J. Prod. Res.* volume 30, no 4, pp 717-732.

Kusiak A. (1990) *Intelligent manufacturing systems*. Englewood Cliffs, NJ. Prentice-Hall.

Michalewicz, Z (1996). *Genetic Algorithms+Data structures=Evolution Programs*. 3rd edition, Springer, Berline.

Ponnambalam, S. G. and Ramkumar V. (2001). A genetic Algorithm for the Design of a Single-Row Layout in Automated Manufacturing Systems. *Int J. Adv. Manuf. Technol.,* volume 18, pp. 512-519.

Papadimitriou A., Georgoulas, G., Stylios, C. and Groumpos, P., (2006), Ant colony algorithm for optimal arrangement of linear machine layout, *Proceedings of 1st IFAC Workshop on Applications of Large Scale Industrial Systems (ALSIS'06)*.

Sarker, B. R., Wilhelm, W. E. and Hogg, G. L. (1994). Backtracking and its Amoebic Properties in One-dimensional Machine Location Problems. *J. Opl Res. Soc.,* volume 45, no 9, pp 1024-1039.

You-Dong, W. (1997). A linear programming approach to linear machine layout problem. *The Journal of Industrial Mathematics Society*, volume 47, no 2, pp. 59-69.

Valle Y., Venayagamoorty, G. K., Mohagheghi, S., Hernandez-Mejia J. C., and Harley, R. G. (2008). "Particle Swarm Optimization- Basic Concepts, Variants and Applications in Power Systems," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 171-195.