Different Chromosome-based Evolutionary Approaches for the Permutation Flow Shop Problem

Krisztián Balázs[1], Zoltán Horváth[2], László T. Kóczy[1,3]

[1] Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Budapest, Hungary, balazs@tmit.bme.hu

[2] Department of Mathematics and Computational Sciences, Széchenyi István University, Győr, Hungary, horvathz@sze.hu

[3] Department of Automation, Széchenyi István University, Győr, Hungary, koczy@sze.hu

Abstract: This paper proposes approaches for adapting chromosome-based evolutionary methods to the Permutation Flow Shop Problem. Two types of individual representation (i.e. encoding methods) are proposed, which are applied on three different chromosome based evolutionary techniques, namely the Genetic Algorithm, the Bacterial Evolutionary Algorithm and the Particle Swarm Optimization method. Both representations are applied on the two former methods, whereas one of them is used for the latter optimization technique. Each mentioned algorithm is involved without and with local search steps as one of its evolutionary operators. Since the evolutionary operators of each technique are established according to the applied representation, this paper deals with a total number of ten different chromosome-based evolutionary methods. The obtained techniques are evaluated via simulation runs carried out on the well-known Taillard's benchmark problem set. Based on the experimental results the approaches for adapting chromosome based evolutionary methods are compared to each other.

Keywords: Chromosome-based evolutionary methods; Memetic algorithms; Combinatorial optimization; Permutation Flow Shop Problem

1 Introduction

One of the most intensively studied combinatorial optimization problems is the Permutation Flow Shop Problem (PFSP) [1]. In this problem, there are given *n* jobs and *m* machines. All the jobs should be processed by all the machines one after another. The machines are deployed in a line and a machine can handle one single job at once, that is, the process of the jobs is pipeline-like. There is also given an *n*-by-*m* processing time matrix defining the necessary amount of time a job has to stay on a machine, for each job-machine pair. A job can be processed on any machine only if the machine is free (the preceding job has finished on the machine) and the job has already been processed on the preceding machine.

The task is to find a permutation (a sequence) of the jobs, in case of which the total processing time of all the jobs on all the machines (i.e. the so-called makespan) is minimal.

This problem is known to be NP-hard [2], thus there are no efficient algorithms to exactly solve this task (and there is not much hope of finding one). This means that every method guaranteeing optimal solutions has impractically long computational time for even moderate problem sizes. Hence, only heuristics resulting in so called quasi-optimal solutions are viable. Over the past few decades, a number of such heuristics have been invented and published (e.g. [3]-[5]).

Since due to the nature of the PFSP problem these heuristics cannot be evaluated analytically, their evaluation and their comparison to other techniques can be made experimentally, i.e. based on results of simulation runs carried out on standard reference tasks, called benchmark problems. Several such comparisons have been made involving a large part of the so far proposed methods (e.g. [3]-[5]). These comparative studies are mostly based on the well-known Taillard's benchmark problem set [6].

This paper proposes approaches for adapting chromosome based evolutionary methods to the Permutation Flow Shop Problem. The proposal includes two types of individual representation (i.e. encoding method): a permutation and a real value based one. They are applied on three different chromosome based evolutionary techniques, namely the Genetic Algorithm [7], the Bacterial Evolutionary Algorithm [8] and the Particle Swarm Optimization [9] method. Both representations are applied on the two former methods, whereas the real value based one is used for the latter optimization technique. Each mentioned algorithm is involved without and with local search steps as one of its evolutionary operators. Since the evolutionary operators of each technique are established according to the applied representation, this paper deals with a total number of ten different chromosome-based evolutionary methods.

The obtained techniques are evaluated via simulation runs carried out on the above mentioned Taillard's benchmark problem set. Based on these experimental results, the methods, and thus the chromosome-based evolutionary technique adapting approaches themselves, are compared to each other.

The next section gives a formal definition to the PFSP problem. Within this, the search space and the makespan function as the objective function are defined.

Then, the third section gives a brief overview of the chromosome-based evolutionary techniques being adapted to the PFSP task. The basic concept and the main steps of the algorithms are also presented. The new encoding approaches for the PFSP problem are proposed in section four. After that, the evolutionary operators constructed based on the newly proposed individual representations are described. The sixth section enumerates the algorithms, which can be established by using the discussed approaches and which are compared via simulation runs. The experimental results and the observed characteristics are discussed in section seven. Finally, in the last section our work is summarized and some conclusions are drawn.

2 The Permutation Flow Shop Problem

As was described in the Introduction, in this problem there are given the number of jobs *n*, the number of machines *m* and an *n*-by-*m* processing time matrix **P** defining the necessary amount of time a job has to stay on a machine, for each job-machine pair. That is, the elements of the matrix are positive and an element $p_{i,j}$ denotes the time the i^{th} job stays on the j^{th} machine.

All the jobs should be worked by all the machines one after another. The machines are deployed in a line and a machine can handle one single job at once. That is, the process of the jobs is pipeline-like. A job can be processed on a machine only if the machine is free (the preceding job has finished on the machine) and the job has already been processed on the preceding machine.

The task is to find a permutation (an order) of the jobs, in case of which the total processing time of all the jobs on all the machines (i.e. the so called makespan) is minimal.

For example, if there are three jobs the permutation *(2,3,1)* denotes the case when the second job goes first, the third goes next, and finally the first goes last. This should not be confused with another interpretation of a permutation, where the same permutation would mean a case when the first job goes second, the second one goes third and the last one goes first. In our interpretation this latter will be referred as the 'inverse' of the permutation defined above. (It can be easily seen that this 'inverse' is the algebraic inverse of the permutation; consequently, the inverse of the inverse of the permutation is the permutation itself.)

Clearly, the search space is the set of the *n*-order permutations S_n, and the objective function is defined over this search space and its range is the set of positive numbers R^+.

Formally, the objective or makespan function *f* can be defined as follows (see e.g. [1]):

$$f : S_n \to R^+$$
$$f(\sigma) = t(n, m, \sigma)$$

$$\begin{aligned} &t(0, j, \sigma) \equiv 0 \\ &t(i, 0, \sigma) \equiv 0 \\ &t(i, j, \sigma) = \max(t(i, j-1, \sigma), t(i-1, j, \sigma)) + p_{\sigma(i), j} \end{aligned} \tag{1}$$

where $p_{\sigma(i),j}$ and $t(i,j,\sigma)$ denote the processing and the completion times of the ith job of the σ permutation on the jth machine, respectively.

The task is to find a σ permutation for which the makespan is optimal (i.e. minimal).

3 Overview of the Evolutionary Techniques Applied

A famous, frequently studied and applied family of iterative stochastic optimization techniques is called chromosome based evolutionary algorithms. These methods, like the Genetic Algorithm (GA) [7] or the Bacterial Evolutionary Algorithm (BEA) [8], imitate the abstract model of the evolution of populations observed in nature. Their aim is to change the individuals in the population (set of individuals) by the evolutionary operators to obtain better and better ones. The goodness of an individual can be measured by its ‘fitness’. If an individual represents a candidate solution for a given problem, the algorithms try to find the optimal solution for the problem. Thus, in the case of optimization problems, the individuals represent elements of the search space and the fitness function is a transformation of the objective function. If an evolutionary algorithm uses an elitist strategy, it means that the best ever individual will always survive and appear in the next generation. As a result, at the end of the algorithm the best individual will represent the (quasi-) optimal element of the search space.

The individuals are usually represented by chromosomes (this is why these methods are called chromosome-based evolutionary algorithms), which are most often vectors holding numbers in their components (i.e. in their genes). The manner in which the individuals are represented as chromosomes is the encoding method.

The steps of the algorithms changing the chromosomes, and thus the candidate solutions, are called evolutionary operators. The evolutionary operators are in strong connections with the encoding technique, since the encoding determines the form of the chromosomes the operators work with.

It is quite obvious that besides the formation of the skeleton of the evolutionary algorithm, the design of the encoding method and the evolutionary operators also play key roles in the efficient government of the evolution process.

There are a huge number of chromosome based evolutionary algorithms. Some of them will be presented below; these are the ones that were investigated in our work.

3.1 Genetic Algorithm

One of the most (if not the most) widely applied chromosome based evolutionary techniques is the Genetic Algorithm (GA) [7]. It comprises the following steps:

1 Initialization:
An initial population is created by selecting random elements of the search space according to some distribution, or by using an initial heuristic.

2 Selection:
Individuals are selected according to their fitness values. The higher fitness value an individual has, the bigger its probability to be selected. There are a number of selection methods, e.g. roulette wheel technique, or stochastic universal sampling.
The selected individuals are called parents.

3 Crossover:
Pairs are formed from the set of parents and a random point of the chromosome is selected for each pair. Then the parents change the sequence of their genes between each other after the selected point. The resulting individuals are called offspring.

4 Mutation:
The genes of each offspring are mutated with a certain probability and take new random values.

5 Substitution:
The offspring are substituted in the population, i.e. they overwrite individuals in it. The individuals to overwrite are selected according to their fitness values. However, unlike in the selection step, in this case the higher fitness value an individual has, the smaller its probability to be selected. If the above mentioned elitist strategy is applied, the best individual will not be overwritten. With minor modifications, the same algorithms can be used for the selection of individuals to overwrite as in the selection step.

The main iteration loop of the algorithm contains steps 2 – 5. A single iteration is called generation. The algorithm stops if at the end of a generation one of the termination criteria fulfills (generation limit reached, time limit exceeded, etc.). After termination the best individual represents the quasi-optimal solution.

3.2 Bacterial Evolutionary Algorithm

Compared to GA, a somewhat different evolutionary technique is called the Bacterial Evolutionary Algorithm (BEA). This algorithm was introduced by Nawa and Furuhashi in [8]. The first version of this algorithm was called the Pseudo-Bacterial Genetic Algorithm (PBGA) [10], which proposed a modified mutation operator called bacterial mutation, based on the natural phenomenon of microbial evolution. The Bacterial Evolutionary Algorithm introduced a new operator called the gene transfer operator. While PBGA incorporates bacterial mutation and crossover operator, the BEA substitutes the classical crossover with the gene transfer operation. Both of these new operators were inspired by bacterial evolution. Bacteria can transfer genes to other bacteria, and thus gene transfer allows the bacteria to directly transfer information to the other individuals in the population.

BEA comprises the following steps:

1 Initialization:
An initial population is created by selecting random elements of the search space according to some distribution, or by using an initial heuristic.

2 Bacterial mutation:
All bacteria are mutated in all their genes multiple times in random orders. In case of each mutation step, if the original value was better, then it is restored; if the new one makes the individual have higher fitness value, then it is kept.

3 Gene transfer:
The population is divided into two parts according to the fitness values. The individuals possessing higher fitness values form the superior and the ones having lower values form the inferior part of the population.
Then pairs are formed, where the first members of the pairs are from the superior part (superior individuals) and the second members come from the inferior part (inferior individuals). For each pair a random point of the chromosome is selected. Then the value of the gene at the selected point in the inferior bacterium is overwritten with the value of the gene at the selected point in the superior bacterium.

The main iteration loop of the algorithm contains steps 2 and 3. The algorithm stops if at the end of a generation one of the termination criteria fulfils (generation limit reached, time limit exceeded, etc.). After termination the best individual represents the quasi-optimal solution.

3.3 Particle Swarm Optimization

Another type of iterative methods is called swarm intelligence techniques. These algorithms, like the nowadays famous Particle Swarm Optimization (PSO) [9], are inspired by social behavior observed in nature, e.g. bird flocking or fish schooling. In these methods a number of individuals try to find better and better places by exploring their environment, led by their own experiences and the experiences of the whole community. Since these methods are also based on processes of the nature, like GA or BEA, and since there is also a type of evolution in them ('social evolution'), they can be categorized amongst the evolutionary algorithms.

Similarly, as was mentioned above, these techniques can also be applied as optimization methods if the individuals represent candidate solutions.

PSO comprises the following steps:

1 Initialization:
An initial population is created by selecting random elements of the search space according to some distribution, or by using an initial heuristic.
In the case of each individual, their current place is considered as its local best place (see its reason below). Moreover, the place of the best individual is stored as the global best place in the search space.

2 Moving the individuals:
The individuals are moved (their position within the search space x_i is changed) based on their local best place l_i and on the global best place g_i (the subscript i denotes the number of the current iteration). The new position x_i is determined by the following equations:

$$v_0 = 0$$
$$v_{i+1} = \varphi_v v_i + \varphi_l r_l (l_i - x_i) + \varphi_g r_g (g_i - x_i) \quad (2)$$
$$x_{i+1} = x_i + v_{i+1}$$

where φ_v, φ_l and φ_g are parameters of the algorithm, r_l and r_g are random values.

3 Updating local and global best places:
The individuals are evaluated and if an individual is at a better position than its local best place, the local best place is set to the current position. If the currently best individual in the population has higher fitness value than the fitness value of the global best place, then the global best place is set to the position of the currently best individual.

The main iteration loop of the algorithm contains steps 2 and 3. The algorithm stops if at the end of a generation one of the termination criteria fulfils (generation limit reached, time limit exceeded, etc.). After termination the global best place represents the quasi-optimal solution.

3.4 Memetic Algorithms

The techniques causing minor modifications to the candidate solutions iteration by iteration and thus exploring only the ‘neighborhood’ of particular elements of the search space are called local search methods.

After a proper amount of iterations, as a result of these minor modification steps, the local search algorithms find the ‘nearest’ local minimum quite accurately. However, these techniques are very sensible to the location of the starting point. In order to find the global optimum, the starting point must be located close enough to it, in the sense that no local optima separate the two points.

Evolutionary computation techniques explore the whole objective function, because of their characteristic, so they find the global optimum, but they approach it slowly. Local search based algorithms, meanwhile, find only the nearest local optimum; however, they converge to it faster.

Avoiding the disadvantages of the two different technique types, evolutionary algorithms (including swarm intelligence techniques) and local search methods may be combined [11], for example, if in each iteration for each individual one or more local search steps are applied. Expectedly, this way the advantages of both local search and evolutionary techniques can be exploited: the local optima can be found quite accurately on the whole objective function, i.e. the global optimum can be obtained quite accurately.

There are several results in the literature confirming this expectation in the following aspect. Usually, the more difficult the applied local search step is, the higher convergence speed the algorithm has in terms of number of generations. It must be emphasized that most often these results discuss the convergence speed in terms of number of generations. However, the more difficult an algorithm is, the greater computational demand it has, i.e. each iteration takes longer.

Therefore the question arises: how does the speed of the convergence change in terms of time if the local search based technique applied in the method is changed?

Apparently, this is a very important question of applicability, because in real world applications time as a resource is a very important and expensive factor, but the number of generations the algorithm executes does not really matter.

This is the reason why the efficiency in terms of time was chosen to be investigated in this paper.

4 Proposed Encoding Methods

Two types of individual representation (i.e. two encoding methods) are proposed in this paper for the evolutionary techniques.

The first one is based on the permutations themselves, thus the evolutionary operators modify the elements of the permutations directly.

The second encoding method is an indirect, real value based encoding approach, which is an obvious extension of those representations applied for numerical optimization problems. Although the operators modify the values of real valued vectors (arrays) – since the objective function is defined over permutations, the chromosomes represent permutations actually – there is a need to convert the real valued vectors to permutations somehow.

In order to reduce time complexity costs, the chromosomes can be 'mirrored' within the individuals in a manner which makes the modifications caused by the evolutionary operators and the evaluation of the individual more simply performable.

4.1 Permutation-based Encoding

This representation is based on the permutations themselves. Each chromosome holds one single permutation, where the genes represent the jobs and each gene holds an element of the permutation. That is, the chromosome is an integer vector, where the values of the genes are between 1 and *n* (where *n* is the number of jobs), additionally, every integer appears once in the chromosome.

Actually, this permutation is not the permutation the objective function gets, i.e. it is not the one defined in Section 2, but its inverse (as was explained by a simple example before). Thus, hereafter this will be called the 'inverse permutation'.

4.2 Real Value-based Encoding

Most often in the case of numerical optimization problems, the individuals have binary or real representations. This means that the chromosomes are binary or real valued vectors (arrays) representing points in the search space, i.e. candidate solutions.

In those most frequent cases when the objective function is defined over R^n (or over a subset of R^n), it is a natural way to encode the individuals in real valued vectors.

This representation can be extended to PFSP tasks easily as follows.

If the number of jobs is *n*, then the chromosomes are real valued vectors with length of *n*. Since in the case of PFSP tasks the objective function is defined over permutations, the real valued vectors must be converted to permutations. This can be done by ordering the genes according to the values they have. Because there is exactly one permutation in S_n corresponding to every gene order, the new gene order is equivalent to a permutation.

There seems to be an unnecessary 'overhead' in the previous encoding technique, because one could say that the chromosome should hold the permutation and the operators should modify the permutations directly, instead of changing a real valued vector and the permutation via this vector.

However, despite the computational overhead, this encoding manner is more useful, as will be presented in the next sections.

4.3 Mirroring the Chromosomes

Performing the effects of the evolutionary operators on the individuals can be made computationally cheaper in the following way.

The individuals do not comprise only one single chromosome as usual, but two chromosomes being similar to each other: an original and a mirrored one. The original chromosome contains a vector of real numbers and the inverse permutation or only the inverse permutation (based on the base of the encoding) as discussed above. The mirrored chromosome contains the inverse of the inverse permutation (i.e. the permutation used by the objective function) and in the case of real value based encoding, the adequate permutated order of the real numbers (i.e. the real numbers in an ascending order).

The chromosome and the mirrored chromosome are updated simultaneously in every step during the application of the evolutionary operators. Thus, they are always equivalent in the sense that they always represent the same candidate solution for the problem.

The reasons why this mirroring technique causes the reduction of computational costs will be explained in the next section during the discussion of the certain operators.

5 Established Evolutionary Operators

The different evolutionary operators used by the algorithms are derived back to three 'atomic operators': mutation, gene transfer and local search.

Mutation in GA and bacterial mutation in BEA can be obviously constructed by using the atomic operator mutation, and gene transfer in BEA can be done by using the atomic operator gene transfer.

Crossover in GA can be considered as a sequence of gene transfers from a given position to a given position in a random order.

The atomic operator local search is exactly the same as the local search operator in all three evolutionary techniques.

It is easy to see that if all the atomic operators are defined so that their results are valid individuals (i.e. individuals representing permutations), the constructed operators also results in valid individuals.

The atomic operators are the following.

5.1 Mutation

5.1.1 Permutation-based Encoding

In the permutation-based case, the mutation of a gene means that the value of the gene is set to a random integer value from 1 to *n* (where *n* is the number of jobs). This modification would lead out from the search space, because the resulting integer vector would not be a permutation, hence a ‘compensation step’ is made, i.e. the gene whose value is taken by the mutated gene changes its value to the previous value of the mutated gene. That is, during mutation, the mutated gene changes its value with a random gene. In this way the mutation operator is closed with respect to the search space.

The change is committed both in the chromosome and in the mirrored chromosome.

Since the permutation-based mutation modifies only two values in the permutation, it makes ‘local’ changes within the chromosome.

5.1.2 Real Value-based Encoding

When a real value based gene is mutated, it is set to a random real value. Thus, the permutation represented by the chromosome changes.

It would be computationally expensive to compute the new corresponding inverse permutation by reordering the whole chromosome. Instead of this, the mirrored chromosome can be applied, where the place of the new value can be found easily by a computationally cheap binary search (since the real values are ordered in the mirrored chromosome). Then, in the mirrored chromosome, the sub-chromosome (i.e. the gene-sequence) between the original and the new place of the mutated gene is shifted one place left (if the new value is higher than the old one) or one place right (if the new value is lower than the old one).

During the shift, the corresponding elements of the inverse permutation are also updated in the chromosome.

Actually, the real value-based mutation means a random direction shift of a random length part of the mirrored chromosome. Thus, it causes not only local effects unlike the permutation based mutation.

After the previous description of the real value-based mutation, one could ask whether an equivalent operator could not be constructed based on only the permutations; i.e. is it possible that there is no difference between the strength of the two different encoding manners?

The answer is no, an equivalent operator could not be constructed based on only the permutations, because although the shift of random length gene-sequences could easily be made, in the case of real value based representation, the distribution of random variables determining the lengths and positions are also developing (implicitly) while the real values are changing. Therefore, the diversity of the real value based encoding is higher.

It was mentioned in the previous section that this representation has a computational overhead, but as it discussed now, it may give higher diversity to the mutation. Thus, certainly there is a difference between the strength of the two different encoding manners; however, it is an open question which one is more efficient, and by how much.

This will also be investigated in Section 6.

5.2 Gene Transfer

5.2.1 Permutation-based Encoding

During gene transfer in the case of permutation based encoding the inverse permutation value of the selected gene in the target individual is set to the inverse permutation value of the corresponding gene of the source individual. Hereafter, a compensation step is made similarly as in the case of mutation.

5.2.2 Real Value-based Encoding

Applying real value-based encoding, the gene transfer is not much different. The real value of the selected gene in the target individual is set to the real value of the corresponding gene of the source individual. Hereafter, a similar shifting in the mirrored chromosome followed by the update of the chromosome is made as in the case of mutation.

5.3 Local Search

The local search is performed the same way in the case of both representations. One iteration cycle of the local search is as follows.

First of all, a random order of the elements of the permutation from the first to the last but one is selected. Then, following this order, the neighboring elements within the mirrored chromosome try to change their values with each other so that if according to the random order the current element is the i^{th}, then it tries to change its value with the $(i+1)^{th}$. After each change between the neighbors, if the resulting permutation is better (i.e. it has a higher fitness value), the change is kept and both the chromosome and the mirrored chromosome are updated according to the modification made. Otherwise, the change is rolled back.

6 Optimization Algorithms Investigated in this Paper

Based on the previous sections ten different evolutionary optimization techniques were constructed and evaluated. These are the following:

- Genetic Algorithm based techniques:
    - GAr: Genetic Algorithm without local search using real value based encoding
    - GAp: Genetic Algorithm without local search using permutation based encoding
    - GMAr: Genetic Algorithm with local search (Memetic Algorithm) using real value based encoding
    - GMAr: Genetic Algorithm with local search (Memetic Algorithm) using permutation based encoding
- Bacterial Evolutionary Algorithm based techniques:
    - BEAr: Bacterial Evolutionary Algorithm without local search using real value based encoding
    - BEAp: Bacterial Evolutionary Algorithm without local search using permutation based encoding
    - BMAr: Bacterial Evolutionary Algorithm with local search (Bacterial Memetic Algorithm) using real value based encoding
    - BMAp: Bacterial Evolutionary Algorithm with local search (Bacterial Memetic Algorithm) using permutation based encoding
- Particle Swarm Optimization based techniques:
    - PSO: Particle Swarm Optimization without local search using real value based encoding

- PMO: Particle Swarm Optimization with local search using real value based encoding

In the remaining part of the paper GAr, GAp, GMAr and GMAp will be referred to as 'genetic' techniques, BEAr, BEAp, BMAr and BMAp will be labeled as 'bacterial' methods, and finally PSO and PMO will be referred to as 'particle swarm' algorithms.

7 Evaluation of the Obtained Techniques

Simulation runs were carried out in order to evaluate and to compare the efficiency of the proposed approaches and the established algorithms. First, the new methods are compared to each other, then the best one is compared to two other heuristics: the well-known Iterated Greedy technique (IG) [12] and a genetic algorithm based memetic method (MA) [13], which is e.g. used in combination with IG in multi-processor systems.

For these purposes, a dozen problems were applied from the well-known Taillard's benchmark set. Exactly one problem from each available problem sizes.

In the simulations, the parameters of the newly proposed methods had the following values, because after a number of test runs these values seemed to be the most suitable.

The number of individuals in a generation was 14 in genetic and 8 in bacterial algorithms, whereas it was 80 in particle swarm methods. In the case of genetic techniques the selection rate was 0.5 and the mutation rate was 0.3; in the case of bacterial techniques, the number of clones was 2 and 1 gene transfer was carried out in each generation. The genetic methods applied the elitist strategy.

In the iterated greedy methods, 4 jobs were selected to remove in each generation and the temperature parameter was 5 (see [12]). The MA technique used 13 individuals as in [13].

The simulation was carried out on a PC with E8500 3.16 GHz Intel Core 2 Duo CPU and using Windows Vista Business 64-bit operating system.

In the case of all ten new algorithms for all benchmark problems 5 runs were carried out. Then the means of the obtained values were taken.

The means of the objective function values of the best individuals during the runs of the new techniques are presented in figures (Figures 1-12) to get a better overview. The horizontal axes show the elapsed computation time in seconds and the vertical axes show the makespan values of the best individuals at the current time.

In the figures, dotted lines show the results of the pure evolutionary algorithms applying permutation based encoding (GAp and BEAp); dashed-dotted lines denote the memetic techniques using permutation-based encoding (GMAp and BMAp); solid lines present the graphs of the pure evolutionary methods applying real value-based encoding (GAr, BEAr and PSO); and dashed lines show the memetic techniques using real value based encoding (GMAr, BMAr and PMO). In each case a dashed horizontal line shows the best known makespan value according to [6].

The means of the resulting values were collected in tables (Tables I-VI). In the tables under the 'Problem' label the 'ID' columns show the identifier of the tasks in Taillard's benchmark problem set [6] and 'Size' denotes the size of the benchmark problem (in the form of "number of jobs times number of machines"). The best known makespan values according to [6] are collected in columns labeled by 'Best known value'. 'Time limit' shows the length of the simulation runs in seconds. The time limits were chosen according to test runs to values, after which the techniques did not show significant improvements (cf. Figures 1-12). Under the algorithm labels the 'Results' columns present the mean of the makespan values produced by the techniques, 'Rel. diff.' shows the mean of the relative difference of these makespan values compared to the known best ones

$$\frac{1}{5}\sum_{i=1}^{5}(\text{Result}_i - \text{Best known value})/\text{Best known value}, \tag{3}$$

whereas 'No. of gen.' denotes the mean of the number of executed generations. The best makespan values for each benchmark problem are bold underlined numbers and the best values of a particular evolutionary algorithm family (genetic, bacterial and particle swarm) for each benchmark problem not being the totally best values are italic underlined numbers.

The results of the runs of the new algorithms and their short explanations follow in the next subsection. After that, the results of the comparison with IG and MA are analyzed. In Subsection 7.3 conclusions will be drawn about the main characteristics of the behavior of the methods.

7.1 Experimental Results for the Established Techniques

The following observations could be made based on the obtained values (see Figures 1-12 and Tables I-V).

Considering the figures and tables probably the most obvious tendency of the results is that bacterial techniques gave better performance in each case than genetic and particle swarm based ones, as they were never outperformed by other methods. Such an unambiguous observation cannot be made between the latter two algorithm families. The superiority of the bacterial algorithms is growing in terms of the difficulty (i.e. the size) of the optimization task. In the case of easier problems, the difference in efficiency between the algorithm families is not so significant (see Figures 1, 2, 4 and 7); however as the problem size increases, the significance grows (see Figures 3, 5, 6, 8, 9 and 10). Finally, in the case of the most difficult tasks (i.e. the biggest problem sizes) the difference is more than significant.

By looking at Table IV it is clear to see that BMAr performed best during the simulation runs, because in half of the cases it produced better results than the other algorithms, whereas in three more cases it found the known best values for the benchmark problems. This means that BMAr was outperformed by other techniques only in a quarter of the problems.

As can also be observed, memetic algorithms (the methods integrating local search steps) had higher efficiency in most of the cases. Among the genetic techniques, GMAr was the best in 8 problems, whereas GAr was the best in its family only 4 times out of 12 (see Tables I-II). In the case of the bacterial methods, the pure evolutionary techniques gave better results only in two cases, whereas the memetic ones had higher performance in seven cases. PMO was outperformed by PSO only once (see Table V).

One more very important feature is characterized by the results. Except for two cases (‘ta011’ in Table II and ‘ta071’ in Table IV) out of 48, the real value based methods were never worse than the corresponding permutation based ones. Moreover, even in those exceptional cases, the differences are insignificantly tiny.

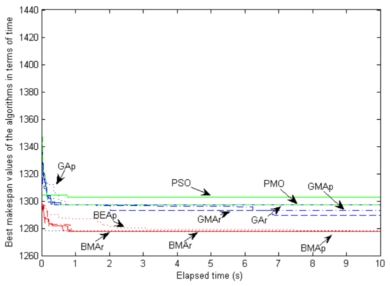

Figure 1
Results for the 20x5 size problem (ta001)

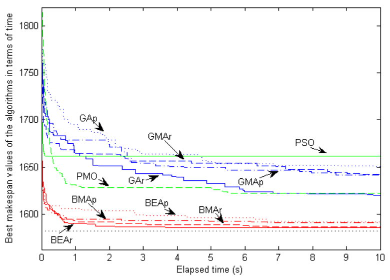

Figure 2
Results for the 20x10 size problem (ta011)

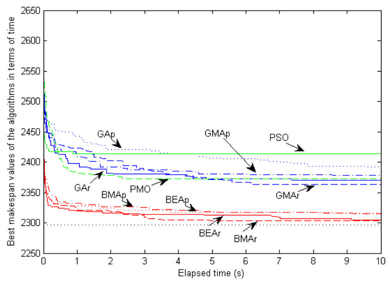

Figure 3
Results for the 20x20 size problem (ta021)

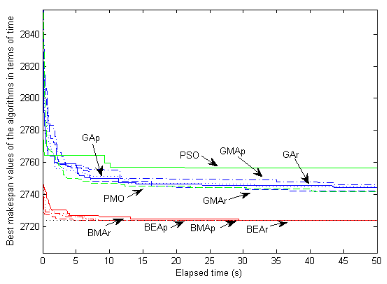

Figure 4
Results for the 50x5 size problem (ta031)

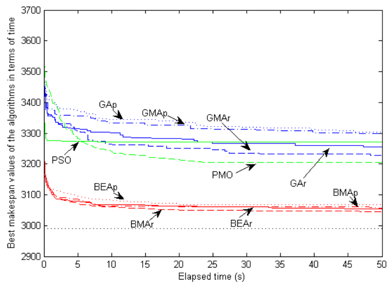

Figure 5
Results for the 50x10 size problem (ta041)

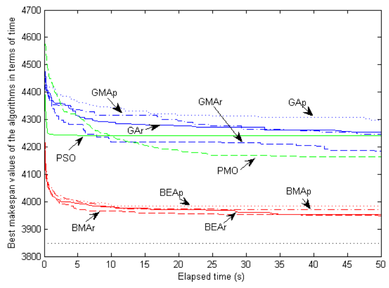

Figure 6
Results for the 50x20 size problem (ta051)

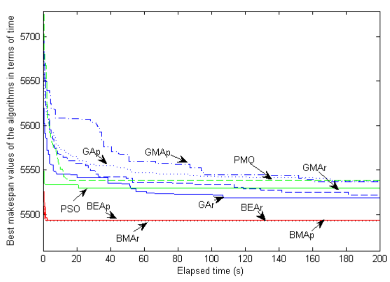

Figure 7
Results for the 100x5 size problem (ta061)

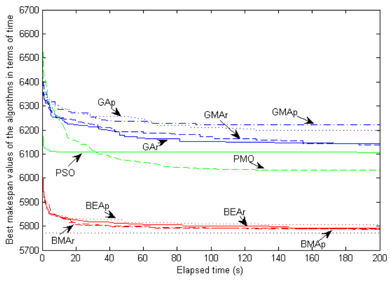

Figure 8
Results for the 100x10 size problem (ta071)

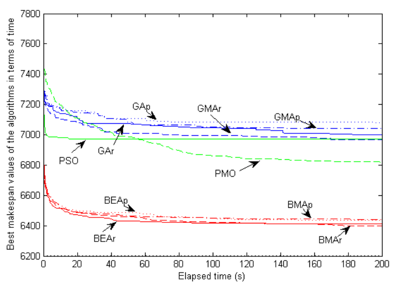

Figure 9
Results for the 100x20 size problem (ta081)

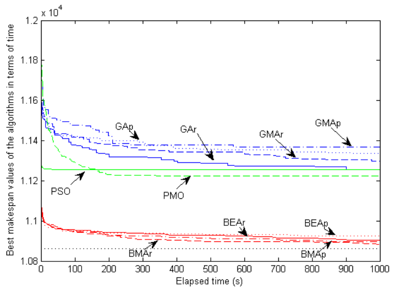

Figure 10
Results for the 200x10 size problem (ta091)

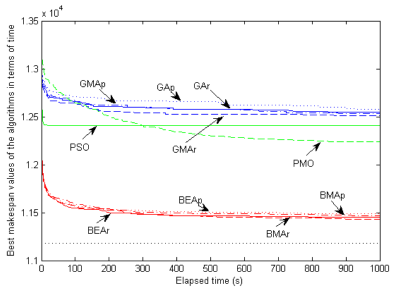

Figure 11
Results for the 200x20 size problem (ta101)

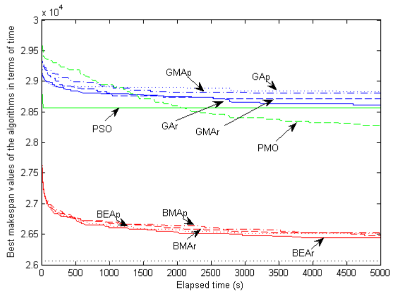

Figure 12
Results for the 500x20 size problem (ta111)

Table I
Results of the pure evolutionary genetic methods

Problem				GAp			GAr		
ID	Size	Best known value	Time limit	Result	Rel. diff.	No. of gen.	Result	Rel. diff.	Num. of gen.
ta001	20x5	1278	10	1297	1.49%	115751.8	1297	1.49%	99583.6
ta011	20x10	1582	10	1650.4	4.32%	85947.4	*1619.8*	2.39%	76262
ta021	20x20	2297	10	2392.6	4.16%	53249.6	2370	3.18%	49463.4
ta031	50x5	2724	50	2745	0.77%	229461.6	2744.4	0.75%	174902.8
ta041	50x10	2991	50	3304.6	10.48%	172645.8	3255.8	8.85%	137602.4
ta051	50x20	3847	50	4298.6	11.74%	111569.2	4254.2	10.58%	96096
ta061	100x5	5493	200	5537.8	0.82%	486640	*5519*	0.47%	301985.4
ta071	100x10	5770	200	6197	7.40%	348813.6	6142.2	6.45%	241455.6
ta081	100x20	6202	200	7077.8	14.12%	224934	7000.4	12.87%	174915.6
ta091	200x10	10862	1000	11336.2	4.37%	858038.4	*11254.4*	3.61%	482466.8
ta101	200x20	11181	1000	12577.6	12.49%	527048.8	12551.2	12.25%	346630.6
ta111	500x20	26059	5000	28827.6	10.62%	1033591.6	*28614.6*	9.81%	490406.2

Table II

Results of the genetic algorithm based memetic techniques

Problem				GMAp			GMAr		
ID	Size	Best known value	Time limit	Result	Rel. diff.	No. of gen.	Result	Rel. diff.	No. of gen.
ta001	20x5	1278	10	1293.2	1.19%	13525.8	*1289.4*	0.89%	13220.6
ta011	20x10	1582	10	1641	3.73%	9520.8	1641.6	3.77%	9422.4
ta021	20x20	2297	10	2378	3.53%	5648.6	*2363.2*	2.88%	5598.2
ta031	50x5	2724	50	2746.2	0.81%	11376.4	*2742*	0.66%	11203.8
ta041	50x10	2991	50	3297.2	10.24%	7934.6	*3228.6*	7.94%	7930.8
ta051	50x20	3847	50	4246.6	10.39%	4911	*4183.6*	8.75%	4831.6
ta061	100x5	5493	200	5537.2	0.80%	11758.4	5522	0.53%	11543.4
ta071	100x10	5770	200	6220.6	7.81%	8079.4	*6136.2*	6.35%	8048
ta081	100x20	6202	200	7040.8	13.52%	5057.4	*6967.8*	12.35%	5014.2
ta091	200x10	10862	1000	11369.6	4.67%	10104.4	11298.4	4.02%	10109.8
ta101	200x20	11181	1000	12532.4	12.09%	5787.8	*12514.4*	11.93%	5857.2
ta111	500x20	26059	5000	28793.6	10.49%	4725.8	28715	10.19%	4642

Table III

Results of the pure evolutionary bacterial methods

Problem				BEAp			BEAr		
ID	Size	Best known value	Time limit	Result	Rel. diff.	No. of gen.	Result	Rel. diff.	No. of gen.
ta001	20x5	1278	10	***1278***	0.00%	5943.4	***1278***	0.00%	5495
ta011	20x10	1582	10	1591.6	0.61%	4320.4	***1585.2***	0.20%	4048.8
ta021	20x20	2297	10	2316	0.83%	2549	2305	0.35%	2452.8
ta031	50x5	2724	50	***2724***	0.00%	5050	***2724***	0.00%	4720
ta041	50x10	2991	50	3067	2.54%	3684	3055.2	2.15%	3508.4
ta051	50x20	3847	50	3983.2	3.54%	2190	3951.6	2.72%	2140
ta061	100x5	5493	200	5493.4	0.01%	4998	***5493***	0.00%	4759
ta071	100x10	5770	200	5804.8	0.60%	3695.6	5791	0.36%	3597.6
ta081	100x20	6202	200	6436	3.77%	2205	6411	3.37%	2199
ta091	200x10	10862	1000	10923.4	0.57%	4497.8	10905.4	0.40%	4274.4
ta101	200x20	11181	1000	11491.4	2.78%	2698	11448.8	2.40%	2597
ta111	500x20	26059	5000	26516.4	1.76%	2159.8	***26440***	1.46%	2097.4

Table IV

Results of the bacterial evolutionary algorithm based memetic techniques

Problem				BMAp			BMAr		
ID	Size	Best known value	Time limit	Result	Rel. diff.	No. of gen.	Result	Rel. diff.	No. of gen.
ta001	20x5	1278	10	***1278***	0.00%	4030.4	***1278***	0.00%	3818
ta011	20x10	1582	10	1591	0.57%	2944.4	1586.4	0.28%	2770.2
ta021	20x20	2297	10	2314.4	0.76%	1728.8	***2303.8***	0.30%	1685.8
ta031	50x5	2724	50	***2724***	0.00%	3340	***2724***	0.00%	3200
ta041	50x10	2991	50	3055.6	2.16%	2455	***3045.6***	1.83%	2365.8
ta051	50x20	3847	50	3970.2	3.20%	1460	***3945.6***	2.56%	1440
ta061	100x5	5493	200	***5493***	0.00%	3332.6	***5493***	0.00%	3194
ta071	100x10	5770	200	***5787.6***	0.31%	2427.4	5788.2	0.32%	2379.6
ta081	100x20	6202	200	6438.6	3.81%	1482.6	***6392.4***	3.07%	1464
ta091	200x10	10862	1000	10897.6	0.33%	2994.6	***10882.2***	0.19%	2884.6
ta101	200x20	11181	1000	11466.2	2.55%	1774.8	***11432.4***	2.25%	1733.6
ta111	500x20	26059	5000	26516.4	1.76%	1393.8	26476.4	1.60%	1377.8

Table V
Results of the particle swarm methods

Problem				PSO			PMO		
ID	Size	Best known value	Time limit	Result	Rel. diff.	No. of gen.	Result	Rel. diff.	No. of gen.
ta001	20x5	1278	10	1302.8	1.94%	12817.4	*1297*	1.49%	1250.6
ta011	20x10	1582	10	1661.6	5.03%	10142.4	*1622.2*	2.54%	904.4
ta021	20x20	2297	10	2413.4	5.07%	7750.6	*2372.8*	3.30%	563.6
ta031	50x5	2724	50	2756.6	1.20%	22930	*2741.6*	0.65%	1070
ta041	50x10	2991	50	3271.2	9.37%	18066	*3205.4*	7.17%	766
ta051	50x20	3847	50	4240.8	10.24%	13640	*4164.6*	8.26%	470
ta061	100x5	5493	200	*5530.4*	0.68%	35867.4	5538.6	0.83%	1053.6
ta071	100x10	5770	200	6104.8	5.80%	31259	*6031*	4.52%	776.2
ta081	100x20	6202	200	6972	12.42%	24472.4	*6819*	9.95%	478
ta091	200x10	10862	1000	11255.6	3.62%	57462.2	*11222.4*	3.32%	959
ta101	200x20	11181	1000	12409.6	10.99%	47035	*12240*	9.47%	569.8
ta111	500x20	26059	5000	28578	9.67%	58591.6	*28274.2*	8.50%	436.4

Now, the question that arose in Section 5 is answered: it is worth applying real value based representation, because despite the computational overhead, the methods based on it are more efficient than the ones using permutation based encoding.

The observed behavior of the different algorithms matches with the results of our previous works comparing evolutionary algorithms on general optimization benchmark problems, and particularly on fuzzy rule based supervised machine learning problems (cf. e.g. [14], [15]).

7.2 Comparison to other Methods

Since BMAr appeared to be the most efficient algorithm, this technique is involved in further investigations: this method is compared to the Iterated Greedy heuristic and to the genetic algorithm based memetic method.

Table VI shows the results of the comparison of BMAr, IG and MA, where the heightened results are the best makespan values.

Table VI
Comparison of BMAr, IG and MA

Problem				BMAr		IG		MA	
ID	Size	Best known value	Time limit	Result	Rel. diff.	Result	Rel. diff.	Result	Rel. diff.
ta001	20x5	1278	10	***1278***	0,00%	***1278***	0,00%	***1278***	0,00%
ta011	20x10	1582	10	1586,4	0,28%	***1583,2***	0,08%	1585,4	0,21%
ta021	20x20	2297	10	2303,8	0,30%	***2301,6***	0,20%	2304,4	0,32%
ta031	50x5	2724	50	***2724***	0,00%	***2724***	0,00%	***2724***	0,00%
ta041	50x10	2991	50	3045,6	1,83%	***3035,2***	1,48%	3062,2	2,38%
ta051	50x20	3847	50	3945,6	2,56%	***3925***	2,03%	3958,4	2,90%
ta061	100x5	5493	200	***5493***	0,00%	***5493***	0,00%	***5493***	0,00%
ta071	100x10	5770	200	5788,2	0,32%	***5786,8***	0,29%	5797,8	0,48%
ta081	100x20	6202	200	6392,4	3,07%	***6350***	2,39%	6387,8	3,00%
ta091	200x10	10862	1000	***10882,2***	0,19%	10888,6	0,24%	10885,2	0,21%
ta101	200x20	11181	1000	11432,4	2,25%	***11392,4***	1,89%	11434,6	2,27%

As can be observed, BMAr was more efficient than MA, because 6 times out of 11 BMAr gave lower makespan values, whereas MA was better only 2 times. However, the most obvious fact appearing in the table is that IG significantly outperformed both other methods.

This result leads to two consequences. First, even the best established chromosome based technique cannot be a rival for one of the state-of-the-art methods, the Iterated Greedy heuristic. Second, although it cannot be a rival, it can be a "helpmate" of IG. In further research it would be worth constructing and evaluating hybrid methods based on BMAr and IG. A reason for this is that in the case of multi-processor systems, the combination of MA and IG resulted in a better technique than approaches applying only parallel IG threads [13].

However, such further investigations are beyond the scope of this paper.

7.3 Summary of the Main Observations

In short, the main observations made can be summarized as follows:

- Generally, bacterial techniques clearly outperformed the genetic and particle swarm ones.
- Usually, memetic methods (i.e. algorithms comprising local search steps as additional evolutionary operators) showed better performance than pure evolutionary approaches.
- Except in 2 cases out of 48, the methods applying real value based encoding technique were better than the ones using permutation based individual representation.
- BMAr seemed to be the overall best chromosome based evolutionary optimization heuristic for the PFSP problems.
- Although, the best constructed method was more efficient than a genetic algorithm based memetic technique applied in multi-processor systems, it was outperformed by one of the state-of-the-art heuristics, the Iterated Greedy method.

Conclusions

Our work proposed approaches for adapting chromosome based evolutionary methods to the Permutation Flow Shop Problem. The proposal included two types of individual representation (i.e. encoding method): a permutation and a real value based one. They were applied on three different chromosome based evolutionary techniques, namely the Genetic Algorithm, the Bacterial Evolutionary Algorithm and the Particle Swarm Optimization method. Both representations were applied on the two former methods, whereas the real value-based one was used for the latter optimization technique. Each mentioned algorithm was involved without and with local search steps as one of its evolutionary operators. Since the evolutionary operators of each technique were established according to the applied representation, this paper investigated a total number of ten different chromosome based evolutionary methods.

The obtained techniques were evaluated via simulation runs carried out on the well-know Taillard's benchmark problem set. Based on the experiments the following observations could be made.

The real value based representation seemed to be better than the permutation based encoding technique. The algorithms applying local search performed better than the corresponding pure evolutionary methods, whereas bacterial techniques outperformed both genetic and particle swarm algorithms overwhelmingly.

Therefore, BMAr appeared to be the best established chromosome based evolutionary optimization method for the PFSP problem.

Although, the best constructed method was more efficient than a genetic algorithm based memetic technique applied in multi-processor systems, it was outperformed by one of the state-of-the-art heuristics, the Iterated Greedy method.

Ongoing research aims to combine the BMAr technique with IG and to establish new hybrid methods more efficient than either of them. That work considers single- as well as multi-threaded algorithms.

Since among chromosome based evolutionary algorithms bacterial methods performed best, in further research, slightly modified bacterial techniques, such as the Bacterial Memetic Algorithm with Modified Operator Execution Order [16], might also be involved.

Future work may also aim to compare the investigated techniques with other state-of-the-art methods published for the PFSP task and to combine the best one among them with the chromosome based evolutionary techniques, thus establishing a promising hybrid algorithm.

Finally, an additional research direction could be the extension of the proposed approaches to other scheduling tasks, such as scheduling problems considering setup times or involving concurrent processing of batches of jobs (see e.g. [17]).

Acknowledgement

The research is supported by the National Development Agency and the European Union within the frame of the project TÁMOP-4.2.2-08/1-2008-0021 at the Széchenyi István University entitled "Simulation and Optimization – basic research in numerical mathematics".

References

[1] S. M. Johnson: Optimal Two- and Three-Stage Production Schedules with Setup Times Included, Naval Research Logistics Quarterly 1, 1954, pp. 61-68

[2] A. H. G. Rinnooy Kan: Machine Scheduling Problems: Classification, Complexity and Computations, Martinus Nijhoff, The Hague, The Netherlands, 1976

[3] E. Taillard: Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem, European Journal of Operational Research, Amsterdam, 1990, pp. 65-74

[4] A. Juan, A. Guix, R. Ruiz, P. Fonseca, F. Adelantado: Using Simulation to Provide Alternative Solutions to the Flowshop Sequencing Problem, 14th ASIM Dedicated Conf. on Simulation in Production and Logistic, Karlsruhe, Germany, 2010, pp. 349-356

[5] Z. Horváth, P. Pusztai, T. Hajba, Ch. Kiss-Tóth: Mathematical Methods and Parallel Codes for Production Line Optimization, Factory Automation 2011 Conference, Győr, Hungary, 2011

[6] E. Taillard, Benchmarks for Basic Scheduling Problems, European Journal of Operational Research 64 (2), 1993, pp. 278-285

[7] J. H. Holland: Adaption in Natural and Artificial Systems, The MIT Press, Cambridge, Massachusetts, 1992

[8] N. E. Nawa and T. Furuhashi: Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm, IEEE Transactions on Fuzzy Systems, 7(5), 1999, pp. 608-616

[9] J. Kennedy and R. Eberhart: Particle Swarm Optimization, Proceedings of the IEEE International Conference on Neural Networks (ICNN '95), 4, Perth, WA, Australia, 1995, pp. 1942-1948

[10] N. E. Nawa, T. Hashiyama, T. Furuhashi, and Y. Uchikawa: Fuzzy Logic Controllers Generated by Pseudo-Bacterial Genetic Algorithm, Proceedings of the IEEE International Conference on Neural Networks, ICNN'97, Houston, 1997, pp. 2408-2413

[11] P. Moscato: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, USA, 1989

[12] R. Ruiz, T. Stützle: A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem, European Journal of Operational Research, 177, 2007, pp. 2033-2049

[13] M. G. Ravetti, C. Riveros, A. Mendes, M. G. C. Resende, and P. M. Pardalos: Parallel Hybrid Heuristics for The Permutation Flow Shop Problem, AT&T Labs Research Technical Report, 2010, p. 14

[14] K. Balázs, J. Botzheim, L. T. Kóczy: Comparison of Various Evolutionary and Memetic Algorithms, Proceedings of the International Symposium on

Integrated Uncertainty Management and Applications, IUM 2010, Ishikawa, Japan, 2010, pp. 431-442

[15] K. Balázs, J. Botzheim, L. T. Kóczy: Comparative Analysis of Interpolative and Non-interpolative Fuzzy Rule-based Machine Learning Systems Applying Various Numerical Optimization Methods, World Congress on Computational Intelligence, WCCI 2010, Barcelona, Spain, 2010

[16] R. Lovassy, L. T. Kóczy, L. Gál: Function Approximation Performance of Fuzzy Neural Networks, Acta Polytechnica Hungarica, Vol. 7, No. 4, 2010, pp. 25-38

[17] E. Kodeekha: Case Studies for Improving FMS Scheduling by Lot Streaming in Flow-Shop Systems, Acta Polytechnica Hungarica, Vol. 5, No. 4, 2008, pp. 125-143