Elsevier Editorial System(tm) for Swarm and Evolutionary Computation
Manuscript Draft

Manuscript Number:

Title: Scalability Properties of Multi-Threaded Bacterial Iterated Greedy Heuristics Applied for the Permutation Flow Shop Problem

Article Type: Regular Paper

Keywords: Bacterial methods, Memetic algorithms, Hybrid Iterated Greedy techniques, Combinatorial optimization, Permutation Flow Shop Problem

Corresponding Author: Mr. Krisztian Balazs,

Corresponding Author's Institution: Budapest University of Technology and Economics

First Author: Krisztian Balazs

Order of Authors: Krisztian Balazs; Pal Pusztai; Zoltan Horvath; Laszlo T Koczy

Abstract: This paper analyzes the improvement in the efficiency of systems using Multi-Threaded Bacterial Iterated Greedy (MBIG) techniques, when the number of parallel processing threads are scaled.
The different variants of MBIG are recently proposed approaches for combining Iterated Greedy techniques, as state-of-the-art methods, with bacterial evolutionary algorithms based on a hybrid technique involving the Multi-Threaded Iterated Greedy heuristic and a Genetic Algorithm based memetic technique in order to efficiently solve the Permutation Flow Shop Problem on parallel computing architectures.

In our present work the MBIG variants have been executed involving various number of processing threads in order to examine the scalability of the approaches. The simulation runs have been carried out on instances from the well-known Taillard's benchmark problem set.
The scalability is then evaluated by comparing the results to each other and to the results given by the virtually parallelized implementation of the techniques discussed in our preceding paper.

Suggested Reviewers: Ruben Ruiz
rruiz@eio.upv.com

Thomas Stutzle
stuetzle@ulb.ac.be

Opposed Reviewers:

Scalability Properties of Multi-Threaded Bacterial Iterated Greedy Heuristics Applied for the Permutation Flow Shop Problem

Krisztián Balázs[a], Pál Pusztai[b], Zoltán Horváth[b], László T. Kóczy[a,c]

[a]*Department of Telecommunications and Media Informatics Budapest University of Technology and Economics*
Magyar tudósok körútja 2. Budapest, H-1117, Hungary
Email: {balazs, koczy}@tmit.bme.hu
[b]*Department of Mathematics and Computational Sciences Széchenyi István University*
Egyetem tér 1. Győr, H-9026, Hungary
Email: {pusztai, horvathz}@sze.hu
[c]*Department of Automation Széchenyi István University*
Egyetem tér 1. Győr, H-9026, Hungary
Email: koczy@sze.hu

Abstract

This paper analyzes the improvement in the efficiency of systems using Multi-Threaded Bacterial Iterated Greedy (MBIG) techniques, when the number of parallel processing threads are scaled. The different variants of MBIG are recently proposed approaches for combining Iterated Greedy techniques, as state-of-the-art methods, with bacterial evolutionary algorithms based on a hybrid technique involving the Multi-Threaded Iterated Greedy heuristic and a Genetic Algorithm based memetic technique in order to efficiently solve the Permutation Flow Shop Problem on parallel computing architectures.

In our present work the MBIG variants have been executed involving various number of processing threads in order to examine the scalability of the approaches. The simulation runs have been carried out on instances from the well-known Taillard's benchmark problem set. The scalability is then evaluated by comparing the results to each other and to the results given by the virtually parallelized implementation of the techniques discussed in our preceding paper.

Keywords: Bacterial methods, Memetic algorithms, Hybrid Iterated Greedy techniques, Combinatorial optimization, Permutation Flow Shop Problem

1. Introduction

One of the most intensively studied combinatorial optimization problems is the Permutation Flow Shop Problem (PFSP) (Johnson, 1954). 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 $n \times m$-size 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 a machine only if the machine is free (the preceding job has finished on the machine) and the job has already 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.

This problem is known to be NP-hard (Kan, 1976), thus there are no efficient algorithms to solve this task (and there is not much hope to find one). It 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. In the past few decades a number of such heuristics are invented and published (e.g. (Taillard, 1990), (Juan et al., 2010), (Horváth et al., 2011)).

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. (Taillard, 1990), (Juan et al., 2010), (Horváth et al., 2011)). These comparative studies are mostly based on the well-known Taillard's benchmark problem set (Taillard, 1993).

In one of our previous work (Balázs et al., 2012a) a uniform approach was proposed for applying various types of chromosome based evolutionary algorithms for the PFSP problem. The proposal included two ways for individual representation and the corresponding evolutionary operators built up from three so called atomic operators.

Another previous work (Balázs et al., 2012b) proposed approaches for combining the Iterated Greedy techniques as state-of-the-art methods with bacterial evolutionary algorithms to efficiently solve the Permutation Flow Shop Problem. The best resulting Bacterial Iterated Greedy method clearly outperformed the Iterated Greedy heuristic.

Studies about parallel Iterated Greedy techniques involving Memetic Algorithm (e.g. (Ravetti et al., 2010)) showed that in case of multi-threaded Iterated Greedy methods the combination with evolutionary algorithms was able to improve the performance of simple multi-threaded Iterated Greedy algorithms.

These results motivated the idea that it might be also worth to try to replace the genetic algorithm based memetic method in (Ravetti et al., 2010) with the Bacterial Memetic Algorithm, which appears to be more effective for the PFSP task (cf. (Balázs et al., 2012a)) and which shows better properties in other fields of optimization, too (see e.g. (Balázs et al., 2010b), (Balázs et al., 2010a)).

Therefore, our preceding paper (Balázs et al., 2012c) proposed approaches for combining multi-threaded Iterated Greedy techniques as state-of-the-art methods with bacterial evolutionary algorithms to efficiently solve the Permutation Flow Shop Problem on parallel computing architectures.

In our present work the MBIG variants have been executed involving various number of processing threads in order to examine the scalability of the approaches. The simulation runs have been carried out on a series of data from the well-known Taillard's benchmark problem set. The scalability is then evaluated by comparing the results to each other and to the results given by the virtually parallelized implementation of the techniques discussed in our preceding paper.

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 briefly describes the bacterial evolutionary and memetic techniques being combined with the iterated greedy algorithms, furthermore it gives a brief overview of the single and multi-threaded Iterated Greedy methods, since these techniques appear in the hybrid approaches. The basic concept and the main steps of the algorithms are also presented. The new combination approaches for multi-threaded heuristics are also described in section three. The experimental results and the observed characteristics are presented in section five. Finally, in the last section our work is summarized and some conclusions are drawn.

2. The Permutation Flow Shop Problem

As it was described in the Introduction, in this problem there is given the number of jobs n, the number of machines m and an $n \times m$-size processing time matrix $\mathbf{P}$ 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 ith job stays on the jth machine.

All the jobs should be worked by all the machines one after another. The machines are deployed in a line and each machine can handle one single job at once. That is, the processing 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 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.

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.

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.

Formally, the objective or makespan function f can be defined as follows (see e.g. (Johnson, 1954)).

$$f : S_n \mapsto \mathbb{R}^+$$

$$f(\sigma) = t(n, m, \sigma)$$

$$t(0, j, \sigma) \equiv 0$$

$$t(i, 0, \sigma) \equiv 0$$

$$t(i, j, \sigma) = \max(t(i, j-1, \sigma), t(i-1, j, \sigma)) + p_{\sigma(i),j} \tag{1}$$

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

3. Overview of the techniques applied

The purpose of this section is to enumerate and shortly describe the techniques applied in the establishment of the hybrid algorithms. Thus, in the first part of this section after a brief introduction to chromosome based evolutionary techniques, the skeleton of the Genetic and Bacterial Evolutionary Algorithms will be presented followed by the idea of memetic algorithms. Then the uniform approach for applying chromosome based techniques, like the Bacterial Memetic Algorithm, to the PFSP problem is described, which is proposed and deeply discussed in (Balázs et al., 2012a). This includes the description of both the encoding methods and the evolutionary operators applied in this research. In the second part of the section the well known Iterated Greedy method together with its bacterial hybridization and multi-threaded variant will be presented shortly.

Due to space limitations the approaches will only be outlined very briefly. For further details the reader should refer to the cited literature.

3.1. Chromosome based evolutionary algorithms

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) (Holland, 1992) or the Bacterial Evolutionary Algorithm (BEA) (Nawa and Furuhashi, 1999), imitate the abstract model of the evolution of populations observed in the nature. Their aim is to change the individuals in the population (set of individuals) by the evolutionary operators to obtain better and better ones.

3.1.1. Genetic Algorithm

One of the most (if not the most) widely applied chromosome based evolutionary techniques is the Genetic Algorithm (GA) (Holland, 1992). Due to its notoriety its further description is omitted here.

3.1.2. Bacterial Evolutionary Algorithm

Compared to GA, a somewhat different evolutionary technique is called Bacterial Evolutionary Algorithm (BEA) (Nawa and Furuhashi, 1999). BEA has proved a rather efficient method among chromosome based techniques for various optimization tasks, including the PFSP problem (cf. e.g. (Balázs et al., 2010b), (Balázs et al., 2010a), (Balázs et al., 2012a)).

BEA comprises the following steps:

1. Initialization
2. Bacterial mutation
3. Gene transfer

3.1.3. 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, while local search based algorithms find only the nearest local optimum, however, they converge to it faster.

Avoiding the disadvantages of the two different technique types, evolutionary algorithms and local search methods may be combined (Moscato, 1989), 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.

3.1.4. Encoding methods

In case of the PSFP problem two types of individual representation (i.e. two encoding methods) are proposed in (Balázs et al., 2012a) for the chromosome based evolutionary techniques, among them for the bacterial algorithms.

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. This can be done by ordering the genes according to the values they have.

There seems to be an unnecessary ‘overhead’ in this 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 turned out to be useful in our recent research (Balázs et al., 2012a). Hence the bacterial techniques applied in our present work are based on the real value based encoding method, and hereafter only this representation type will be considered in this paper.

3.1.5. Evolutionary operators

The different evolutionary operators used by the algorithms investigated in (Balázs et al., 2012a) are derived from three ‘atomic operators’: mutation, gene transfer and local search.

In bacterial methods bacterial mutation can be obviously constructed by using the atomic operator mutation as well as gene transfer can be made by using the atomic operator gene transfer.

The atomic operator local search is exactly the same as the local search operator in case of memetic techniques.

The short description of atomic operators is as follows:

- Mutation:

  When a gene is mutated, it is set to a random real value. Thus, the permutation represented by the chromosome changes, because the order of the real values in the chromosome changes.

- Gene transfer:

  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.

- Local search:

  One iteration cycle of the local search is the following. 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 according to the permutation represented by the chromosome are tried to change their values with each other so that if according to the random order the current element is the ith, then it is tried 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. Otherwise, the change is rolled back.

3.2. Iterated Greedy methods

The Iterated Greedy (IG) technique (Ruiz and Stützle, 2007) is a very simple and intuitive but rather efficient heuristic for the PFSP problem. The basic method will be described below followed by the bacterial hybrid version and the multi-threaded variant.

3.2.1. The Iterated Greedy technique

Basically, the Iterated Greedy (IG) technique (Ruiz and Stützle, 2007) comprises four steps:

1. Initialization:
   An initial permutation is created by using the deterministic NEH heuristic (Nawaz et al., 1983) (which is not described in this paper). This permutation is stored as best.
2. Destruction phase:
   A predefined number of jobs are selected randomly, and they are removed from the permutation.
3. Construction phase:
   The removed jobs are reinserted into the destructed permutation to those places in case of which the partial permutations have the lowest makespan values.
4. Acceptance check:
   If the newly created permutation is better than the original one, it is kept. If it is better than the ever best, it is stored as the new best solution. If the newly created permutation is worse than the original one, it may also be kept with a probability depending on a so-called 'temperature parameter' and on the difference between the makespan value of the original and the new solution. Otherwise the new permutation is ignored.

The main iteration loop of the algorithm contains steps 2 – 4. The algorithm stops, if at the end of an iteration one of the termination criteria fulfills (iteration limit reached, time limit exceeded, etc.). After termination the stored best permutation represents the quasi-optimal solution.

Local search steps can also be applied within the Iterated Greedy method. Usually, in this case the so called Iterative Insertion Improvement algorithm (III) (Ruiz and Stützle, 2007) is used between steps 3 and 4. During one iteration of this local search a destruction and a construction phase take place for one single job at once, but this is performed for each job in the permutation. The local search iterates until the last iteration shows no improvement.

3.2.2. The Bacterial Iterated Greedy algorithm

In our recent work (Balázs et al., 2012b) a number of hybrid bacterial iterated greedy variants were established, however in this paper only the best one will be described and involved in the multi-threaded approaches.

In the (best established) Bacterial Iterated Greedy (BIG) algorithm the above discussed Bacterial Memetic Algorithm is embedded into the Iterated Greedy heuristic. Since the Iterated Greedy method considers only one candidate solution at once, whereas the bacterial algorithms maintain a whole population of the permutations, at the embedding point a number of individuals must be derived from one candidate solution. This means that the embedding point must be such a point in the base algorithm, where the base heuristic can easily run on side-roads, i.e. it can fork to slightly different ways.

This point is the beginning of the destruction phase, where the jobs to remove are selected randomly, because different random numbers cause different processions of the destruction resulting different candidate solutions forming the population. Thus, the population of the bacterial algorithm is created by the multiple execution of the destruction. Apparently, after each destruction the corresponding construction phase follows in order to have valid permutations before the embedded bacterial heuristic starts. Here comes the main loop of the embedded technique, which iterates a predefined number of times on the created population. After that, the iterated greedy method continues with the acceptance check immediately involving the best individual from the bacterial population.

Thus, the hybrid algorithm comprises the following steps:

1. Initialization
2. Multiple destruction phase
3. Construction phase
4. Embedded bacterial algorithm
    (a) Bacterial mutation

   (b) Gene transfer
   (c) Local search
5. The best individual is selected from the bacterial population for further usage and the rest of the population is disregarded.
6. Acceptance check

The main iteration loop of the algorithm contains steps 2 – 7. The algorithm stops, if at the end of an iteration one of the termination criteria fulfills (iteration limit reached, time limit exceeded, etc.). After termination the stored best permutation represents the quasi-optimal solution.

3.2.3. Multi-threaded Iterated Greedy techniques

Considering multi-processor computer systems parallel Iterated Greedy algorithm types running on multiple threads were also invented.

A simple parallel extension of the IG method is the Multi-threaded Iterated Greedy (MIG) technique (Ravetti et al., 2010). In this case an instance of the IG algorithm is running on each thread until the termination condition is fulfilled. Then, the best one among the permutations given by the threads will be the quasi-optimal solution. The steps of MIG are the following:

1. Initialization of the parallel threads:
   An initial permutation is created for each thread by using the deterministic NEH heuristic (Nawaz et al., 1983).
2. Parallel execution of iterated greedy methods:
   Every thread executes an iterated greedy technique until a termination condition is fulfilled.
3. Selection of the best candidate solution:
   The resulting permutations given by the parallel executed methods are collected and the best one is selected as the result of the optimization process.

A more sophisticated parallel extension of the IG method is a combination of a genetic algorithm based memetic technique and the MIG algorithm. This is referred to as MA+MIG (Memetic Algorithm + Multi-threaded Iterated Greedy) method by the inventors (Ravetti et al., 2010). In this heuristic one thread is executing a genetic algorithm based memetic technique, while the others are running the IG method. During the optimization process the memetic algorithm and the IG threads are communicating with each other via a migration pool, where a larger number of individuals take place. This way, candidate solutions are continuously migrating between the threads. The steps of MA+MIG are the following:

1. Initialization of the parallel threads:
   The migration pool is filled with individuals partly by using initial heuristics and partly by generating random permutations. The threads take their initial candidate solutions from this pool.
2. Parallel execution:
   A genetic algorithm based memetic technique is running on one thread, while each other thread is executing an iterated greedy technique concurrently until a termination condition is fulfilled. During this process in case of the fulfillment of certain conditions, the threads exchange (migrate) their candidate solutions by using the migration pool.
3. Selection of the best candidate solution:
   The resulting permutations given by the parallel executed methods are collected and the best one is selected as the result of the optimization process.

3.3. Hybrid multi-threaded bacterial approaches

Unlike in the single threaded case, if multiple threads are considered, i.e. there are more than one algorithms running parallel, even if the original techniques are executed on every thread, a hybrid method can be obtained by running different methods on different threads. An example to such a hybrid multi-threaded heuristic is the above described MA+MIG method, where the original algorithms are running on all the threads, mostly IG techniques parallel, but there is one exceptional thread, where the genetic algorithm based memetic method is executed.

Our preceding paper proposed similar (but hopefully better) hybrid techniques by exchanging the heuristics on the threads (Balázs et al., 2012c). Considering our recent work on chromosome based evolutionary methods (Balázs et al., 2012a) and the previously discussed single-threaded techniques three hybrid bacterial approaches are proposed for multi-threaded PFSP optimization. They are originated from the MA+MIG method by making the following changes, respectively:

1. The genetic algorithm based memetic heuristic is exchanged with the bacterial memetic technique and the iterated greedy threads are left untouched (BMA+MIG).

2. The genetic algorithm based memetic thread is left untouched and the iterated greedy method is exchanged with a hybrid single-threaded bacterial iterated greedy technique on every thread (MA+MBIG).
3. Both heuristics are exchanged, i.e. the bacterial memetic technique is executed on one thread, while a hybrid single-threaded bacterial iterated greedy method is applied on the other threads (BMA+MBIG).

During the optimization the candidate solutions migrate between the threads via the migration pool and at the beginning of the optimization process the threads are also initialized with the individuals from the pool. The size of the migration pool equals to the sum of the size of the bacterial population and the number of iterated greedy threads. Thus, the corresponding individuals can be determined for every thread, which assignment is used during the migration.

The migration between the threads and the pool occurs according to a clock. When a certain amount of time elapsed after the last migration each thread overwrites the individual(s) in the pool assigned to the algorithm instance. Then, they take new permutations from the pool randomly. Although, the threads take new individuals, they keep their best ever permutations. The time gap between migrations is a parameter of the multi-threaded algorithm.

On the threads arbitrary single-threaded bacterial iterated greedy algorithm can be applied.

The run of the Multi-Threaded Bacterial Iterated Greedy (MBIG) methods are illustrated in Figure 1.

4. Evaluation of the obtained techniques

Simulation runs were carried out in order to evaluate and to compare the scalability of the recently proposed approaches and the established algorithms on parallel computer architectures.

For this purpose, three problems (with identifiers 'ta071', 'ta081' and 'ta101') were applied from the well-known Taillard's benchmark set (Taillard, 1993). Exactly one problem from three different problem sizes, namely 100×10, 100×20 and 200×20 ("number of jobs × number of machines"). They will be referred to as 'easy', 'medium' and 'hard' tasks, respectively.

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

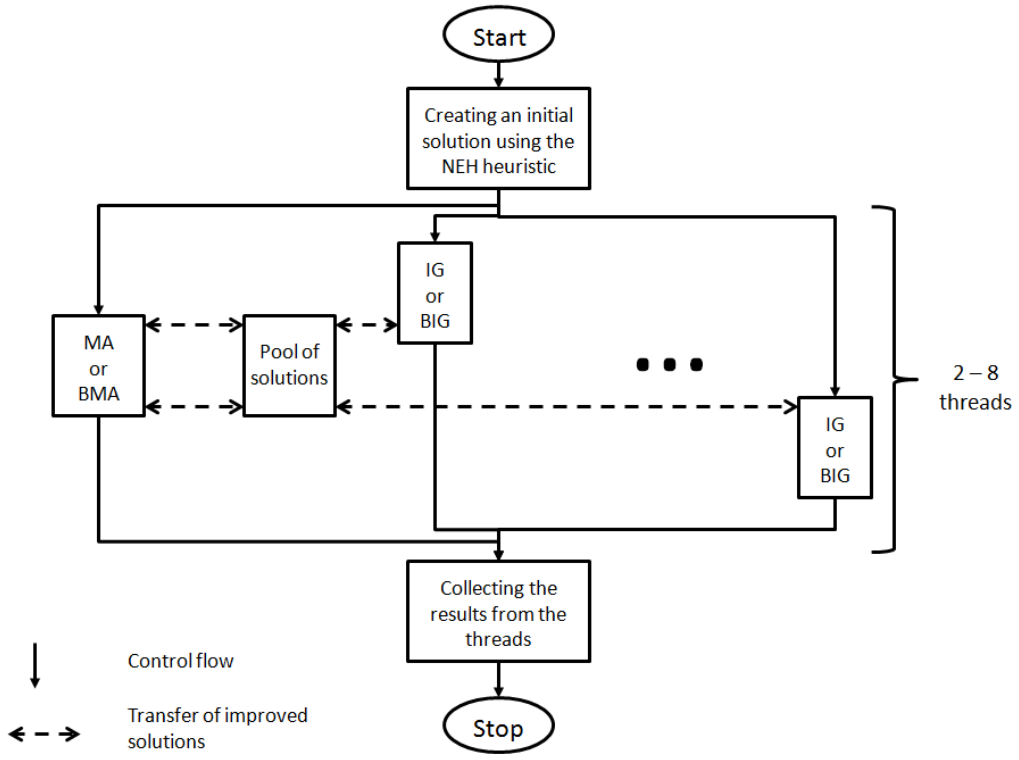

Start
Creating an initial solution using the NEH heuristic
MA or BMA
Pool of solutions
IG or BIG
IG or BIG
2 – 8 threads
Collecting the results from the threads
Stop
Control flow
Transfer of improved solutions

Figure 1: Illustration of the MBIG methods

In the bacterial algorithms the number of individuals in a generation was 8, the number of clones was 2 and 1 gene transfer was carried out in each generation. In the iterated greedy methods 4 jobs were selected to remove in each generation and the temperature parameter was 5 (see (Ruiz and Stützle, 2007)). The run of the embedded techniques took 3 iterations. The strength parameter for the parameterized distributions was 0.99.

In case of all the algorithms for all benchmark problems 10 runs were carried out. Then the mean of the obtained values were taken.

The means of the resulting values were collected in tables. In Table 1 and Table 2 next to the 'Problem' label the 'ID' rows show the identifiers of the tasks in Taillard's benchmark problem set (Taillard, 1993) 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 the website of Taillard[1] (which was last updated in 2005) are collected in rows labeled by 'B.k.m.v.'. 'Time limit' shows the length of the simulation runs in seconds. The time limits were chosen according to an accepted formula (Ruiz and Stützle, 2007): number of jobs $\times$ number of machines $\times$ 30 milliseconds. Next to the algorithm labels the 'Mean' and 'Std. dev.' rows present the mean and the standard deviation of the makespan values produced by the techniques, respectively. 'Rel. diff.' shows the mean of the relative differences of these makespan values compared to the known best ones:

$$\frac{1}{10}\sum_{i=1}^{10}(\text{Result}_i - \text{Best known value})/(\text{Best known value}). \qquad (2)$$

4.1. Experimental results for the hybrid methods

From the results in Table 1, it can be observed that in case of each problem the real parallel versions produced lower makespan values than the one applying virtual threads. In case of the easy problem among the real parallel variants the scalability can be clearly seen, i.e. as the number of threads involved are increased, the makespan as well as the relative difference values decrease. However, for the other two (more difficult) tasks the scalability is not so obvious, because in the results of the medium problem the 2-thread version, whereas in case of the hard task the 8-thread variant breaks the tendency of descending makespan values. The reason behind this phenomenon

[1]http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

can be the fact that during the predefined execution time the algorithms converged to each other so much (notice that the maximal difference between their relative difference values is 0.063% for the medium and hard problems altogether), that the relatively big deviation of the results (cf. 'Std. dev.' values) can strongly affect these differences.

It can be observed form Table 2 that in this real parallel environment in case of the easy problem the original MA+MIG method gave the best makespan values, however, for the medium and hard tasks the recently proposed BMA+MIG technique showed better performance (like in the virtual parallel environment, see (Balázs et al., 2012c)).

Table 1: Results for the BMAMBIG technique with various number of threads

Problem	ID	ta071	ta081	ta101
	Size	100x10	100x20	200x20
	B.k.m.v.	5770	6286	11294
	Time limit	30	60	120
8 virtual threads	Mean	5810	6419	11490
	Std. dev.	10.200	11.377	31.270
	Rel. diff.	0.693%	2.116%	1.735%
2 threads	Mean	5809	6399	11485
	Std. dev.	3.553	8.015	25.101
	Rel. diff.	0.676%	1.798%	1.691%
4 threads	Mean	5808	6403	11481
	Std. dev.	3.887	7.875	20.608
	Rel. diff.	0.659%	1.861%	1.656%
8 threads	Mean	5801	6402	11486
	Std. dev.	4.327	5.763	27.865
	Rel. diff.	0.537%	1.845%	1.700%

5. Conclusions

In this paper the recently proposed Multi-Threaded Bacterial Iterated Greedy (MBIG) techniques capable of solving the Permutation Flow Shop Problem were analyzed from the point of view of scalability, i.e. the improvement of their efficiency when more and more processing threads are applied in the executing parallel computing architecture.

Table 2: Results produced by the different hybrid multi-threaded methods on 8 threads

Problem	ID	ta071	ta081	ta101
	Size	100x10	100x20	200x20
	B.k.m.v.	5770	6286	11294
	Time limit	30	60	120
BMA+MBIG	Mean	5802	6402	11486
	Std. dev.	6.651	5.763	27.656
	Rel. diff.	0.555%	1.845%	1.700%
BMA+MIG	Mean	5800	6397	11485
	Std. dev.	7.945	13.214	25.171
	Rel. diff.	0.520%	1.766%	1.691%
MA+MBIG	Mean	5798	6418	11488
	Std. dev.	5.982	19.102	15.749
	Rel. diff.	0.485%	2.100%	1.718%
MA+MIG	Mean	5795	6413	11486
	Std. dev.	5.578	16.901	15.157
	Rel. diff.	0.433%	2.020%	1.700%

The scalability was evaluated via simulation runs carried out on the well-known Taillard's benchmark problem set. The scalability was then evaluated by comparing the results to each other and to the results given by the virtually parallelized implementation of the techniques discussed in our preceding paper.

During the experimental analysis the scalability of the approaches could be clearly observed in case of the 100×10 task, i.e. when the number of processing threads were increased, the efficiency of the techniques improved. However, for the more difficult problems the scalability is not so obvious, because during the predefined execution time the algorithms converged to each other so much, that the relatively big deviation of the results could strongly affect these differences.

The three novel techniques proposed in our preceding paper (Balázs et al., 2012c) was also compared to each other and to the original MA+MIG method. Whereas the original algorithm gave better results for the 100×10 task, in case of the more difficult problems the recently proposed BMA+MIG technique seemed to be the best.

Future work may aim at establishing more hybrid methods involving other state-of-the-art optimization algorithms for the PFSP problem, including parallel multi-threaded methods as well, and to compare them with the ones discussed in our recent and present works.

Acknowledgment

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”, the National Scientific Research Fund Grant OTKA K105529, a Széchenyi István University Main Research Direction Grant and the Social Renewal Operation Programmes TÁMOP-4.2.2 08/1-2008-0021 and 421 B.

References

Balázs, K., Botzheim, J., Kóczy, L. T., 2010a. Comparative analysis of interpolative and non-interpolative fuzzy rule based machine learning systems applying various numerical optimization methods. In: World Congress on Computational Intelligence, WCCI 2010. Barcelona, Spain, pp. 875–982.

Balázs, K., Botzheim, J., Kóczy, L. T., 2010b. Comparison of various evolutionary and memetic algorithms. In: Proceedings of the International Symposium on Integrated Uncertainty Management and Applications, IUM 2010. Ishikawa, Japan, pp. 431–442.

Balázs, K., Horváth, Z., Kóczy, L. T., 2012a. Different chromosome based evolutionary approaches for the permutation flow shop problem. Acta Polytechnica Hungarica 2 (2), 115–138.

Balázs, K., Horváth, Z., Kóczy, L. T., 2012b. Hybrid bacterial iterated greedy heuristics for the permutation flow shop problem. In: World Congress on Computational Intelligence, WCCI 2012. Brisbane, Australia, pp. 1–8.

Balázs, K., Horváth, Z., Kóczy, L. T., 2012c. Multi-threaded bacterial iterated greedy heuristics for the permutation flow shop problem. In: 13th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2012. Budapest, Hungary, pp. 63–66.

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

Horváth, Z., Pusztai, P., Hajba, T., Kiss-Tóth, C., 2011. Mathematical methods and parallel codes for production line optimization. In: Factory Automation 2011 Conference. Győr, Hungary.

Johnson, S. M., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68.

Juan, A., Guix, A., Ruiz, R., Fonseca, P., Adelantado, F., 2010. Using simulation to provide alternative solutions to the flowshop sequencing problem. In: 14th ASIM Dedicated Conf. on Simulation in Production and Logistic. Karlsruhe, Germany, pp. 349–356.

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

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA.

Nawa, N. E., Furuhashi, T., Oct. 1999. Fuzzy system parameters discovery by bacterial evolutionary algorithm. IEEE Transactions on Fuzzy Systems 7 (5), 608–616.

Nawaz, M., Jr., E. E. E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 11 (1), 91–95.

Ravetti, M. G., Riveros, C., Mendes, A., Resende, M. G. C., Pardalos, P. M., 2010. Parallel hybrid heuristics for the permutation flow shop problem. Research technical report, AT&T Labs.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049.

Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research, 65–74.

Taillard, E., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64 (2), 278–285.

```
This is pdfTeX, Version 3.1415926-1.40.10 (Web2C 2009) (format=latex 2011.5.3)  28 AUG 2013 13:55
entering extended mode
 %&-line parsing enabled.
**balazs_et_al.tex
(./balazs_et_al.tex
LaTeX2e <2009/09/24>
Babel <v3.8l> and hyphenation patterns for english, usenglishmax, dumylang, noh
yphenation, german-x-2009-06-19, ngerman-x-2009-06-19, ancientgreek, ibycus, ar
abic, basque, bulgarian, catalan, pinyin, coptic, croatian, czech, danish, dutc
h, esperanto, estonian, farsi, finnish, french, galician, german, ngerman, mono
greek, greek, hungarian, icelandic, indonesian, interlingua, irish, italian, ku
rmanji, latin, latvian, lithuanian, mongolian, mongolian2a, bokmal, nynorsk, po
lish, portuguese, romanian, russian, sanskrit, serbian, slovak, slovenian, span
ish, swedish, turkish, ukenglish, ukrainian, uppersorbian, welsh, loaded.
(./elsarticle.cls
Document Class: elsarticle 2009/09/17, 1.20b: Elsevier Ltd
\@bls=\dimen102
(c:/texlive/2009/texmf-dist/tex/latex/base/article.cls
Document Class: article 2007/10/19 v1.4h Standard LaTeX document class
(c:/texlive/2009/texmf-dist/tex/latex/base/size12.clo
File: size12.clo 2007/10/19 v1.4h Standard LaTeX file (size option)
)
\c@part=\count79
\c@section=\count80
\c@subsection=\count81
\c@subsubsection=\count82
\c@paragraph=\count83
\c@subparagraph=\count84
\c@figure=\count85
\c@table=\count86
\abovecaptionskip=\skip41
\belowcaptionskip=\skip42
\bibindent=\dimen103
) (c:/texlive/2009/texmf-dist/tex/latex/graphics/graphicx.sty
Package: graphicx 1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
(c:/texlive/2009/texmf-dist/tex/latex/graphics/keyval.sty
Package: keyval 1999/03/16 v1.13 key=value parser (DPC)
\KV@toks@=\toks14
) (c:/texlive/2009/texmf-dist/tex/latex/graphics/graphics.sty
Package: graphics 2009/02/05 v1.0o Standard LaTeX Graphics (DPC,SPQR)
(c:/texlive/2009/texmf-dist/tex/latex/graphics/trig.sty
Package: trig 1999/03/16 v1.09 sin cos tan (DPC)
) (c:/texlive/2009/texmf-dist/tex/latex/latexconfig/graphics.cfg
File: graphics.cfg 2009/08/28 v1.8 graphics configuration of TeX Live
)
Package graphics Info: Driver file: dvips.def on input line 91.
(c:/texlive/2009/texmf-dist/tex/latex/graphics/dvips.def
File: dvips.def 1999/02/16 v3.0i Driver-dependant file (DPC,SPQR)
))
\Gin@req@height=\dimen104
\Gin@req@width=\dimen105
) (c:/texlive/2009/texmf-dist/tex/latex/psnfss/pifont.sty
Package: pifont 2005/04/12 PSNFSS-v9.2a Pi font support (SPQR)
LaTeX Font Info:    Try loading font information for U+pzd on input line 63.
(c:/texlive/2009/texmf-dist/tex/latex/psnfss/upzd.fd
File: upzd.fd 2001/06/04 font definitions for U/pzd.
)
LaTeX Font Info:    Try loading font information for U+psy on input line 64.
(c:/texlive/2009/texmf-dist/tex/latex/psnfss/upsy.fd
File: upsy.fd 2001/06/04 font definitions for U/psy.
))
\c@tnote=\count87
\c@fnote=\count88
\c@cnote=\count89
\c@ead=\count90
\c@author=\count91
\@eadauthor=\toks15
\c@affn=\count92
\absbox=\box26
\keybox=\box27
\Columnwidth=\dimen106
\space@left=\dimen107
\els@boxa=\box28
\els@boxb=\box29
\leftMargin=\dimen108
\@enLab=\toks16
\@sep=\skip43
\@@sep=\skip44
(./balazs_et_al.spl) (c:/texlive/2009/texmf-dist/tex/latex/natbib/natbib.sty
Package: natbib 2009/07/16 8.31 (PWD, AO)
\bibhang=\skip45
\bibsep=\skip46
LaTeX Info: Redefining \cite on input line 694.
\c@NAT@ctr=\count93
)
\splwrite=\write3
\openout3 = `balazs_et_al.spl'.

\appnamewidth=\dimen109
) (c:/texlive/2009/texmf-dist/tex/latex/amsfonts/amssymb.sty
Package: amssymb 2009/06/22 v3.00
(c:/texlive/2009/texmf-dist/tex/latex/amsfonts/amsfonts.sty
Package: amsfonts 2009/06/22 v3.00 Basic AMSFonts support
\@emptytoks=\toks17
\symAMSa=\mathgroup4
\symAMSb=\mathgroup5
LaTeX Font Info:    Overwriting math alphabet `\mathfrak' in version `bold'
(Font)                  U/euf/m/n --> U/euf/b/n on input line 96.
)) (c:/texlive/2009/texmf-dist/tex/latex/multirow/multirow.sty
\bigstrutjot=\dimen110
) (c:/texlive/2009/texmf-dist/tex/latex/placeins/placeins.sty
Package: placeins 2005/04/18  v 2.2
) (c:/texlive/2009/texmf-dist/tex/latex/tools/multicol.sty
Package: multicol 2008/12/05 v1.6h multicolumn formatting (FMi)
```

```
\c@tracingmulticols=\count94
\mult@box=\box30
\multicol@leftmargin=\dimen111
\c@unbalance=\count95
\c@collectmore=\count96
\doublecol@number=\count97
\multicoltolerance=\count98
\multicolpretolerance=\count99
\full@width=\dimen112
\page@free=\dimen113
\premulticols=\dimen114
\postmulticols=\dimen115
\multicolsep=\skip47
\multicolbaselineskip=\skip48
\partial@page=\box31
\last@line=\box32
\mult@rightbox=\box33
\mult@grightbox=\box34
\mult@gfirstbox=\box35
\mult@firstbox=\box36
\@tempa=\box37
\@tempa=\box38
\@tempa=\box39
\@tempa=\box40
\@tempa=\box41
\@tempa=\box42
\@tempa=\box43
\@tempa=\box44
\@tempa=\box45
\@tempa=\box46
\@tempa=\box47
\@tempa=\box48
\@tempa=\box49
\@tempa=\box50
\@tempa=\box51
\@tempa=\box52
\@tempa=\box53
\c@columnbadness=\count100
\c@finalcolumnbadness=\count101
\last@try=\dimen116
\multicolovershoot=\dimen117
\multicolundershoot=\dimen118
\mult@nat@firstbox=\box54
\colbreak@box=\box55
) (c:/texlive/2009/texmf-dist/tex/latex/blindtext/blindtext.sty
Package: blindtext 2009/06/14 V1.9b blindtext-Package
(c:/texlive/2009/texmf-dist/tex/latex/tools/xspace.sty
Package: xspace 2006/05/08 v1.12 Space after command names (DPC,MH)
)
\c@blindtext=\count102
\c@Blindtext=\count103
\blind@countxx=\count104
```

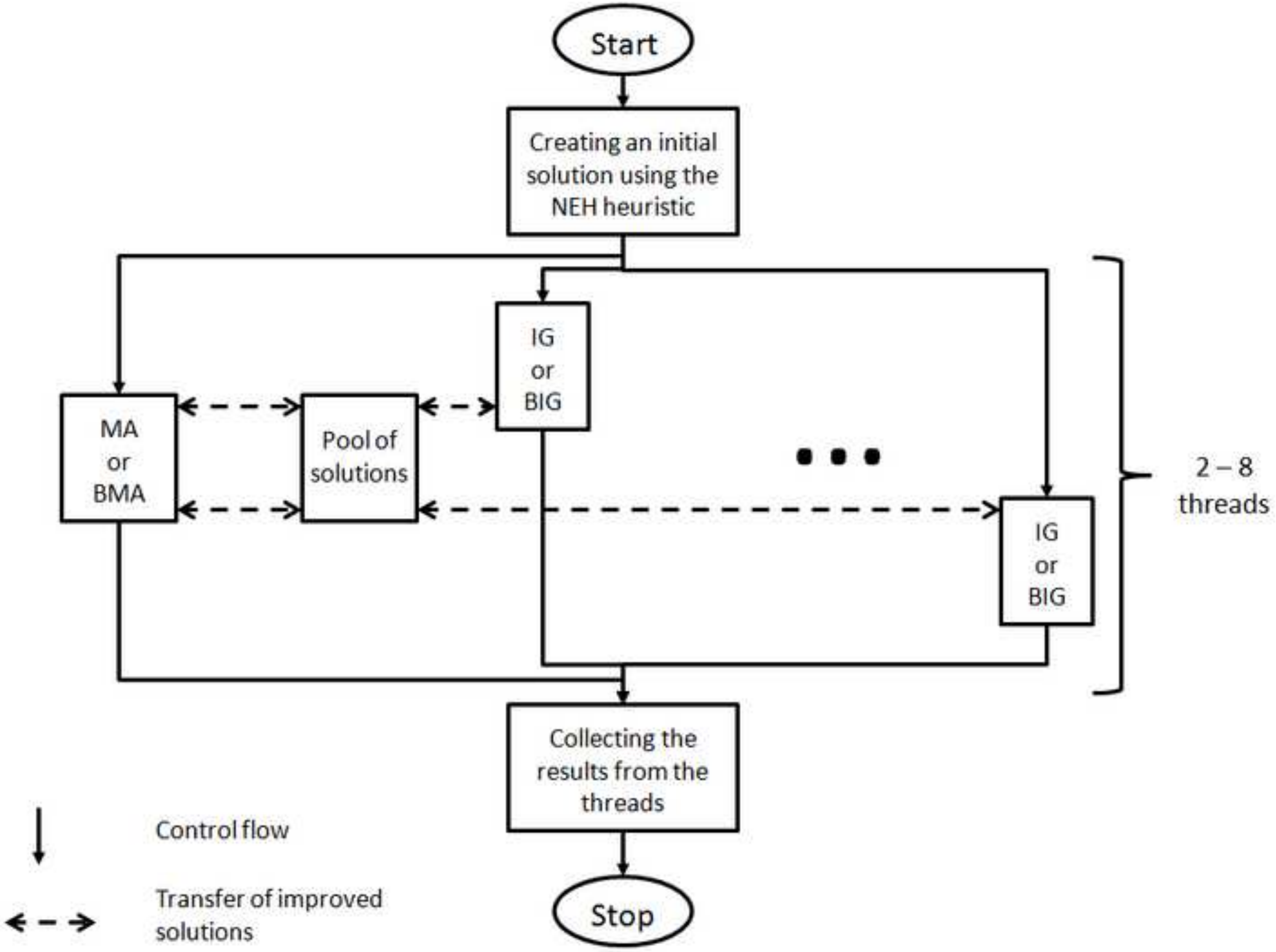
```
\blindtext@numBlindtext=\count105
\blind@countyy=\count106
\c@blindlist=\count107
\c@blindlistlevel=\count108
\c@blindlist@level=\count109
\blind@listitem=\count110
\c@blind@listcount=\count111
\c@blind@levelcount=\count112
\blind@mathformula=\count113
\blind@Mathformula=\count114
) (./balazs_et_al.aux)
\openout1 = `balazs_et_al.aux'.

LaTeX Font Info:    Checking defaults for OML/cmm/m/it on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Checking defaults for T1/cmr/m/n on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Checking defaults for OT1/cmr/m/n on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Checking defaults for OMS/cmsy/m/n on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Checking defaults for OMX/cmex/m/n on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Checking defaults for U/cmr/m/n on input line 92.
LaTeX Font Info:    ... okay on input line 92.
LaTeX Font Info:    Try loading font information for U+msa on input line 165.
(c:/texlive/2009/texmf-dist/tex/latex/amsfonts/umsa.fd
File: umsa.fd 2009/06/22 v3.00 AMS symbols A
)
LaTeX Font Info:    Try loading font information for U+msb on input line 165.
(c:/texlive/2009/texmf-dist/tex/latex/amsfonts/umsb.fd
File: umsb.fd 2009/06/22 v3.00 AMS symbols B
)
LaTeX Font Info:    Try loading font information for OMS+cmr on input line 165.

(c:/texlive/2009/texmf-dist/tex/latex/base/omscmr.fd
File: omscmr.fd 1999/05/25 v2.5h Standard LaTeX font definitions
)
LaTeX Font Info:    Font shape `OMS/cmr/m/it' in size <10> not available
(Font)              Font shape `OMS/cmsy/m/n' tried instead on input line 165.
[1


]

Package natbib Warning: Citation `Johnson54' on page 2 undefined on input line
172.


Package natbib Warning: Citation `Kan76' on page 2 undefined on input line 176.
Package natbib Warning: Citation `Taillard90' on page 2 undefined on input line
 176.


Package natbib Warning: Citation `Juan10' on page 2 undefined on input line 176
.


Package natbib Warning: Citation `Horvath11' on page 2 undefined on input line
176.


Package natbib Warning: Citation `Taillard90' on page 2 undefined on input line
 178.


Package natbib Warning: Citation `Juan10' on page 2 undefined on input line 178
.


Package natbib Warning: Citation `Horvath11' on page 2 undefined on input line
178.


Package natbib Warning: Citation `Taillard93' on page 2 undefined on input line
 178.


Package natbib Warning: Citation `Balazs12APH' on page 2 undefined on input lin
e 180.


Package natbib Warning: Citation `Balazs12CEC' on page 2 undefined on input lin
e 184.

[2]

Package natbib Warning: Citation `Ravetti10' on page 3 undefined on input line
186.


Package natbib Warning: Citation `Ravetti10' on page 3 undefined on input line
188.


Package natbib Warning: Citation `Balazs12APH' on page 3 undefined on input lin
e 188.


Package natbib Warning: Citation `Balazs10IUM' on page 3 undefined on input lin
e 188.
```

```
Package natbib Warning: Citation `Balazs10flat' on page 3 undefined on input li
ne 188.


Package natbib Warning: Citation `Balazs12CINTI1' on page 3 undefined on input
line 190.

[3]

Package natbib Warning: Citation `Johnson54' on page 4 undefined on input line
211.


Package natbib Warning: Citation `Balazs12APH' on page 4 undefined on input lin
e 233.

[4]

Package natbib Warning: Citation `Holland92' on page 5 undefined on input line
239.


Package natbib Warning: Citation `Nawa99' on page 5 undefined on input line 239
.


Package natbib Warning: Citation `Holland92' on page 5 undefined on input line
253.


Package natbib Warning: Citation `Nawa99' on page 5 undefined on input line 284
.


Package natbib Warning: Citation `Balazs10IUM' on page 5 undefined on input lin
e 285.


Package natbib Warning: Citation `Balazs10flat' on page 5 undefined on input li
ne 285.


Package natbib Warning: Citation `Balazs12APH' on page 5 undefined on input lin
e 285.

[5]

Package natbib Warning: Citation `Moscato89' on page 6 undefined on input line
326.
```

```
Package natbib Warning: Citation `Balazs12APH' on page 6 undefined on input lin
e 340.


Overfull \hbox (5.27379pt too wide) in paragraph at lines 340--341
[]\OT1/cmr/m/n/12 In case of the PSFP prob-lem two types of in-di-vid-ual rep-r
e-sen-ta-tion (i.e. two
 []

[6]

Package natbib Warning: Citation `Balazs12APH' on page 7 undefined on input lin
e 348.


Overfull \hbox (7.20699pt too wide) in paragraph at lines 348--349
[]\OT1/cmr/m/n/12 However, de-spite the com-pu-ta-tional over-head, this en-cod
-ing man-ner turned
 []


Package natbib Warning: Citation `Balazs12APH' on page 7 undefined on input lin
e 354.

LaTeX Font Info:    Font shape `OMS/cmr/m/n' in size <12> not available
(Font)              Font shape `OMS/cmsy/m/n' tried instead on input line 362.
[7]

Package natbib Warning: Citation `Ruiz07' on page 8 undefined on input line 379
.


Package natbib Warning: Citation `Ruiz07' on page 8 undefined on input line 420
.


Package natbib Warning: Citation `Nawaz83' on page 8 undefined on input line 42
4.


Overfull \hbox (1.74464pt too wide) in paragraph at lines 424--425
[]\OT1/cmr/m/n/12 An ini-tial per-mu-ta-tion is cre-ated by us-ing the de-ter-m
in-is-tic NEH heuris-
 []


Package natbib Warning: Citation `Ruiz07' on page 8 undefined on input line 441
.

[8]
```

```
Package natbib Warning: Citation `Balazs12CEC' on page 9 undefined on input lin
e 447.

[9]

Package natbib Warning: Citation `Ravetti10' on page 10 undefined on input line
 479.


Package natbib Warning: Citation `Nawaz83' on page 10 undefined on input line 4
84.


Package natbib Warning: Citation `Ravetti10' on page 10 undefined on input line
 495.

[10]

Package natbib Warning: Citation `Balazs12CINTI1' on page 11 undefined on input
 line 518.


Package natbib Warning: Citation `Balazs12APH' on page 11 undefined on input li
ne 518.

[11]

! LaTeX Error: File `illustration' not found.

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.535 ...graphics[width=1\textwidth]{illustration}

I could not locate the file with any of these extensions:
.eps,.ps,.eps.gz,.ps.gz,.eps.Z
Try typing  <return>  to proceed.
If that doesn't work, type  X <return>  to quit.


Package natbib Warning: Citation `Taillard93' on page 12 undefined on input lin
e 548.

[12]

Package natbib Warning: Citation `Ruiz07' on page 13 undefined on input line 55
2.


Package natbib Warning: Citation `Taillard93' on page 13 undefined on input lin
e 565.
```

```
Overfull \hbox (25.50098pt too wide) in paragraph at lines 565--565
[][]\OT1/cmr/m/n/10 http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancem
ent.dir/ordonnancement.html|
 []


Package natbib Warning: Citation `Ruiz07' on page 13 undefined on input line 56
5.

[13]

Package natbib Warning: Citation `Balazs12CINTI1' on page 14 undefined on input
 line 580.

[14]

Package natbib Warning: Citation `Balazs12CINTI1' on page 15 undefined on input
 line 692.

[15]
No file balazs_et_al.bbl.

Package natbib Warning: There were undefined citations.

[16] (./balazs_et_al.aux) )
Here is how much of TeX's memory you used:
 1774 strings out of 493849
 21226 string characters out of 3152231
 88490 words of memory out of 3000000
 5027 multiletter control sequences out of 15000+200000
 13870 words of font info for 52 fonts, out of 3000000 for 9000
 714 hyphenation exceptions out of 8191
 32i,9n,34p,1126b,276s stack positions out of 5000i,500n,10000p,200000b,50000s

Output written on balazs_et_al.dvi (16 pages, 49736 bytes).
```


Start
Creating an initial solution using the NEH heuristic
IG or BIG
MA or BMA
Pool of solutions
IG or BIG
2 – 8 threads
Collecting the results from the threads
Stop
Control flow
Transfer of improved solutions