Metaheuristic Optimization - CAT 3

Metaheuristics

Óscar Gómez Álvarez
Juan José Rodriguez Aldavero

September 23, 2022

Contents

1 Reading and assessment of papers

1.a Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues

The PFSP is a combinatorial optimization problem where the objective is to organize a set of k tasks distributed around m independent machines, each with a particular processing time p_{ij}. Each machine can execute a single job, and the goal is to find a arrangement of jobs such that some parameter (usually the total execution time) is minimized.

Historically, a multitude of algorithms have been used to deal with this problem, the most efficient requiring a high level of fine-tuning. However, there is a set of algorithms within the Iterated Local Search (ILS) framework that require a reduced number of tuning parameters and are easy to understand and implement.

The authors propose a new cathegory of ILS algorithms named ILS-ESP (Easy, Simple, Parallelizable) that are just an adaptation of ILS algorithms using simpler rules and biased randomization (in particular, the use of modified NEH heuristics to generate random initial solutions).

In some ways, this type of algorithms (ILS-ESP) can be seen as a hybridation of ILS with GRASP techniques. For this reason, it is appropriate to discuss some of their properties: GRASPs (Greedy Randomized Adaptive Search Procedures) are a cathegory of multistart algorithms that are used to solve difficult combinatorial optimization problems. They are built around two stages: the first constructive stage consists on the creation of new solutions (good but not optimal) by inserting elements over a partial solution. The second stage is the search for these elements in in the solutions space over the environment of the previous solution. These phases follow iteratively one after the other until a sufficiently good solution is found. Also, each element from the construction phase is added following a biased random procedure, using for example geometrical distributions for the selection. This, in turn, produces new solutions for every execution.

The ILS-ESP algorithm has several interesting properties. The first one is the possibility to use parameters that do not need fine-tuning. This considerably improves the simplicity of the algorithm, as one of the main complexities of more classical algorithms is this fine-tuning process. Another property is its high degree of flexibility, that is, being able to use the same algorithm after adding new constraints to the problem. Finally, there is another motivation for the development of this algorithm: it allows the use of parallelization techniques, which dramatically increases the execution speed by means of specific hardware such as GPUs or multi-core processors.

The ILS-ESP algorithm relies on ILS and then complements it with GRASP-based techniques. We can outline four critical phases, which are: 1) generation of an initial solution, 2) local search, 3) perturbation and 4) acceptation criteria.

Let us first look the perturbation stage, which employs a so-called "enhanced swap" operator to exchange two randomly selected jobs and apply heuristic operation such as NEH to find the best position of that job within the queue.

Then, there is the acceptation stage, which consists on the update of the best available solution, baseSol, with a new one, aSol, if it is better or if it is worse but there haven't been two consecutive degradations (in order to escape local minima).

A third critical phase of the algorithm is the creation of the initial solution. Traditionally, an initial solution was created just using NEH heuristics over all the queues, but in order not to always start with the same solution and be able to apply parallelization techniques, a modified version of NEH heuristics is used. It applies biased randomization techniques in order to maintain the logic behind this heuristic but create different solutions for each execution. In more detail, a NEH heuristic is used along with a triangular distribution to spread the selection probabilities (unlike the NEH heuristic that simply orders the works from longer to shorter execution time).

The final computational cost of this algorithm is polynomial, so that it can be compared to other algorithms based on local search. In order to test the proposed algorithm, a brief computational experiment in the Java language is done. The ILS-ESP is compared with other more classical algorithms as well as with a modified algorithm using biased randomization. The authors propose the experiment for two different metrics: BEST10 (best experimental solution found) and AVG10 (average value of the different replicas) while we will only show results for the former metric. The results are as follows:

Taillard set	RandIG-D4T04 (%)	IG-D4T04 (%)	ILS-ESP (%)	ILS98-T04 (%)	IG-D2T03 (%)	Maximum time (seconds)
20_5	0.00	0.04	0.00	0.04	0.00	1
20_10	0.00	0.00	0.00	0.00	0.04	2
20_20	0.00	0.01	0.00	0.00	0.03	4
50_5	0.00	0.00	0.00	0.00	0.00	2.5
50_10	0.38	0.48	0.47	0.45	0.53	5
50_20	0.62	0.66	0.71	0.74	1.00	10
100_5	0.00	0.01	0.00	0.01	0.00	5
100_10	0.10	0.07	0.10	0.07	0.10	10
100_20	0.94	1.04	1.13	1.07	1.25	20
200_10	0.08	0.08	0.09	0.11	0.14	20
200_20	1.18	1.29	1.24	1.32	1.48	40
500_20	0.62	0.63	0.63	0.67	0.71	100
Averages	0.33	0.36	0.36	0.37	0.44	–

Figure 1.1: *Computational results for a collection of Taillard data sets*

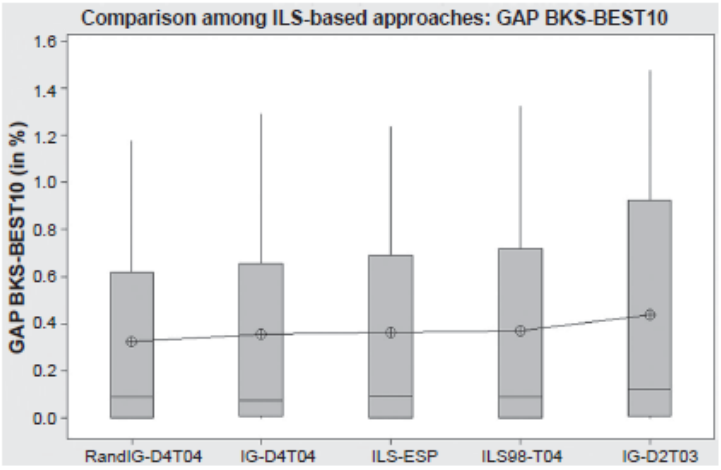


Figure 1.2: *Box plots for the previous results*

We see how both the ILS-ESP algorithm and the randomized version of IG-D4T04 give very satisfying results, compared to more classical solutions.

The fact that randomization techniques have been used gives us the possibility to do another type of computational experiment to test the effects or parallelization (in particular of the number of parallel agents). The results can be seen in the following chart:

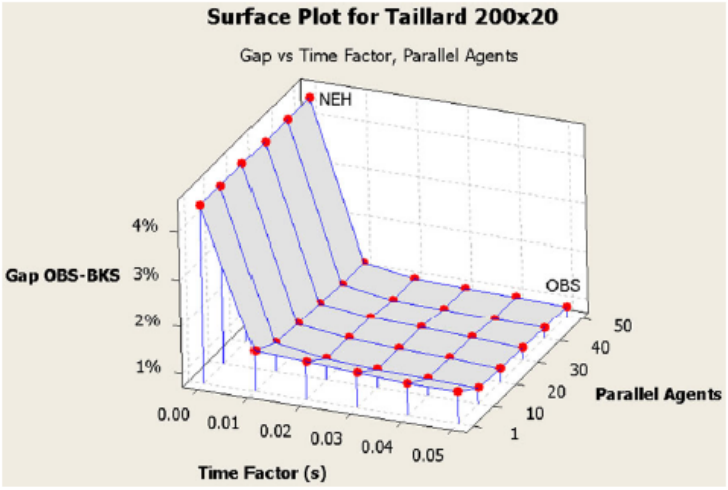


Figure 1.3: *Surface plot for the* 200×20 *Taillard data set*

We can see how the time needed to reduce most of the gap is in the order a hundreth of a second in this experiment, and that incrementing the parallel agents can help reduce even further these costs.

In brief, we can appreciate how ILS-based algorithms bring significant advantages when solving PFSP problems, such as computational efficiency and simplicity (no need to fine-tune parameters). In addition, parallelization brings extra advantages, like execution speed and the fact that solutions can be found "in real time".

1.b Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem

In this paper, the authors plan to address the MDVRP problem with ILS metaheuristics combined with biased randomization using theoretical geometric probability distributions, particularly in two distinct phases of the algorithm.

The MDVRP problem is a generalization of Capacitated VRP for multiple depot nodes: Given a set of customers, the goal is to establish a set of closed routes between customers and depot nodes in such a way that the demands of all customers are met by means of a visit from one of the vehicles in the fleet, and in such a way that vehicles leave and return from the same depot node. This adds an extra complication in the form of combinatorial allocation: a set of customers must be assigned to each of the depot nodes. There are a multitude of additional restrictions which, depending on the complexity of the problem, can be added or removed.

The fact that the authors approach this problem with probabilistic methods means that each time it is executed a different solution is obtained, and opens the door to the use of parallelization

methods that considerably increase the efficiency of the algorithm. In addition, using biased probability distributions such as geometry allows the logic behind the original heuristics to be retained. This work uses Monte Carlo techniques, which are merely techniques that, based on probabilistic procedures by means of random number generators and probability distributions, allow us to solve deterministic problems in a stochastic way. Monte Carlo techniques, when combined with heuristic algorithms, provide very powerful approaches to solving computational problems.

For example, the SR-GCWS-CS algorithm deals with VRP problems using a CWS heuristic combined with a Monte Carlo technique: a geometric probability distribution combined with a random number generator. It is very important to maintain the logic of the original heuristic after applying the Monte Carlo method in order to keep its advantages.

In this work, we apply a similar procedure for the MDVRP problem, as we will explain below. This procedure is based on the Iterated Local Search (ILS) framework, in order to generate a random initial solution (a) and to incorporate a stochastic perturbation procedure that maintains the CWS logic (b).

The first step consists in the generation of a list of potential nodes for each depot. These nodes will be ordered according to a magnitude called marginal gains which is calculated depending on the relative position of each node with respect to all depots. In this way, nodes very close to depot k will have high marginal gains for this node, and low gains for the rest; and nodes located between several depots will have low marginal gains for all of them.

The next step is the application of a biased randomization procedure to the previous process. To do this, the authors use a geometric distribution which is applied to the ordered list defined above. Then, on this randomized list a selection procedure is applied to definitively assign the nodes to each one of the depots and create the allocation map. There are different procedures, each with its advantages and disadvantages. We can highlight the round-robin tournament procedure that allows depots to be assigned nodes individually; allocation depending on the capacity of each of the depots, giving more balanced allocation maps.

Once this is done, we will have a set of single depot VRP problems, which can be solved with a CWS heuristic to generate the first initial solution. This procedure is very fast and generates reasonably good solutions. Once we have this first solution, we apply the perturbation procedure within the ILS environment with the CWS heuristic to iteratively improve the initial solution until reaching a termination criterion. It is important to see that the disturbance procedure not only changes the routes generated by the CWS heuristic, but also changes the node allocation map generated in step 1, as long as this leads to an improvement of the initial solution. This is characterized by assigning a percentage p^* of the nodes to a different depot for each iteration.

Finally, the five best solutions obtained by this procedure are saved for further post-optimization using the SR-GCWS-CS algorithm to achieve the best possible performance.

In summary, the procedure followed is as follows:

1. Generate a list of potential nodes using the marginal gain metric.
2. Apply a biased randomization by means of a geometric distribution.
3. Generate the allocation maps by a selection procedure (round-robin or by capacity).
4. Solve the allocation maps using CWS to obtain the initial solution.
5. Initiate an ILS procedure to apply perturbations iteratively, using CWS heuristics and a selection procedure, generating new allocation maps and new solutions slightly different from the initial one.
6. Once the termination criteria is reached, save the five best solutions and apply a post-optimization using the SR-GCWS-CS algorithm.

In order to study the efficiency of this process, a computational experiment is made from a data set containing 33 MDVRP problems solved by classical algorithms. The authors solved each problem five times using the abovementioned algorithm, with a limit of 300,000 iterations and choosing the parameters stochastically from values given by a quick and rough fine-tuning. The results are as follows:

Instance						Other approaches			Our ILS approach			
Instances	*n*	*m*	*d*	Maximum route length	*Q*	CGL97 (1)	PR07 (2)	VCGLR11 (3)	OBS (4)	Gap (%) (1)–(4)	Gap (%) (2)–(4)	Gap (%) (3)–(4)
p01	50	4	4	n/a	80	576.87	576.87	576.87	576.87	0.00	0.00	0.00
p02	50	2	4	n/a	160	473.53	473.53	473.53	473.87	0.07	0.07	0.07
p03	75	3	5	n/a	140	641.19	641.19	641.19	641.19	0.00	0.00	0.00
p04	100	8	2	n/a	100	1001.59	1001.04	1001.04	1003.45	0.19	0.24	0.24
p05	100	5	2	n/a	200	750.03	751.26	750.03	751.9	0.25	0.09	0.25
p06	100	6	3	n/a	100	876.5	876.7	876.5	876.5	0.00	−0.02	0.00
p07	100	4	4	n/a	100	885.8	881.97	881.97	885.19	−0.07	0.37	0.37
p08	249	14	2	310	500	4437.68	4387.38	4372.78	4409.23	−0.64	0.50	0.83
p09	249	12	3	310	500	3900.22	3873.64	3858.66	3882.58	−0.45	0.23	0.62
p10	249	8	4	310	500	3663.02	3650.04	3631.11	3646.67	−0.45	−0.09	0.43
p11	249	6	5	310	500	3554.18	3546.06	3546.06	3547.09	−0.20	0.03	0.03
p12	80	5	2	n/a	60	1318.95	1318.95	1318.95	1318.95	0.00	0.00	0.00
p13	80	5	2	200	60	1318.95	1318.95	1318.95	1318.95	0.00	0.00	0.00
p14	80	5	2	180	60	1360.12	1360.12	1360.12	1360.12	0.00	0.00	0.00
p15	160	5	4	n/a	60	2505.42	2505.42	2505.42	2511.92	0.26	0.26	0.26
p16	160	5	4	200	60	2572.23	2572.23	2572.23	2573.78	0.06	0.06	0.06
p17	160	5	4	180	60	2709.09	2709.09	2709.09	2709.09	0.00	0.00	0.00
p18	240	5	6	n/a	60	3702.85	3702.85	3702.85	3702.85	0.00	0.00	0.00
p19	240	5	6	200	60	3827.06	3827.06	3827.06	3840.91	0.36	0.36	0.36
p20	240	5	6	180	60	4058.07	4058.07	4058.07	4063.64	0.14	0.14	0.14
p21	360	5	9	n/a	60	5474.84	5474.84	5474.84	5574.63	1.82	1.82	1.82
p22	360	5	9	200	60	5702.16	5702.16	5702.16	5737.04	0.61	0.61	0.61
p23	360	5	9	180	60	6095.46	6078.75	6078.75	6084.32	−0.18	0.09	0.09
Average gap p01–p23										*0.08*	*0.21*	*0.27*
pr01	48	1	4	500	200	–	861.32	861.32	861.32	–	0.00	0.00
pr02	96	2	4	480	195	–	1307.34	1307.34	1307.34	–	0.00	0.00
pr03	144	3	4	460	190	–	1806.53	1803.80	1803.80	–	−0.15	0.00
pr04	192	4	4	440	185	–	2060.93	2058.31	2072.10	–	0.54	0.67
pr05	240	5	4	420	180	–	2337.84	2331.20	2342.20	–	0.19	0.47
pr06	288	6	4	400	175	–	2685.35	2676.30	2687.73	–	0.09	0.43
pr07	72	1	6	500	200	–	1089.56	1089.56	1089.56	–	0.00	0.00
pr08	144	2	6	475	190	–	1664.85	1664.85	1668.08	–	0.19	0.19
pr09	216	3	6	450	180	–	2136.42	2133.20	2133.56	–	−0.13	0.02
pr10	288	4	6	425	170	–	2889.49	2868.26	2889.70	–	0.01	0.75
Average gap p01–p23											*0.07*	*0.25*
Average gaps for all problems											*0.17*	*0.26*

Figure 1.4: *Results for the computational experiment*

We see how 12 of the 33 best classical solutions can be reached, resulting in very reduced gaps and very short execution times, as opposed to the computational effort required to use classical methods. In addition, by using biased randomization, parallelization can be applied, resulting in these results:

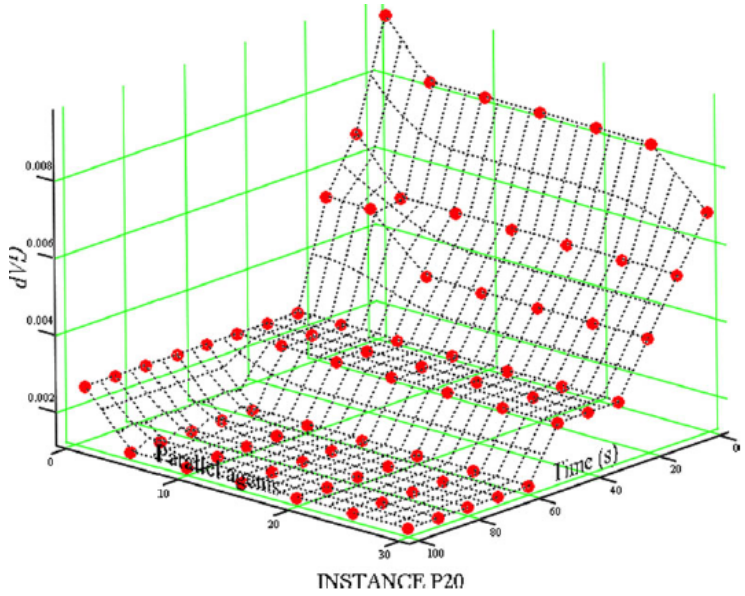


Figure 1.5: *Surface plot for the p20 instance, varying parallel agents*

We see how the use of parallel agents decreases significantly the execution time and improves the quality of the problem.

In summary, the method shown achieves competitive solutions with respect to the best classical solutions and offers very interesting advantages such as short execution times, simplicity and little need for fine-tuning, as well as the possibility of parallelization. It is important to point out that this methodology can be applied to other problems of combinatorial optimization by simply substituting the particular heuristics with others that are more adapted, but maintaining the same structure.

1.c Routing fleets with multiple driving ranges: Is it possible to use greener fleet configurations?

In this article the authors present the use of metaheuristic algorithms with biased randomization to approach the Vehicle Routing Problem with Multiple Driving Ranges (VRPMDR), an extension of the classic Vehicle Routing Problem where the distance each vehicle can travel is limited. This is an issue that can be particular to fleets of electric and hybrid vehicles, because their main drawback is the limited range and the impossibility to refuel quickly.

To deal with this problem, we start using the algorithms applied to the simple VRP problem. More exactly, following a particular strategy we can decompose a VRPMDR problem into a set of CVRP problems with restrictions. As there are a multitude of metaheuristics adapted to this latter problem, it can be simplified considerably. In this case, Juan et Al's SR-GCWS algorithm will be used for each of the CVRPs along with restrictions on route length and without applying the splitting techniques. Furthermore, it can be shown that the simplest algorithm CVRP algorithm is able to offer acceptable solutions fro the decomposed VRPMD problem.

In order to address the problem, an iterative approach has been considered: we start with a simple problem and gradually increase the conditions over it. The authors begin with a fleet of vehicles of unlimited range, and then gradually reduce the range to form a fleet of intermediate

range and another of short range. For each of these fleets, the associated CVRP* problem is solved. The algorithm followed is:

1. Define the CVRP* for the fleet of vehicles used
2. Resolve the CVRP* using one of the available algorithms.
3. Assign the routes to the fleet vehicles ordered from highest to lowest range. By design, there will be routes without assigned vehicles because the range of the vehicle is less than the length of the route.
4. Save the routes and nodes in memory, and repeat the CVRP problem* for all nodes that have not been assigned vehicles.
5. Repeat the above steps until all nodes are served.
6. Create a final solution.

We can see this algorithm sketched in the following graph:

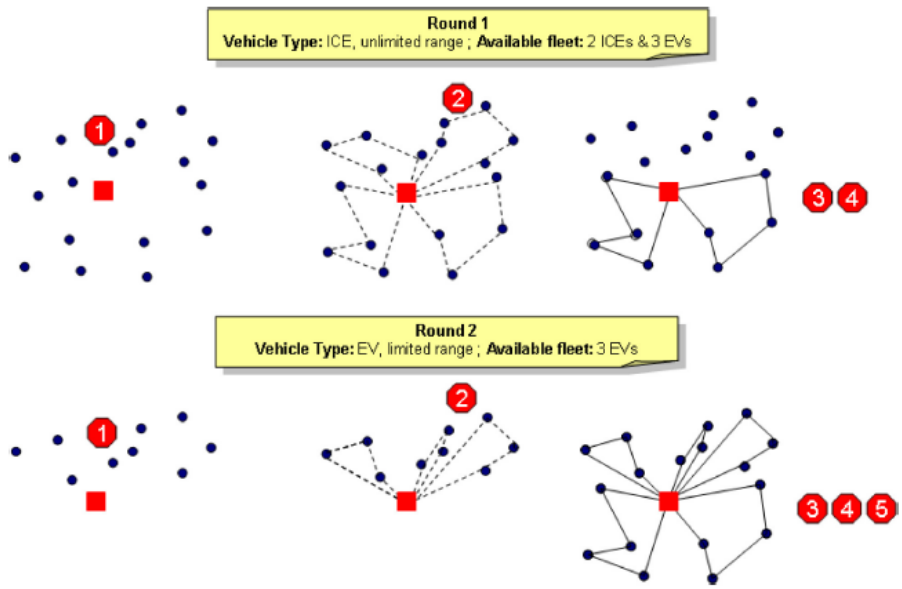


Figure 1.6: *Algorithm used for the decomposition of the VRPMDR problem*

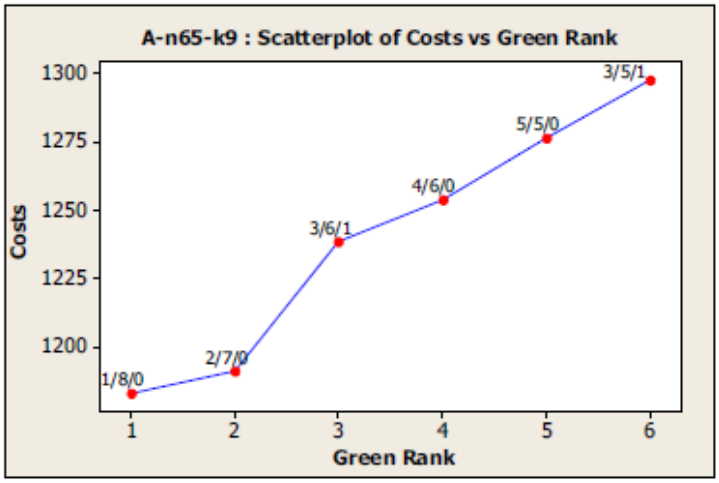
To test the efficiency of the algorithm, the authors performed a computational experiment in Java and the algorithm was compared with a number of classical results selected from a public source, corresponding to quasi-optimal solutions. The computational experiment is considered for three categories of vehicles: ICE type unlimited range vehicles, electric type intermediate range vehicles and short range vehicles, also electric. The results are as follows:

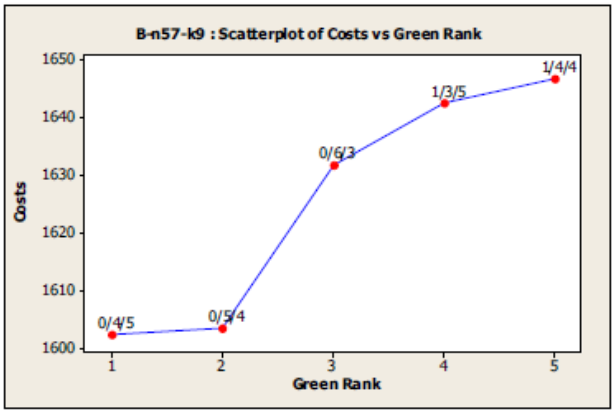
(a) Name	(b) # Nodes	(c) Capacity	(d) BKS Cost	(e) Fleet configuration S/M/L	(f) OS cost	(g) Gap (d)–(f)
A-n32-k5	32	100	787.81	2/1/2	787.08	–0.09%
				1/3/1	829.41	5.28%
				0/5/0	733.95	–0.03%
A-n38-k5	38	100	734.18	1/3/1	734.18	0.00%
				1/4/0	735.05	0.12%
				3/3/0	763.13	3.94%
A-n65-k9	65	100	1181.69	1/8/0	1183.31	0.14%
				2/7/0	1191.27	0.81%
				3/6/1	1238.33	4.79%
				4/6/0	1253.81	6.10%
				5/5/0	1276.21	8.00%
				3/5/1	1297.31	9.78%
A-n80-k10	80	100	1766.50	2/5/3	1776.19	0.55%
				1/7/2	1785.05	1.05%
				2/5/0	744.23	–0.07%
B-n50-k7	50	100	744.78	3/4/0	744.67	–0.01%
				4/3/0	751.24	0.87%
B-n52-k7	52	100	750.08	4/2/1	752.63	0.34%
				3/4/0	756.71	0.88%
				0/4/5	1602.29	–0.08%
B-n57-k9	57	100	1603.63	0/5/4	1603.37	–0.02%
				0/6/3	1631.66	1.75%
				1/3/5	1642.53	2.43%
				1/4/4	1646.65	2.68%
B-n78-k10	78	100	1229.27	4/5/1	1236.33	0.57%
				3/7/0	1251.83	1.83%
				4/4/2	1252.76	1.91%
				4/6/0	1253.10	1.94%
				6/5/0	1292.60	5.15%

Figure 1.7: *Computational results for 7 classical data sets.*

The first four columns correspond to the classical results, while the column (e) and the column (f) are the greener fleet's configuration and cost. We see how the proposed algorithm obtains cost (f) results very similar to those obtained through classical procedures, sometimes even better as seen in the gap (g).

In a second round of evaluation, the authors create new problems by increasing the number of electric vehicles instead of using the configuration that minimises the cost. The result is that our algorithm is able to use greener fleets with a very little increase in costs. In the following two charts we show the evolution of the costs when increasing electring vehicles for two classical data sets:





In conclusion, the proposed VRPMDR algorithm is a pioneer in tackling VRP type problems with length limitations and opens the door to a multitude of new lines of research, among which are:

1. Optimisation of the type of vehicles in a fleet
2. Include new conditions such as a hypothetical emission tax when solving VRP problems.
3. Study the effect of node topography and customer density on the potential impact of electric vehicle fleets.

In my opinion, the fact that new VRP algorithms allow us to optimise something as important as the ecological impact of certain types of logistics operations brings significant added value to this type of research. The fact that one of the greatest current world trend is to reduce the environmental impact of greenhouse gas emissions as much as possible makes this type of research lines extremely interesting from a social as well as technological point of view.

2 Design and development of an algorithm

2.a Title

An algorithm for solving the permutation flow shop problem with local search.

2.b Introduction

The Permutation Flowshop Sequencing Problem (PFSP) is one of the most thoroughly studied problems in combinatorial optimisation. Its simple definition conceals a very difficult problem which has attracted the attention of numerous researchers in the past decades. It can be described as follows: a set $J = \{1, ..., n\}$ of n independent jobs has to be processed on a set $M = \{1, ..., m\}$ of m independent machines. Every job $j, j \in J$, requires a given fixed processing time $p_{ij} \geq 0$ on every machine $i, i \in M$. Each machine can execute at most one job at a time, and it is assumed that the jobs are processed by all machines in the same order they have been processed by the first machine. The classical goal is to find a sequence for processing the jobs in the shop so that a given criterion is optimized. The criterion that is most commonly used is the minimization of the maximum completion time (makespan) of the production sequence denoted by C_{max}. Figure 2.1 illustrates this problem for a simple case of $n = 3$ jobs and $m = 3$ machines.

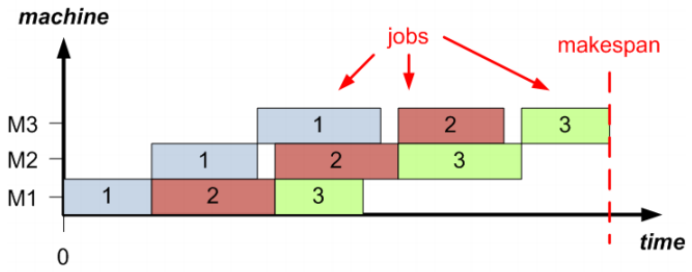


Figure 2.1: *Flowshop Sequencing Problem.*

The described problem is usually denoted as $Fm|prmu|C_{max}$, using the notation proposed by Graham et al. (1979). It is a combinatorial problem with $n!$ possible sequences which, in general, is NP-complete. Similar to what has happened with other combinatorial problems, a large number of different approaches have been developed to deal with the PFSP. These approaches range from the use of pure optimization methods, such as mixed integer programming, for solving small-sized problems, see Reddi and Ramamoorthy (1972), to the use of heuristics and meta-heuristics that provide near-optimal solutions for medium and large-sized problems, Nawaz et al. (1983).Most of these methods focus on minimizing makespan. However, utility functions of decision-makers are difficult to model and, frequently, criteria other than total time employed to process the list of jobs should be also considered in real-life scenarios. For that reason, there is a need for more flexible methods able to provide a large set of alternative near-optimal solutions with different properties, so that decision-makers can choose among different alternative solutions according to their specific necessities and preferences.

2.c Literature review

A large number of heuristics and metaheuristics have been proposed to solve the PFSP, mainly because of the difficulty encountered by exact methods to solve medium or large instances. As explained before, most existing approaches focus on the objective of minimizing makespan. Nawaz et al. (1983) introduced the NEH heuristic, which is commonly considered as the best performing heuristic for the PFSP. Basically, what the NEH heuristic proposes is to calculate the total processing time required for each job (i.e., the total time each jobs requires to be processed by the set of machines), and then to create an "efficiency list" of jobs sorted in a descending order according to this total processing time. At each step, the job at the top of the efficiency list is selected and used to construct the solution, that is: the "common sense" rule is to select first those jobs with the highest total processing time. Once selected, a job is inserted in the sorted set of jobs that are configuring the on-going solution. The exact position that the selected job will occupy in that ongoing solution is given by the minimizing makespan criterion. Taillard (1990) introduced a data structure that reduces the NEH complexity. Other interesting heuristics are those from Suliman (2000) or Framinan and Leisten (2003), which also consider several extensions of NEH when facing objectives other than makespan.

Different metaheuristic approaches have been also proposed for the PFSP. Osman and Potts (1989) used Simulated Annealing. Widmer and Hert (1989) proposed a Tabu Search algorithm known as SPIRIT. Other Tabu Search algorithms, which make use of the NEH heuristic, were introduced by Taillard (1990) and Reeves (1993). Chen et al. (1995), Reeves (1995) and Aldowaisan and Allahvedi (2003) proposed the use of Genetic Algorithms for solving the PFSP which were also based on the NEH heuristic. As examples of other works which also rely on the use of the NEH heuristic we can cite the Ant Colony Optimization algorithm of Chandrasekharan and Ziegler (2004).

All the aforementioned work has in common that the algorithms proposed are easy to code and therefore the results can be reproduced without too much difficulty. In addition, many of the above algorithms can be adapted to other more realistic flowshop environments Ruiz and Maroto (2005). Additionally, there are other highly elaborated hybrid techniques for solving the PFSP. However, as Ruiz and Stützle (2007) point out, "they are very sophisticated and an arduous coding task is necessary for their implementation". In other words, it is unlikely that they can be used for solving realistic scenarios without direct support from the researchers that developed them.

2.d Algorithm developed

The approach presented in this assessment aims to provide a simple probabilistic algorithm that will improve the results obtained with the NEH heuristic in a few iterations. This is a multi-start algorithm based on a randomized construction process, which is combined with a local search. Basically, we initiate a search process inside the space of feasible solutions. Each such solution will consist of a list of jobs sorted in some order. As explained before, the NEH heuristic is an iterative algorithm which employs a list of jobs sorted by their total completion time on all the machines to construct a solution for the PFSP. At each step of this iterative process, the NEH removes the job which is at the top of that list (with maximum completion time) and adds it to a new list at the position that results in the best partial solution with respect to makespan. As a result, the NEH provides a "common sense" deterministic solution, by trying to schedule the most demanding jobs first. Our approach, instead, assigns a probability of selecting each job in the list. According to our design, this probability should be coherent with the total time that each job needs to be processed by all the machines, i.e., jobs with higher total times will be more likely to be selected from the list before those with lower total times. The method employs a discrete version of a triangular distribution as well as the geometric statistical distribution with

parameter β ($0 < \beta < 1$). It is used during the solution-construction process: each time a new job has to be selected from the list, a triangular or geometric distribution is randomly selected. This distribution is then used to assign exponentially diminishing probabilities to each eligible job according to its position inside the list, which has been previously sorted by its corresponding total-processing-time value. That way, jobs with higher processing times are always more likely to be selected from the list, but the probabilities assigned are variable and they depend upon the concrete distribution selected at each step. By iterating this procedure, an oriented random search process is started. The resulting job orderings is called Randomized NEH solutions.

After we generate a solution by Randomized NEH (RNEH), we try to further optimize it by applying a local search which in our case is the same method used by many other authors, e.g., Ruiz and Stützle (2007). In the local search, the shift-to-left operator proposed in Juan et al. (2014) is utilized. The main idea behind this operator is to iteratively examine all the jobs and try to shift them to the left (Figure 2.3). If the job movement improves the makespan, the movement is accepted, otherwise the solution is not modified. Roughly speaking, the local search consists in an iterative process whose pseudocode is shown in Algorithm 1 and it is explained with the following steps at each iteration (Figure 2.2):

1. A position *k* is randomly selected from the permutation of jobs which define the current solution.
2. A 'shift-to-left' movement is applied on the job at position *k*.
3. Finally, the job is set at the position providing the best makespan –including the original one if no improvement is reached. These steps are iterated until all positions have been visited or an improvement is reached – in the latter case, the set of already visited positions becomes empty and the entire process is restarted.

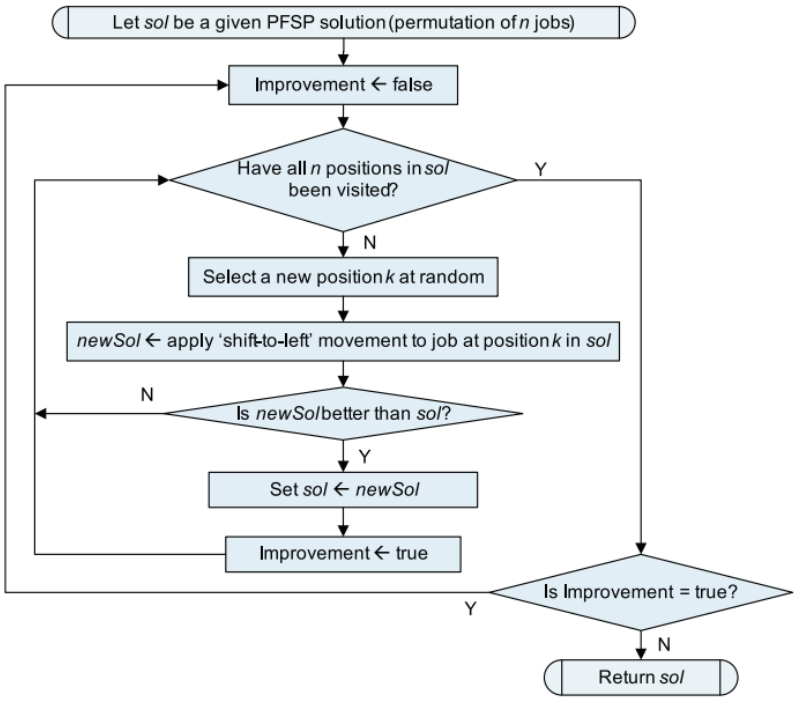


Figure 2.2: *Flowchart diagram of the local search process.*

The local search process is repeated as long as we keep getting better arrangements of the jobs. Note that this second stage of our algorithm will never output a solution that is worse than the

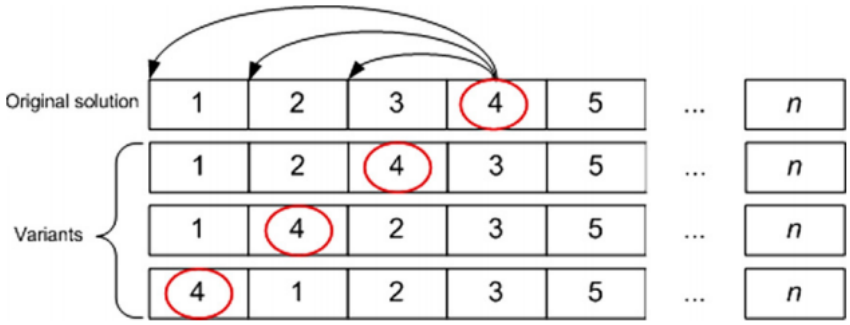


Figure 2.3: *Shift-to-left operator.*

Algorithm 1 Local search to minimize makespan

```
1:  procedure localSearch(currentSolution)
2:      baseSol ← copySolution(currentSolution)
3:      newSol ← copySolution(currentSolution)
4:      positions ← calcPositionsRandom(distribution)        ▷ Triangular or Geometric
5:      improvement ← false
6:      while All positions in baseSol is not visited do
7:          newSol ← improveByShiftingJobToLeft(baseSol, positions)
8:          if getCosts(newSol < getCosts(baseSol) then
9:              baseSol ← copySolution(newSol)
10:             improvement ← true
11:         end if
12:     end while
13:     if improvement == true then
14:         return localSearch(baseSol)                      ▷ Recursive call
15:     else
16:         return baseSol                                   ▷ Return the best known solution
17:     end if
18: end procedure
```

base solution. The completion time for the configuration of jobs obtained at the end of this process is thus always less than the RNEH makespan. However, if we would stop our algorithm at this point, we would have a high chance of ending in the vicinity of a local minimum that can be still far away from the optimal solution. To avoid this problem, we restart the entire process, compute a new base solution, and improve it with the local search procedure as long as the total running time is less than a certain threshold. In our experiments, this threshold has been defined as follows:

$$maxTime = n * m * t \tag{2.1}$$

where n is the number of jobs, m is the number of machines, and t is a time factor (initially set to 0.010 seconds). In the multi-start process we keep track of the best solution seen so far and we update it whenever necessary. By starting our search at different locations in the solution space we also increase considerably the chances of finding configurations that are close to the actual optimum. The main steps of our algorithm are summarized next (Figure 2.4):

1. Given a PFSP instance, construct the corresponding data model.
2. Choose a probability distribution (e.g. triangular or geometric) for adding random behavior to the algorithm.
3. Use the Randomized NEH (RNEH) algorithm to solve it. Assign the RNEH solution to the variable holding the best solution seen so far.
4. Check if the maximum time is met.
5. Start an iterative process to generate new solutions applying a local search to the base solution computed in step 3 or the improved solution as long as it has been updated.
6. Compare the configuration obtained in step 5 with the best solution seen so far, and update the latter if necessary.
7. Go back to step 4 and restart the process if time permits.

To increase the probability of getting a good solution we run each instance with 15 different seeds. The results are presented in the Section 2.f.

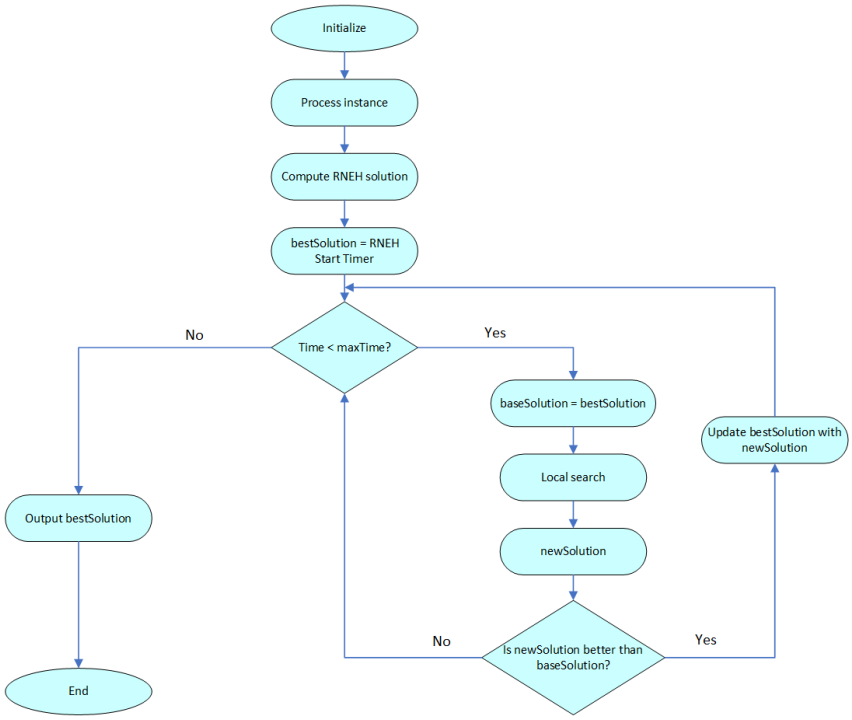


Figure 2.4: *Flowchart diagram of the algorithm.*

2.e Computational experiment

This section presents a set of extensive computational experiments carried out to test the RNEH algorithm for PFSP. Firstly, the section introduces the instances that will be used to test our approach. Secondly, the algorithm parameters are discussed. Finally, the computational results are provided, they will be fully analysed in the next section. The algorithm has been implemented as a Java application. A standard personal computer with an Intel Core i7 CPU at 1.8 GHz and 16 GB RAM has been employed to perform all the experiments.

2.e.1 Benchmark instances

As a benchmarks for the test, Taillard's benchmark suite by Taillard (1993) and Watson's benchmark suite by Watson et al. (2002) which were originally proposed by the assessment are selected. The Taillard's benchmark suite consists of 11 sets of 10 instances each, ranging from 20 to 200 jobs to be completed on 5 to 20 machines. The Watson's benchmark suite consists of 6 sets of 100 instances each, considering 50 jobs with 20 machines for the PFSP.

We tested the first 3 benchmark instances of each set of the Taillard's benchmark suite, resulting in 33 tests. The first 5 instances were chosen for the Watson's benchmark suite, in this case, resulting in 30 tests.

2.e.2 Parameters setting

One of the advantages of the RNEH algorithm is that it does not require a complex fine-tuning process. In fact, after some quick trial-and-error experiments, the following values were set for each parameter:

- The randomized selection of the algorithm can be done by using a triangular distribution or by using a geometric probability distribution with parameter $\beta \in (0.05, 0.25)$.
- For each instance, the algorithm was run 15 times, each time employing a different seed for the pseudo-random number generator.
- The time factor for the threshold explained in the Section 2.d, initially set to 0.010 seconds.

2.e.3 Computational results

The results of our experiments comparing the different approaches are summarized in Tables 2.1, 2.2, 2.3 and 2.4. It is structured as follows. Taillard's instances are show in Table 2.1 (triangular distribution) and Table 2.2 (geometric distribution). Watson's instances are show in Table 2.3 (triangular distribution) and Table 2.4 (geometric distribution).

In Taillard's instances the first 6 columns offer general information: the order of the instance, the number of jobs, the number of machines, the makespan of the BKS, the makespan of the RNEH solution, and the gap between the two. In Watson's instances the first 5 columns offer general information: the problem of the instance, the order of the instance, the makespan of the BKS, the makespan of the RNEH solution, and the gap between the two. The last 6 columns contain data describing the performance of our algorithm. Note that we run each instance with 15 different seeds, so we provide in the table an average cost (AvCost), an average time (AvTime) and an average gap with respect to the cost of the BKS for these runs. The next two columns give information about the makespan of the best solution we find (MinCost),and finally, its associated running time (MinTime).

Table 2.1: *Results for Taillard's instances using a triangular distribution.*

Taillard's Instance	# Jobs	# Machines	BKS	RNEH Solution	Gap BKS-RNEH	AvCost	AvTime	AvGap	MinCost	MinTime
1	20	5	1278	1286	0,62%	1281	0,1476	0,41%	1278	0,0002
2	20	5	1359	1365	0,44%	1359	0,0618	0,44%	1359	0,0012
3	20	5	1081	1159	6,73%	1099	0,2377	5,19%	1081	0,0001
4	20	10	1583	1680	5,77%	1611	0,5596	4,12%	1583	0,0038
5	20	10	1682	1786	5,82%	1693	0,5444	5,21%	1682	0,0400
6	20	10	1514	1557	2,76%	1531	0,4678	1,69%	1514	0,0185
7	20	20	2318	2410	3,82%	2342	1,3894	2,83%	2318	0,0021
8	20	20	2112	2150	1,77%	2132	0,9313	0,86%	2112	0,0001
9	20	20	2367	2429	2,55%	2379	0,9113	2,07%	2367	0,0673
10	50	5	2724	2733	0,33%	2728	0,1611	0,20%	2724	0,0002
11	50	5	2724	2733	0,33%	2728	0,1611	0,20%	2724	0,0002
12	50	5	2621	2643	0,83%	2626	0,5350	0,63%	2621	0,0004
13	50	10	3085	3168	2,62%	3115	1,9016	1,68%	3085	0,0002
14	50	10	3085	3168	2,62%	3115	1,9016	1,68%	3085	0,0002
15	50	10	2915	3004	2,96%	2955	0,7181	1,65%	2915	0,0012
16	50	20	3990	4082	2,25%	4014	2,7162	1,67%	3990	0,0059
17	50	20	3861	3921	1,53%	3881	4,5687	1,02%	3861	0,0007
18	50	20	3789	3927	3,51%	3845	3,5769	2,08%	3789	0,0007
19	100	5	5493	5565	1,29%	5500	0,4195	1,16%	5493	0,0007
20	100	5	5268	5349	1,51%	5283	0,6353	1,23%	5268	0,0008
21	100	5	5179	5237	1,11%	5195	1,0024	0,81%	5179	0,0008
22	100	10	5818	5911	1,57%	5845	2,9203	1,12%	5818	0,0018
23	100	10	5382	5465	1,52%	5420	4,0884	0,82%	5382	0,0009
24	100	10	5696	5837	2,42%	5737	4,1650	1,71%	5696	0,0009
25	100	20	6526	6670	2,16%	6558	8,4625	1,69%	6526	0,0040
26	100	20	6419	6565	2,22%	6497	10,5615	1,04%	6419	0,7832
27	100	20	6489	6650	2,42%	6559	10,1784	1,37%	6489	0,0013
28	200	10	10892	10972	0,73%	10930	1,7330	0,38%	10892	0,0036
29	200	10	10600	10708	1,01%	10648	4,7902	0,56%	10600	0,0036
30	200	10	11017	11081	0,58%	11035	3,6162	0,42%	11017	0,0036
31	200	20	11017	11081	0,58%	11035	3,6162	0,42%	11017	0,0036
32	200	20	11625	11785	1,36%	11684	9,7925	0,86%	11625	0,0052
33	200	20	11765	11907	1,19%	11802	9,2064	0,89%	11765	0,0053
Averages					2,09%		2,9297	1,46%		0,0292

Table 2.2: *Results for Taillard's instances using a geometric distribution.*

Taillard's Instance	# Jobs	# Machines	BKS	RNEH Solution	Gap BKS-RNEH	AvCost	AvTime	AvGap	MinCost	MinTime
1	20	5	1278	1286	0,62%	1280	0,0431	0,45%	1278	0,0001
2	20	5	1359	1365	0,44%	1359	0,0994	0,44%	1359	0,0031
3	20	5	1085	1159	6,38%	1106	0,1955	4,60%	1085	0,0001
4	20	10	1597	1680	4,94%	1615	0,8173	3,89%	1597	0,0016
5	20	10	1682	1786	5,82%	1696	0,6013	5,01%	1682	0,0000
6	20	10	1508	1557	3,15%	1532	0,3881	1,62%	1508	0,0060
7	20	20	2322	2410	3,65%	2349	1,1652	2,52%	2322	0,0110
8	20	20	2117	2150	1,53%	2130	0,7029	0,91%	2117	0,0001
9	20	20	2359	2429	2,88%	2376	0,7796	2,20%	2359	0,0001
10	50	5	2724	2733	0,33%	2730	0,0016	0,12%	2724	0,0002
11	50	5	2836	2882	1,60%	2840	0,5610	1,47%	2836	0,0002
12	50	5	2621	2643	0,83%	2627	0,3925	0,62%	2621	0,0002
13	50	10	3096	3168	2,27%	3121	1,4373	1,48%	3096	0,0002
14	50	10	3096	3168	2,27%	3121	1,4373	1,48%	3096	0,0002
15	50	10	2922	3042	3,94%	2966	1,7552	2,49%	2922	0,0005
16	50	20	2922	3042	3,94%	2966	1,7552	2,49%	2922	0,0005
17	50	20	3858	3921	1,61%	3894	1,9009	0,68%	3858	0,0003
18	50	20	3814	3927	2,88%	3861	2,0608	1,68%	3814	0,0003
19	100	5	5495	5546	0,92%	5510	0,0032	0,64%	5495	0,0007
20	100	5	5268	5351	1,55%	5278	0,9870	1,36%	5268	0,0008
21	100	5	5175	5245	1,33%	5194	0,5826	0,98%	5175	0,0074
22	100	10	5795	5921	2,13%	5841	0,2074	1,35%	5795	0,0009
23	100	10	5384	5486	1,86%	5422	1,1572	1,16%	5384	0,0009
24	100	10	5711	5837	2,16%	5755	2,5777	1,40%	5711	0,0009
25	100	20	6497	6642	2,18%	6543	4,6649	1,50%	6497	0,0013
26	100	20	6420	6565	2,21%	6477	6,0691	1,34%	6420	0,0013
27	100	20	6537	6680	2,14%	6577	1,4869	1,54%	6537	0,0013
28	200	10	10918	10992	0,67%	10943	0,0396	0,45%	10918	0,0037
29	200	10	10648	10795	1,36%	10675	3,0248	1,11%	10648	0,0036
30	200	10	11025	11092	0,60%	11041	0,4541	0,46%	11025	0,0037
31	200	20	11559	11736	1,51%	11607	2,5493	1,10%	11559	0,0052
32	200	20	11620	11810	1,61%	11672	6,5486	1,16%	11620	0,0053
33	200	20	11723	11915	1,61%	11807	4,8459	0,91%	11723	0,0052
Averages					2,21%		1,5543	1,53%		0,0020

Table 2.3: *Results for Watson's instances using a triangular distribution (50 jobs and 20 machines).*

Watson's Problem	Watson's Instance	BKS	RNEH Solution	Gap BKS-RNEH	AvCost	AvTime	AvGap	MinCost	MinTime
0	**1**	153	157	2.55%	154.3	1.2532	1.74%	153	0.0003
0	**2**	5585	5595	0.18%	5589.5	2.2699	0.10%	5585	0.0007
0	**3**	2641	2651	0.38%	2644.8	1.6221	0.23%	2641	0.0003
0	**4**	510	524	2.67%	513.1	4.3560	2.09%	510	0.0006
0	**5**	5218	5230	0.23%	5221.9	2.9768	0.15%	5218	0.0003
1	**1**	5066	5074	0.16%	5070.2	1.0491	0.07%	5066	0.0006
1	**2**	3067	3076	0.29%	3070.8	2.8524	0.17%	3067	0.0010
1	**3**	1152	1158	0.52%	1154.7	1.7603	0.29%	1152	0.0003
1	**4**	5373	5386	0.24%	5376.0	2.7349	0.19%	5373	0.0942
1	**5**	3519	3529	0.28%	3523.0	3.3357	0.17%	3519	0.0122
2	**1**	3453	3462	0.26%	3456.6	2.1536	0.16%	3453	0.0003
2	**2**	1737	1743	0.34%	1738.8	2.7502	0.24%	1737	0.1451
2	**3**	5585	5595	0.18%	5589.5	2.2699	0.10%	5585	0.0007
2	**4**	3795	3798	0.08%	3797.1	0.7157	0.02%	3795	0.0017
2	**5**	2069	2074	0.24%	2071.2	1.4295	0.14%	2069	0.0003
3	**1**	4534	4541	0.15%	4536.5	1.5038	0.10%	4534	0.0003
3	**2**	2988	2998	0.33%	2992.3	2.0307	0.19%	2988	0.0003
3	**3**	1311	1314	0.23%	1312.1	1.0617	0.15%	1311	0.0004
3	**4**	4745	4749	0.08%	4746.1	2.1776	0.06%	4745	0.0003
3	**5**	3289	3295	0.18%	3290.9	2.4461	0.12%	3289	0.0007
4	**1**	5279	5279	0.00%	5279.0	0.0008	0.00%	5279	0.0003
4	**2**	3876	3876	0.00%	3876.0	0.0020	0.00%	3876	0.0003
4	**3**	2506	2512	0.24%	2507.3	2.3948	0.19%	2506	0.0003
4	**4**	5394	5402	0.15%	5395.6	3.3676	0.12%	5394	0.0991
4	**5**	4092	4093	0.02%	4092.3	0.1202	0.02%	4092	0.0020
5	**1**	2645	2647	0.08%	2645.7	1.2367	0.05%	2645	0.0007
5	**2**	4969	4973	0.08%	4969.8	2.4441	0.06%	4969	0.0009
5	**3**	3821	3821	0.00%	3821.0	0.0007	0.00%	3821	0.0003
5	**4**	2651	2656	0.19%	2652.2	2.8687	0.14%	2651	0.0004
5	**5**	4994	4994	0.00%	4994.0	0.0005	0.00%	4994	0.0003
Averages				0.34%		1.8395	0.24%		0.0122

Table 2.4: *Results for Watson's instances using a geometric distribution (50 jobs and 20 machines).*

Watson's Problem	Watson's Instance	BKS	RNEH Solution	Gap BKS-RNEH	AvCost	AvTime	AvGap	MinCost	MinTime
0	**1**	152	158	3.80%	154.9	0.0015	1.94%	152	0.0003
0	**2**	4794	4805	0.23%	4798.8	2.1706	0.13%	4794	0.0003
0	**3**	2642	2653	0.41%	2645.6	1.7564	0.28%	2642	0.0005
0	**4**	510	522	2.30%	514.6	1.9415	1.42%	510	0.0004
0	**5**	5219	5229	0.19%	5222.3	1.5982	0.13%	5219	0.0004
1	**1**	5069	5074	0.10%	5070.9	1.7358	0.06%	5069	0.0004
1	**2**	3067	3076	0.29%	3071.9	1.6433	0.13%	3067	0.0004
1	**3**	1153	1158	0.43%	1155.9	0.4983	0.18%	1153	0.0004
1	**4**	5371	5386	0.28%	5377.3	1.5340	0.16%	5371	0.0004
1	**5**	3522	3529	0.20%	3524.4	1.1758	0.13%	3522	0.0004
2	**1**	3455	3462	0.20%	3458.7	2.7132	0.10%	3455	0.0004
2	**2**	1737	1743	0.34%	1739.9	1.5772	0.18%	1737	0.0004
2	**3**	5586	5595	0.16%	5592.3	1.2635	0.05%	5586	0.0004
2	**4**	3795	3798	0.08%	3797.5	0.5518	0.01%	3795	0.0005
2	**5**	2071	2074	0.14%	2072.5	0.6189	0.07%	2071	0.0004
3	**1**	4538	4541	0.07%	4539.9	0.5003	0.02%	4538	0.0004
3	**2**	2990	2998	0.27%	2994.8	2.0744	0.11%	2990	0.0004
3	**3**	1311	1314	0.23%	1312.5	0.5398	0.11%	1311	0.0004
3	**4**	4747	4749	0.04%	4747.8	0.0007	0.03%	4747	0.0004
3	**5**	5279	5279	0.00%	5279.0	0.0004	0.00%	5279	0.0004
4	**1**	3289	3295	0.18%	3291.7	1.3116	0.10%	3289	0.0004
4	**2**	3876	3876	0.00%	3876.0	0.0011	0.00%	3876	0.0004
4	**3**	2507	2512	0.20%	2509.3	0.9464	0.11%	2507	0.0004
4	**4**	5393	5402	0.17%	5396.3	3.1764	0.11%	5393	0.0004
4	**5**	4092	4093	0.02%	4092.5	0.5649	0.01%	4092	0.0008
5	**1**	2646	2647	0.04%	2646.3	0.1664	0.03%	2646	0.0004
5	**2**	4969	4973	0.08%	4972.3	0.3010	0.01%	4969	0.0004
5	**3**	3821	3821	0.00%	3821.0	0.0005	0.00%	3821	0.0003
5	**4**	2652	2656	0.15%	2653.6	0.5103	0.09%	2652	0.0003
5	**5**	4994	4994	0.00%	4994.0	0.0004	0.00%	4994	0.0003
Averages				0.35%		1.0292	0.19%		0.0004

2.f Analysis of results

We can see that for the triangular distribution in Table 2.1 we always obtain an improvement with respect to the solution obtained by means of the RNEH heuristic. In particular, we obtain a gap between 0.33% and 6.73% with an average of 2.09% for all the dataset. Furthermore, we see how the execution times are very short, in the order of seconds, and even reaching the order of hundredths of a second for the fastest instances. Because of parallelization, this allows the executions of many instances at once in a short time, which explains the gap obtained.

On the other hand, we have repeated the experiment for the geometric distribution in Table 2.2 and we see how the results are slightly better, because of the larger gaps and shorter execution times. It would be interesting to study the behavior of the experiment varying the parameter β of the geometric distribution, which we have kept at $\beta = 0.2$.

In Table 2.3 can be observed that the reduction in the average gap with respect the best-known solution, 0.34% is not as significant as shown in Taillard's instances. The same applies for Table 2.4 with a average gap of 0.35%. Unlike Taillard's instances, in Watson's instances the algorithm not always is able to provide a solution better than the RNEH solution.

However, the average times of Watson's instances, both for triangular and geometric distributions, are shorter than the Taillard's instances.

2.g Conclusion

In this work we have given an extensive and comprehensive review of RNEH heuristic for the permutation flowshop scheduling problem with the objective of minimising the total makespan. In particular, starting from RNEH heuristics and using the Iterated Local Search framework, we have developed an algorithm that, by means of biased probability distributions, obtains competitive solutions compared to those obtained only with the heuristic. We have done a computational experiment to test its efficiency and we have checked how we always get some kind of improvement, and the execution times are reasonably short.

The advantage of this is that the required fine-tuning is tiny because there are almost no parameters to be set, and conceptually it is easy to understand and implement. In addition, it produces many different solutions that allow the use of parallelization techniques.

2.h Future work

In the future we intend to detect stagnation situations in the local search in order to reduce its execution time without significantly reducing the quality of the solutions. Furthermore, this work studies the makespan, since this is commonly used in other researches, an additional study of the total tardiness is interesting. It seems a simple adaptation of this method. Therefore, an interesting line of research would be to compare methods that minimizing the makespan against the ones that minimizing the total tardiness.

Finally, another interesting future work would be the parallelization of the algorithm, since it could split each run with different seed in different threads CPU or GPU. This will allow us to execute multiple instances of the algorithm at the same time, each with a different seed, thus each of these threads will start to search in a different region of the solution space (by using different seeds).

2 Bibliography

T. Aldowaisan and A. Allahvedi. New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research*, 30(8):1219–1231, 2003.

R. Chandrasekharan and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2): 426–438, 2004.

C.L. Chen, V.S. Vempati, and N. Aljaber. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80(2):389–396, 1995.

J.M. Framinan and R. Leisten. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *OMEGA*, 31:311–317, 2003.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. G. H. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, pages 287–326, 1979.

Angel Juan, Helena Lourenço, Manuel Mateo, Rachel Luo, and Quim Castella. Using iterated local search for solving the flow-shop problem: parametrization, randomization and parallelization issues. *International Transactions in Operational Research*, 21:103–126, 01 2014.

M. Nawaz, E. E. Enscore, and I Ham. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA*, 11:91–95, 1983.

L. Osman and C. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA*, 17:551–557, 1989.

S. S. Reddi and C. V. Ramamoorthy. On flowshop sequencing problem with nowaitin process. *Operational Research Quarterly*, pages 323–331, 1972.

C.R Reeves. Improving the efficiency of tabu search for machine scheduling problems. *Journal of the Operational Research Society*, 22(1):375–382, 1993.

C.R Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22 (1):5–13, 1995.

R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165:479–494, 2005.

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

S Suliman. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, pages 143–152, 2000.

E Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74, 1990.

E Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

J.P. Watson, L. Barbulescu, L.D. Whitley, and A.E. Howe. Contrasting structured and random permutation flowshop scheduling problems search space topology and algorithm performance. *ORSA Journal of Computing*, 14:98–123, 2002.

M. Widmer and A. Hert. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193, 1989.