A HEURISTIC FOR THE JOB SHOP SCHEDULING PROBLEM

Hugo Zupan, Niko Herakovič
Faculty of Mechanical Engineering, University of Ljubljana, Slovenia
hugo.zupan@fs.uni-lj.si, niko.herakovic@fs.uni-lj.si

Janez Žerovnik
Faculty of Mechanical Engineering, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia
janez.zerovnik@fs.uni-lj.si

Abstract Multistart local search heuristics Remove and Reinsert that is based on a simple schedule constructing heuristics is tested on several benchmark instances of the job shop scheduling problem. The heuristics provides very good near optimal solutions within reasonably short computation time. The implementation within a plant simulation software is compared to the build-in genetic algorithm.

Keywords: Digital factory, Discrete event simulation, Genetic algorithm, Job shop scheduling problem, Remove and reinsert heuristics.

1. Introduction

Optimization of assembly and handling systems and processes (AHSS) is important in terms of reducing costs, shortening lead times, delivery terms, etc., thus ensuring the competitiveness of enterprises. It has been clearly shown [25] that it can cause disturbances to reduce the overall effectiveness of equipment (OEE), which can represent up to 50% of the costs. For this reason, it is essential to optimize AHSS. Different approaches and methods have been used to optimize AHSS in order to effectively achieve the optimum, respectively nearly optimal solution [2, 9, 20].

One of the most famous optimization problems of AHSS is the Job-Shop Scheduling Problem (JSSP). Over the past few decades, a great number of studies have been made on the job-shop scheduling problem

(JSSP). JSSP can be regarded as a scheduling problem and it is one of the most challenging combinatorial optimization problems [20]. It is of both theoretical and practical interest, c.f. it is highly popular in production industry [8].

The JSSP is known to be an NP-hard optimization problem [17]. Therefore, application of metaheuristics for the JSSP is justified when looking for optimal or near optimal solutions in reasonable amount of time. This paper proposes such an algorithm, based on Remove and Reinsert algorithm (RaR) [1, 19].

The rest of the paper is organized as follows. In the next section we briefly explain the practical motivation for this research. In Section 3 the JSSP is defined, in Section 4 the metaheuristic RaR is outlined and its operation is illustrated with an example. Results of the preliminary experiments are given in Section 5. The paper ends with a summary of conclusions and ideas for future work.

2. Motivation

For conventional JSSP, it is usually assumed that all time parameters are known exactly and in deterministic values. An instance of JSSP can be described as follows: we have a set of n jobs that need to be operated on a set of m machines [22]. Each job has its own processing route; that is, jobs visit machines in different orders. Each job may need to be performed only on a fraction of m machines, not all of them. The task is to determine a processing order of all jobs on each machine that minimizes the total flow time.

Another usual assumption is that each job can be processed by at most one machine at a time and each machine can process at most one job at a time. When the process of an operation starts, it cannot be interrupted before the completion; that is, the jobs are non-preemptive. The jobs are independent; that is, there are no precedence constraints among the jobs and they can be operated in any sequence. All the jobs are available for their process at time 0. We assume that there is a buffer of unlimited size between machines for semi-finished jobs; meaning that if a job needs a machine that is occupied, the job must and can wait until the machine becomes available. There is no machine breakdown (i.e., machines are continuously available).

Besides the conventional JSSP, there is a number of variants in the literature. We mention here briefly the JSSP with setup times [15] as this was the problem on which we tested the RaR heuristic originally [27]. Setup times of machines are typically sequence dependent (or SDST), that is, the magnitude of setup strongly depends on both current and

immediately processed jobs on a given machine. For example, this may occur in a painting operation, where different initial paint colours require different levels of cleaning when being followed by other paint colours. We also assume that setup is non-anticipatory, meaning that the setup can only begin when the job and the machine are both available.

A lot of different methods and metaheuristics were used for solving the JSSP, from very simple heuristics based on priority rules to more complex methods. For example, we only mention here applications of Tabu-search (TS) [18, 21] and Simulated Annealing [3], which are among the most popular local search based metaheuristics used in combinatorial optimization. In the literature, there are also reports on applications of more advanced metaheuristics such as Genetic algorithms [10, 16], and recently popular heuristics based on Swarm intelligence including the Firefly algorithm [6], particle swarm optimization [6], Cuckoo search [13], the artificial bee colony algorithm [12, 26], ant colony optimization [6], and others.

In contrast to above metaheuristics that appear to be rather complex, in the sense that some very specific knowledge is needed for implementation and in particular for parameter tuning, our attempt was to design a conceptually simple heuristic. Therefore we started with a simple solution construction heuristic and used the same idea to define procedures for generating a new feasible solution with a perturbation of a previously given feasible solution. This naturally leads to a local search heuristic, or, more precisely, multi-start iterative improvement heuristic. As already mentioned, we initially tested the ideas on JSSP with setup times, and, as the heuristic has proven to be surprisingly competitive on JSSP with start-up times, at least on our dataset, we decided to study the same type of heuristic on the conventional JSSP, which is much more extensively studied. Consequently, there are many benchmark instances available. Below we first formally define the problem, and then introduce the RaR algorithm. In particular we define several procedures that slightly differ in the neighbourhood structure they imply on the set of feasible solutions. Finally, experimental comparison of our heuristic with genetic algorithms that are built-in as an object in Siemens Tecnomatix Plant Simulation is provided on several benchmark instances.

3. Job Shop Scheduling Problem

The Job Shop Scheduling Problem (JSSP) is formally defined as follows. Given is a set of jobs $J = \{j_1, j_2, \ldots, j_n\}$ and a set of machines $M = \{m-1, m_2, \ldots, m_n\}$. Every job is assigned a set of operations $O_i = \{O_{i1}, O_{i2}, \ldots, O_{im}\}$ with processing times $p_{ab} = p(O_{ab})$. The op-

erations are ordered, i.e., there is a binary relation A that induces a linear order on the set of operations O_i. No precedence exists between operations of different jobs. At any time, only one operation can be performed on a machine, and it may not be interrupted.

A schedule is a function $S : O \rightarrow IN \cup \{0\}$ that for each operation v defines a start time $S(v)$. A schedule S is feasible if:

$$\forall v \in O : S(v) \geq 0$$

$$\forall v, w \in O, (v, w) \in A : S(v) + p(v) \leq S(w)$$

$$\forall v, w \in O, v \neq w, M(v) = M(w) : S(v)+p(v) \leq S(w) \vee S(w)+p(w) \leq S(v)$$

The length of a schedule S is $len(S) = \max_{v \in O}(S(v)+p(v))$. The goal is to find an optimal schedule, that is a feasible schedule of minimum length, $\min(len(S))$.

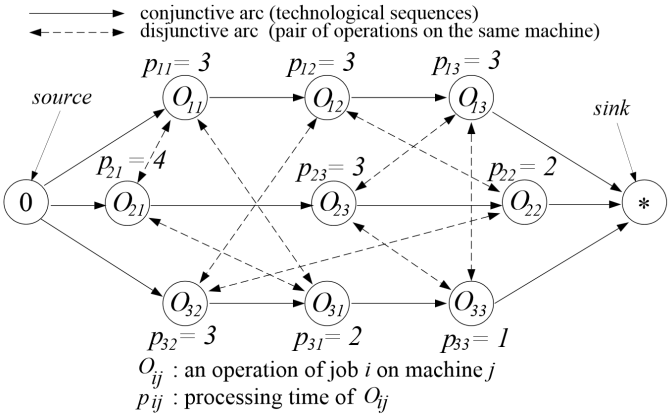

Figure 1: A disjunctive graph of a 3 × 3 problem [24].

An instance of the JSSP can be represented by means of a disjunctive graph $G = (V, C \cup D)$ where: V is a set of nodes representing operations of the jobs together with two special nodes, a source(0) and a sink(*), representing the beginning and the end of the schedule, respectively. C is a set of conjunctive arcs representing partial order of the operations. D is a set of disjunctive arcs (edges) representing pairs of operations that must be performed on the same machines. The processing time for each operation is the value attached to the corresponding node. Figure 1 shows the disjunctive graph for a simple example of JSSP with three jobs consisting of three operations each [24].

Often, the buffers (waiting queues) at machines are some standard queues, cf. First-In-First-Out (FIFO). Note that in this case, the schedule at a particular machine is fully determined by the arrival times of

the jobs. Consequently, assuming that the order of processing jobs at a machine is determined by the arrival times, it is sufficient to consider the order of jobs entering the virtual machine at the source. In this paper we study this variant of the problem.

4. Remove and Reinsert (RaR) Algorithm

Our algorithm is inspired by some applications of several similar heuristics that appeared under various names. These heuristics were successfully applied to the probabilistic traveling salesman problem (PTSP) [28], the asymmetric traveling salesman problem (ATSP) [1] and to the classical resource-constrained project scheduling problem (RCPSP) [19]. It may be rather surprising that such a simple heuristic outperforms much more complicated metaheuristics such as (in this study) a commercial implementation of a genetic algorithm. On the other hand, we think that this phenomena is not that unexpected, see [29] and the references there. In other words, as the basic idea of the heuristic is very simple and somehow natural for the particular problem, the present authors speculate that this may in fact be a reason for good results.

RaR can be regarded as a local search based on RaR neighbourhood or as a constructive heuristic. For example, on the traveling salesman problems (TSP, ATSP, PTSP) a new neighbour of a given solution is obtained by first removing a number of cities from the tour, and then reinserting them one by one into the best position that does not change the relative order of the other cities. The tour constructing heuristic on TSP (ATSP, PTSP) starts with a small subset of cities, computes their optimal permutation, and then inserts the other cities in arbitrary order. An iteration of iterative improvement consists of first removing some of the cities and then reinserting them in arbitrary order. The results on PTSP were encouraging [28], and extremely good on ATSP [1], where they were competitive with the best known heuristics of the same type. At the time, this was rather surprising as TSP is one of the most extensively studied problems in combinatorial optimization and operational research. On the resource-constrained project scheduling problem (RCPSP), the same idea with some obvious adaptations proved to be competitive with the best heuristics for the problem [19].

On the JobShop Scheduling Problem (JSSP) that is studied here, we apply the basic ideas above as follows. Below we first explain the basic neighbourhood which corresponds to a perturbation of a feasible solution into a new feasible solution. Given a parameter k and a feasible solution, k jobs are removed and reinserted, or, in this case we better say, the relative positions in the sequence of the selected k jobs may change. In

contrast to RaR on the traveling salesman problems, we do not remove all the jobs at the same time, but remove and reinsert the selected jobs one at a time. In a pseudo programming language, it could be written as

Procedure **GenerateNeighbourBasic(S0, k)** Returns(S1)
1. S1 = S0
2. Choose k jobs J_k
3. For i = 1 to k do
   a. Select a job j_i from J_k (J_k = J_k - j_i)
   b. Insert j_i into S1 on the best position
4. Return(S1)

Based on the basic neighbourhood given by procedure **GenerateNeighbourBasic**, several other neighbourhoods can be naturally defined. Here we first define a neighbourhood called large neighbourhood defined by procedure **GenerateNeighbourLarge** that can change the given solution substantially. The **GenerateNeighbourLarge** procedure first removes a subset of jobs, but, before reinserting them, it solves the subproblem to optimality. This implies that large k have to be used as otherwise the procedure would be very time consuming.

Procedure **GenerateNeighbourLarge(S0, k)** Returns(S1)
1. S1 = S0
2. Choose k jobs J_k
3. Remove jobs J_k from S1
4. Find optimal order of jobs J – J_k within S1
5. For i = 1 to k do
   a. Select a job j_i from J_k (J_k = J_k – j_i)
   b. Insert j_i into S1 on the best position
6. Return(S1)

Local search using different neighbourhoods is not a new idea. For example, it is extensively studied under name variable neighbourhood search [7, 14].

Note that **GenerateNeighbourLarge** can also be seen as procedure for generating an initial solution. Given any S0, **GenerateNeighbourLarge(S0,k)** is a heuristic that provides a good solution. (Note that there is some clear analogy to the well-known Arbitrary Insertion tour constructing heuristic for TSP.)

A version of **GenerateNeighbourLarge** that we use below allows selection of the k jobs to be perturbed outside the procedure. For this aim we define the jobs to be selected by their positions, therefore a dif-

ferent name for this variant:

Procedure **GenerateNeighbourLargePos(S0,P_k)** Returns(S1)
1. S1 = S0
2. Let J_k be k jobs at positions P_k
3. Find optimal order of jobs J – J_k within S1
4. For i = 1 to k do
   a. Select a job j_i at the next position among P_k
   b. Insert j_i into S1 on the first best position
5. Return(S1)

In our implementation of RaR, we run a multistart RaR based local search heuristic. However, as we avoid randomization after the initial solution is given, the sequence of selected neighbours is predefined. In more details, the heuristic used in the experiment is

1. Generate a Random initial solution S
2. While (there is time left) do
3. Repeat
4. S0 = S
5. Run a sequence of moves:
6. For w = 1 to n – m
7. P = positions w + m,. . . , n
8. S1 = **GenerateNeighbourLargePos(S, P)**
9. S = better between S and S1
10. Until S not better than S0

We conclude the Section with a list of remarks emphasizing some basic facts regarding our implementation of the heuristic.

1 We run a multistart of iterative improvements of the large neighbourhood **GenerateNeighbourLargePos**.

2 We speed up the implementation by avoiding randomization in the first experiments. In particular, the choices of the jobs that generate a subproblem are always a sequence of $(n-k)$ jobs starting at some position, say w. Again, this seemingly counterintuitive decision is based on the speculation that such subproblems may provide relatively good starting solution before reinsertion.

3 Removing from the set of jobs from solution is performed one at a time, thus all the jobs contribute to the cost of intermediate solutions. This decision was taken because we have observed that

completely removing many jobs may cause that the properties of the subproblems differ too much from the full problem and, consequently, even very good solutions of a subproblem may provide poor starting solution before reinsertion.

4 The testing environment is software program Siemens Tecnomatix Plant Simulation, which is used for evaluations of our scenarios and for comparison of our RaR algorithm with built-in genetic algorithms [8]. Therefore the most natural measure of time here is the number of scenarios. We also measure wall clock time, but note that this information is not very useful, as the software provides a user friendly front end with lots of graphics etc. that we do not control but tends to be very time consuming.

5. Results

For computational results the RaR algorithm was combined with discrete event simulation. It is well known [26] that using discrete event simulation or virtual factory is very effective tool for "what–if" scenarios, for every type of production system. In our case we have transformed JSSP with all the features and limitations into virtual factory. The idea is that the metaheuristic proposes initial and iteratively improved schedules of orders while the discrete event simulation performs "what–if" scenario for each proposed schedule thus providing the quality measure of the schedule. This process is repeated until the metaheuristic can no longer provide better schedule.

Recall that we assume that all queues are FIFO. Hence the algorithm optimizes only the schedule on the source (see Fig. 1).

The algorithm was tested on some well-known and one of the most used benchmark instances for the JSSP – LA01 to LA05 problems from Lawrence [11] and MT06, MT10 and MT20 problems from Fisher and Thompson [5]. We compared the RaR algorithm with the built-in "Siemens" genetic algorithm (SGA), which is already installed in the Siemens programming environment Plant Simulation [7]. The results are presented in Table 1. Note that both algorithms always started with the same initial scheduling $O_1, O_2, \ldots, O_n$.

From the results we see that RaR algorithm finds a very good solution in a relatively short time compared to SGA. The great advantage of RaR algorithm is that it is not necessary to store large amounts of data, since the algorithm works sequentially, and in almost every step takes only the best solution and while discarding the others.

We made also a test to see effect of different initial schedules on our RaR algorithm, to see how they effect on end results. The results showed

Table 1: Comparison of results between SGA and RaR.

	SGA		RaR	
Problem	best solution	time	best solution	time
LA01	705	13 s	705	3 s
LA02	758	17 s	778	3 s
LA03	679	21 s	681	2 s
LA04	660	10 s	660	2 s
LA05	593	6 s	593	2 s
MT06	59	5 s	59	1 s
MT10	1092	24 s	1092	4 s
MT20	1496	114 s	1496	9 s

that with different initial scheduling different best solutions are possible. The results were tested on benchmark LA02 and are presented in Table 2. Note that in 7 runs, the quality of the best solution by RaR has improved. In one of the runs it was even better than solution given by SGA.

Table 2: Effect of different initial scheduling on best solution found by RaR algorithm.

Initial scheduling	RaR best solution
O1, O2, O3, O4, O5, O6, O7, O8, O9, O10	778
O9, O8, O7, O10, O5, O3, O6, O1, O4, O2	758
O5, O8, O6, O3, O1, O7, O2, O4, O10, O9	766
O5, O8, O6, O3, O1, O9, O2, O4, O7, O10	754
O5, O9, O7, O10, O8, O3, O6, O1, O4, O2	758
O8, O6, O5, O9, O10, O7, O2, O1, O3, O4	778
O10, O9, O8, O7, O6, O5, O4, O3, O2, O1	778

As mentioned in Section 2, in real production there are setup times of machines which are sequence dependent - this problem is called Sequence Depended Setup Times JSSP (SDST JSSP) [15]. RaR algorithm was also tested for two SDST JSSP; with 10 jobs and with 100 jobs. In both cases we compared the algorithm with the genetic algorithm. The results are as presented in Table 3.

Characteristics of the genetic algorithm (GA):

- instance with 10 jobs: 50 generations: size of generation: 100;
- instance with 100 jobs: 500 generations; size of generation: 100.

According to the results of the first tests that have been carried out (reported above and some those that are not described in here), we can

Table 3: The results of GA and RaR algorithm on SDST JSSP.

Algorithm	Total time ...	for finding best solution	Quality of the best solution
Instance with 10 orders			
GA	1 min	33 sec	2468
RaR	20 sec	8 sec	2468
Instance with 100 orders			
GA	4h 5 min 22 sec	4h 5 min 22 sec	14385
RaR	29 min 30 sec	22 min 41 sec	13768

say that RaR algorithm works very well and in quick time gives good solutions.

6. Conclusion

This paper proposes Remove and Reinsert (RaR) heuristic for the job-shop scheduling problem. Preliminary results show that it provides very good solutions thus nearly minimizing the expected average flow time within a reasonable amount of calculation time.

For executions of "what–if" scenarios of the initial schedules, the discrete event simulation software Technomatix Plant Simulation was used. A comparison of RaR algorithm with the Genetic Algorithm which is built-in module in Technomatix Plant Simulation software showed that RaR algorithm finds good solution in shorter time compared to Genetic Algorithm. We have to mention that we did not change any parameters of the built-in GA. Maybe some better tuning of parameters for GA would improve its performance, but this was not possible as the software we use does not allow such user intervention.

Motivated by the promising results outlined here, we have also executed "what–if" scenarios for different priority rules inside the production processes. The results showed that by changing priority rules of processing the orders on the machines, we get even shorter flow time of all orders. Details will be given in the full paper.

In our future work, further experiments will be conducted to shorten the calculation time of getting the optimal solutions from the RaR algorithm.

Finally, as the RaR heuristic performs remarkably well on the JSSP, it may be worth considering the same idea on the other versions of the JSSP and some other NP-hard problems.

Acknowledgment: This research was supported in part by the Slovenian Research Agency.

References

[1] J. Brest and J. Žerovnik. An approximation algorithm for the asymmetric traveling salesman problem. *Ricerca Operativa*, 28:59–67, 1999.

[2] M. Debevec, M. Šimic, and N. Herakovič. Virtual factory as an advanced approach for production process optimization. *International Journal of Simulation Modelling*, 13(1):66–78, 2014.

[3] A. Elmi, M. Solimanpur, S. Topaloglu, and A. Elmi. A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & Industrial Engineering*, 61(1):171–178, 2011.

[4] H. Eskandari, M. A. Rahaee, M. Memarpour, E. Hasannayebi, and S. A. Malek. Evaluation of different berthing scenarios in Shahid Rajaee container terminal using discrete-event simulation. *Proceedings of the Simulation Conference (WSC)*, 2013.

[5] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson (Eds.) *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[6] I. Fister Jr., X. S. Yang, I. Fister, J. Brest, and D. Fister. A Brief Review of Nature-Inspired Algorithms for Optimization. *Elektrotehniški vestnik*, 80(3):116–122, 2013.

[7] P. Hansen and N. Mladenovi. Variable Neighborhood Search Methods. In *Encyclopedia of Optimization*, pages 3975–3989, 2009.

[8] X. Hao, L. Lin, M. Gen, and K. Ohno. Effective Estimation of Distribution Algorithm for Stochastic Job Shop Scheduling Problem. *Procedia Computer Science*, 20:102–107, 2013.

[9] N. Herakovič, P. Metlikovič, and M. Debevec. Motivational lean game to support decision between push and pull production strategy. *International Journal of Simulation Modelling*, 13(4):433–446, 2014.

[10] X. W. Huang, X. Y. Zhao, and X. L. Ma. An improved genetic algorithm for jobshop scheduling problem with process sequence flexibility. *International Journal of Simulation Modelling*, 13(4):510–522, 2014.

[11] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[12] J. Q. Li, Q. K. Pan, and M. F. Tasgetiren. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling*, 38(3):1111–1132, 2014

[13] M. K. Marichelvam, T. Prabaharan, and X. S. Yang. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing*, 19:93–101, 2014.

[14] N. Mladenović, P. Hansen, and J. Brimberg. Sequential clustering with radius and split criteria. *Central European Journal of Operations Research*, 21(Supplement-1):95–115, 2013.

[15] B. Naderi, S. M. T. Fatemi Ghomi, and M. Aminnayeri. A high performing metaheuristic for job shop scheduling with sequence-dependent setup times. *Applied Soft Computing*, 10:703–710, 2010.

[16] N. M. Nidhiry and R. Saravanan. Scheduling optimization of a flexible manufacturing system using a modified NSGA-II algorithm. *Advances in Production Engineering & Management*, 9(3):139–151, 2014.

[17] B. Ombuki and M. Ventresca. Local search genetic algorithms for the job shop scheduling problem. *Applied Intelligence*, 21:99–109, 2004.

[18] B. Peng, Z. Lü, and T. C. E. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53(1):154–164, 2015.

[19] I. Pesek, A. Schaerf, and J. Žerovnik. Hybrid local search techniques for the resource-constrained project scheduling problem. *Lecture Notes in Computer Science*, 4771:57–68, 2007.

[20] M. L. Pindeo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York Dordrecht, Heidelberg, London, 2012.

[21] A. Ponsich and C. A. Coello Coleo. A hybrid Differential EvolutionTabu Search algorithm for The solution of Job-Shop Scheduling Problems. *Applied Soft Computing*, 13(1):462–474, 2013.

[22] D. R. Sule. *Industrial Scheduling*. PWS Publishing Company, 1997.

[23] Tecnomatix Plant Simulation, Siemens PLM Software. `http://www.emplant.de/english/fact` [accessed on 29/02/2016].

[24] T. Yamada and R. Nakano. Job-shop scheduling. In: A. M. S. Zalzala and P. J. Fleming (Eds.) *Genetic algorithms in engineering systems*. IEE Control Engineering Series 55, pages 134–160, 1997.

[25] T. Ylipää. Correction, prevention and elimination of production disturbances. PROPER project description, Department of Product and Production Development (PPD), Chalmers University of Technology, Gothenhurg, 2002.

[26] R. Zhang, S. Song, and C. Wu. A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*, 141(1):167–178, 2013.

[27] H. Zupan, N. Herakovič, and J. Žerovnik. A hybrid metaheuristic for job-shop scheduling with machine and sequence-dependent setup times. *Proceedings of the International Symposium on Operational Research in Slovenia (SOR)*, pages 129–134, 2015.

[28] J. Žerovnik. A Heuristics for the Probabilistic Traveling Salesman Problem. *Proceedings of the International Symposium on Operational Research in Slovenia (SOR)*, pages 165–172, 1995.

[29] J. Žerovnik. Heuristics for NP-hard optimization problems : simpler is better!? *Pre-conference proceedings of the 11th International Conference on Logistics & Sustainable Transport*, 2014.