A Simulated Annealing Algorithm for Scheduling Problems

Article *in* Journal of Applied Mathematics and Physics · October 2019

DOI: 10.4236/jamp.2019.simann

2 authors, including:

Crescenzio Gallo
Università degli studi di Foggia
118 PUBLICATIONS **233** CITATIONS

SEE PROFILE

Open Journal of Applied Mathematics and Physics, 2019, *,**
Published Online **** 2019 in SciRes. http://www.scirp.org/journal/jamp
http://dx.doi.org/10.4236/****.2019.*****

A Simulated Annealing Algorithm for Scheduling Problems

Crescenzio Gallo[1], Vito Capozzi[2]

[1]Department of Clinical and Experimental Medicine, Foggia, Italy
Email: crescenzio.gallo@unifg.it

[2]Department of Clinical and Experimental Medicine, Foggia, Italy
Email: vito.capozzi@unifg.it

Received Sep. 17, 2019

Abstract

An algorithm using the heuristic technique of *Simulated Annealing* to solve a scheduling problem is presented, focusing on the scheduling issues. The approximated method is examined together with its key parameters (freezing, tempering, cooling, number of contours to be explored), and the choices made in identifying these parameters are illustrated to generate a good algorithm that efficiently solves the scheduling problem.

Keywords

Scheduling; Simulated Annealing; Discrete optimization algorithm

1. Introduction

With reference to the management of the Central Processing Unit, it is fundamental to have an algorithm that specifies the sequence according to which the processing service is assigned to a specific job. This algorithm is called scheduler and implements some computing methodologies which can be very time-consuming [8, 17]. For example, it may be more expensive to run the "scheduler" job than the entire sequence to process.

It is therefore important to indicate a "cost" function with which the weight of each possible sequence can be measured: based on this, it is necessary to find a sequence order that allows the minimization of the cost function. It is therefore clear that scheduling issues [2, 20] are intimately linked to decisions about what needs to be done and how it needs to be done.

Given a problem, we will say that we can extract the sequence requests associated with this problem

when the following assumptions are met:

1. The jobs to be executed are fully known, and may not be announced to the scheduling process simultaneously but sequentially over time;
2. The resources that can be used in the execution of the work are fully specified;
3. The sequence of core tasks required to perform each of the jobs is known.

The scheduling problem [5, 6, 7, 16, 19] can be solved in several ways [13, 29]. A first way is to enumerate all the possible sequences, calculating the value of the target function, to choose the sequence which corresponds to the lowest value of the target function. For example if we have 10 jobs on 4 machines, we have to calculate the value of the target function for each of the 10! possible permutations. This method then becomes not available for medium and large scheduling problems.

Other classic methods that solve the problem exactly consist of algorithms for branch-and-bound, branch-and-bound truncated, dynamic programming. In the following we will examine the solution approach known as "Simulated Annealing" [22, 27], an approach which derives from the controlled lowering of temperature in physical systems.

2. Simulated Annealing

Simulated annealing [10, 11, 30] is an approach based on statistical mechanics concepts, and it is motivated by an analogy with the behaviour of physical systems during the cooling process. This is an heuristic method for global optimization which requires no particular assumption on objective function (e.g. convexity).

This analogy is best illustrated in terms of the physics of single crystal formation starting from a melting phase. The temperature of this "molten crystal" is then very slowly reduced until the crystal structure is formed. If cooling is carried out very quickly, undesirable phenomena occur as dislocations and polycrystalline phases. In particular, very large irregularities are enclosed in the structure of the crystal and the level of potential energy incorporated is much higher than that which would exist in a perfect single crystal.

This "quick cooling" process can be seen as similar to local optimization (Table 1). The states of the physical system correspond to the solutions of a combinatorial optimization problem; the energy of a state corresponds to the cost of a solution and the minimum energy, or fundamental state, corresponds to an optimal solution.

Physical System	Optimization Problem
State	Feasible solution
Energy	Cost
Fundamental state	Optimal solution
Quick cooling	Local search
Accurate annealing	Simulated annealing

Table 1: The analogy between the physical system and the discrete optimization problem.

When the temperature is, theoretically, at absolute zero Kelvin no state transition can lead to a higher energy state. Therefore, as in local optimization, upward movements are forbidden and the consequences of this may be undesirable.

When crystals begin to form, the risk of undesirable local states is avoided by lowering the temperature very slowly, with a process called accurate annealing. In this process the temperature drops very slowly through a series of levels, each maintained long enough to allow the search for equilibrium — at that temperature — for the crystal. As long as the temperature is larger than zero Kelvin, upward movements are always possible. By preventing the temperature from deviating from that compatible with the energy

level of the current equilibrium, we can hope to avoid local optima until we are relatively close to the basic state.

Simulated annealing is the algorithmic counterpart of this physical annealing process. The name simulated annealing refers to the technique of simulation of the physical annealing process in conjunction with an annealing schedule of temperature decrease.

Simulated annealing can be seen as an extension of the local optimization technique [3, 4, 18, 31], where the initial solution is repeatedly improved by small local perturbations until none of these perturbations improves the solution. Simulated annealing randomizes this procedure in such a way as to occasionally allow upward movements, i.e. perturbations that worsen the solution, and this in an attempt to reduce the probability of blocking in a locally optimal but overall poor solution.

Simulated Annealing was proposed by [26] and used in the field of statistical physics to determine the properties of metal alloys at a given temperature. The possibility of solving combinatorial optimization problems have been demonstrated independently by [23] and [9].

Further applications have been carried out with great success, as shown in Refs. [1, 14, 23, 24, 33].

A simple scheme of simulated annealing [15] is as follows:

1. Consider an initial solution S.
2. Until you have a freeze
   (a) For $w := 0$ up to the number of neighbors considered
      i. Let S' an unexamined neighbor of S
      ii. If Cost(S')<Cost(S) then $S := S'$
      else set $S := S'$ with a certain probability
   (b) Cool the temperature
3. Return S

From what reported above it is clear that the fundamental parameters for the simulated annealing phase are:

Freezing. This consists of establishing the criteria for stopping the algorithm.

Temperature. This is a value on which the probability of accepting upward movements depends. At each iteration this temperature is reduced by a constant rate called *cooling*.

Number of Neighbors to Explore. Each iteration considers a (fixed) number of sequences close to the one considered.

Simulated annealing is widely applicable in local optimization problems. It seems to produce better solutions than other local optimization techniques, as it allows you to get out of locally excellent solutions but overall poor allowing to make upward movements with a certain probability.

3. Mathematical models

We indicate with the term *completion time* a value that depends on the particular scheduling sequence. If a job is in position i in the sequence, its completion time is the completion time of the job at position $i-1$ plus its processing time. We will also indicate with the term *total completion time* the sum of the completion times of each individual job.

The sequencing problems that we will deal with are called *non preemptive*: the job that is in the processing state cannot be interrupted for any reason.

For the solution of scheduling problems [21, 28, 32] we describe two mathematical models that formalize it. The first model formulates the problem of scheduling n jobs on m machines that minimizes the total completion time. We indicate with x_{ij}^k the variable that is equal to 1 if job j is the $k-$th job executed on the machine i,

0 otherwise. We also indicate p_{ij} the processing time of process j on the machine i; we will therefore represent with $p_j = \sum_{i=1}^{m} p_{ij}$ the processing time of process j on all the machines.

The model is therefore as follows:

$$\text{Minimize } \sum_{k=1}^{n}\sum_{j=1}^{n} p_j x_{ij}^k \ (i = 1, \dots, m) \text{ subject to:}$$

$$\sum_{k=1}^{n}\sum_{i=1}^{m} x_{ij}^k = 1 \quad (j = 1, \dots, n)$$

$$\sum_{j=1}^{n} x_{ij}^k \leq 1 \quad (k = 1, \dots, n;\ i = 1, \dots, m)$$

$$x_{ij}^k \in \{0, 1\} \quad (i = 1, \dots, m; j, k = 1, \dots, n)$$

where the first constraint binds the j-th job to run as k-th on the i-th machine for some i and j.

The second model concerns the scheduling of n jobs on m machines that minimizes the sum of single completion times. By specifying with $C_j^k = \sum_{l=1}^{k}\sum_{j=1}^{n} p_j x_{ij}^l$ the completion time of job j scheduled as k-th on machine i, the related mathematical model is therefore:

$$\text{Minimize } \sum_{k=1}^{n}\sum_{i=1}^{m} C_j^k \text{ subject to:}$$

$$\sum_{k=1}^{n}\sum_{i=1}^{m} x_{ij}^k = 1 \ \ (j = 1, \dots, n)$$

$$\sum_{j=1}^{n} x_{ij}^k \leq 1 \ \ (k = 1, \dots, n;\ i = 1, \dots, m)$$

$$x_{ij}^k \in \{0, 1\} \ \ (i = 1, \dots, m;\ j, k = 1, \dots, n)$$

4. Simulated Annealing Algorithm

The problem that we are going to solve with the simulated annealing technique can be summarized as follows: *schedule a sequence of n jobs on m machines establishing that the order of n jobs is the same on each of the m machines*.

4.1. Basic parameters

Basic parameters for the phase of simulated annealing are:

Initialization. Choice of an initial solution. In theory the choice of an initial solution has no effect on the quality of the final solution, i.e. the solution converges to the overall optimum independently of the initial solution [25]. There are, however, experimentally verified exceptions to this theory, in which it has been shown that sometimes the process of convergence to the optimal solution is more rapid if one takes as the initial

solution a solution obtained by means of a good heuristic [12]. However, the overall computational time for an approximate solution starting from a "good" initial solution is often higher than the computational time needed to obtain an approximate solution starting from any initial solution. The reason for this approach is to be found in the peculiarity of the mechanism that generates the perturbations.

Selection of a mechanism that generates small perturbations, to pass from one configuration to another one. The perturbation pattern is a crucial element to obtain good performances [9, 23]. The chosen perturbation scheme consists in considering two randomly generated numbers and exchanging places for the jobs that in the scheduling queue occupy the positions relative to the random numbers chosen.

4.2. Data structure

The (data) structure chosen for the representation of the $i-$th job (with $0 < i \leq n-1$) is as follows:

p(j), $\forall\, 0 \leq j \leq m-1$
It is a vector that stores the job's processing time on the $j-$th processor. These data are read from the input stream.

tot$_p$
This is the total processing time of the job, obtained by summing the processing time of the job on *all* processors.

d
Job due date. This data is read from the input stream and is calculated by the generator in the following way:

$$d = tot_p + (\text{input percentage of } tot_p)$$

c
This is the completion time of the job; its value depends on the particular permutation of the scheduling queue. If the job is in the $i-$th position in the scheduling queue, then its completion time is given by the value of the completion time of the job that precedes it in the scheduling queue plus its own processing time.

late
This is the late work value of the job. It is computed as follows:

- If $c - d \leq 0$ then $late = 0$
- If $0 < c - d \leq tot_p$ then $late = c - d$
- If $c - d > tot_p$ then $late = tot_p$

Other useful information for understanding the algorithm are:

Actual order
This is the scheduling order of the jobs currently considered. The order following the reading of the data from the input stream is assumed to be $0, 1, 2, \ldots, n-1$.

Best order
It is the job scheduling order that obtains the minimum from the objective function. This minimum value is stored in the variable BOF (Best Objective Function).

Parameters chosen for the simulated annealing phase are:

Freezing criterion
A *freeze* occurs when the algorithm, for MAXITERATIONS times, does not perform neither a downhill nor an uphill movement.

Temperature
The initial temperature for the simulated annealing phase is set in the $temperature$ parameter.

Cooling
At each iteration, the temperature is cooled by a parameter *cooling*.

L
Number of neighbors to visit at each iteration.

4.3. The proposed algorithm

The proposed algorithm consists of two nested cycles. Basically, it chooses two integer random numbers (i and j) in the range $1, \ldots, n$; then exchanges job i and job j in the scheduling queue; finally it calculates the value of the target function. If this value is lower than the value stored in BOF, the scheduling order is stored and the variable BOF is updated. Otherwise, if the value worsens the target function, a probability is computed on the basis of which the worsening of the target function is accepted.

The detailed algorithm is shown in Fig. 1. Its strength lies in calculating the value of the target function without having to physically exchange jobs i and j.

4.4. Application example

Let's take an example of how the algorithm works. Let's suppose we have the situation represented in Table 2, and we have chosen the random indexes 2 and 4. The resulting queue by exchanging the jobs that occupy these positions into the actual order queue is shown in Table 3.

$actual\,order$	0	1	2	3	4	5	6
	job 0	job 1	job 2	job 3	job 4	job 5	job 6
machine 0	10	40	80	30	50	15	70
machine 1	8	45	87	22	52	13	75
machine 2	12	30	85	20	50	18	48
tot_p	30	115	252	72	152	46	193
d	33	126.5	277.2	79.2	167.2	50.6	212.3
c	30	145	397	469	621	667	860
$late$	0	18.5	119.8	72	152	46	193
$best\,order$	0	1	2	3	4	5	6

Table 2: Initial job sequence

In this case we have worsened the value of the target function. In fact the $best\,order$ queue is unchanged, but it remains evident that for the completed time values:

- jobs 0 and 1 remained unchanged;
- jobs 2 and 3 have been increased by the same amount (100); this increase is precisely equal to the difference *diff* $= tot_p(4) - tot_p(2)$ of the initial scheme;
- jobs 4, 5 and 6 remained unchanged.

For late work values, we have instead:

- $late$ has not changed for jobs 0 and 1 in the new scheduling queue;

Consider an initial solution S
Consider an initial $temperature > 0$; reset the variable BOF and the $cooling$ counter
while you don't have a freeze

for $w := 0$ to L

Generate two (different) random indexes i and $j \in \{0, 1, \ldots, n-1\}$

Consider the scheduling queue S', obtained by exchanging jobs that are in the $actual\,order(i)$ and $actual\,order(j)$ position in the scheduling queue S

Compute the value of the target function s' for the schedule queue S'

if $s' \leq s$ /* downward movement */

$s := s'$ /* put $S = S'$, i.e. physically exchange jobs */

if BOF$> s'$

BOF:$= s'$

$best\,order := actual\,order$

endif

else

$p := \min\{1, e^{-(s'-s)/temperature}\}$

Let ε a random number in the interval $[0, 1] \cap \mathcal{R}$

if $\varepsilon > p$ /* upward movement */

$s := s'$ /* put $S = S'$ */

else

increase the $cooling$ counter

endif

endif

endfor

$temperature := cooling * temperature$

endwhile

Figure 1: The proposed algorithm

$actual\,order$	0	1	4	3	2	5	6
	job 0	job 1	job 4	job 3	job 2	job 5	job 6
machine 0	10	40	50	30	80	15	70
machine 1	8	45	52	22	87	13	75
machine 2	12	30	50	20	85	18	48
tot_p	30	115	152	72	252	46	193
d	33	126.5	167.2	79.2	277.2	50.6	212.3
c	30	145	297	369	621	667	860
$late$	0	18.5	129.8	72	252	46	193
$best\,order$	0	1	2	3	4	5	6

Table 3: Permuted job sequence

- for the job that occupies position 2 in the new queue, calculated the new completed time, the latter is then compared with the total processing time of job 4 to compute the new $late$ work value;
- for job 3, once the new completed time value has been calculated, it is compared with the due date value of job 3 in the previous scheme;
- for job in position 4 in the new scheduling queue, the completed time value is unchanged, so you have simply to compare it with the total job 2 processing time in the old scheduling queue to get the new late work value;
- for jobs 5 and 6, the late values are unchanged.

In general, if we intend to exchange job i with job j $(i < j)$:

- the values of late work will be unchanged for jobs in positions $\{0, 1, \ldots, i-1\}$;
- the following value is computed: *diff* $= tot_p(j) - tot_p(i)$;
- for job in position i, the new late work value is calculated by comparing the quantity: *diff* $+c(i) - d(j)$ with the due date value of process j;
- for jobs in position $\{i+1, i+2, \ldots, j-1\}$ the late work value is computed by comparing the quantity: *diff* $+c(k) - d(k)$ with the due date value of process k, where $k \in \{i+1, i+2, \ldots, j-1\}$;
- for job in position j the new late work value is computed by comparing the quantity: *diff* $+c(j) - d(i)$ with the due date value of process i;
- for jobs in position $\{j+1, j+2, \ldots, n-1\}$ the late work values are unchanged.

We can therefore calculate the value of the target function without having to build the tables for the new schedule queue, but simply by processing information that is already available without using memory quantities.

5. Computational tests

Before analyzing the computational tests, let us consider the (random) generator of the numbers that form the processing time values. For each job on each machine a random integer number is generated in the range $[0, 100] \cap \mathcal{N}$ which represents the processing time value of the job on the processor.

The seed of the random number generator is initialized to a different number depending on the system time. The due date time is instead calculated by adding the sum of the values of the processing time of the job on all the available machines, increased by a percentage given by input.

Data to be considered common to all tests are:

- increase percentage for the due date values: 10;
- number of neighbors to be explored: 10;
- number of iterations per problem: 15.

With regard to the number of neighbors to be explored, the above value was chosen because from an experimental check it was noted that for a smaller number (of neighbors) you get results close to the optimal one, while for a larger number you will keep (in the cases we tested) values that often coincide with the optimal one but, being computationally severe, they negatively affect the execution time of the approximation algorithm.

Some test cases are presented to verify the computational weight on the basis of the hypotheses made.

Guidance for understanding computational testing.
Table 4 summarises the tests performed. The indications at the top of the columns indicate:

n
: number of jobs to be scheduled;

m
: number of machines used;

OF
: optimum value of the target function;

OF=SA
: number of times that the result obtained with the approximated algorithm of simulated annealing coincides with the optimal one;

T_{min}
: minimum time (in seconds) taken by the simulated annealing algorithm to find a result that coincided with the optimum;

T_{max}
: maximum time (in seconds) taken by the simulated annealing algorithm to find a result that coincided with the optimum;

OF≠SA
: number of times the result obtained by the approximated simulated annealing algorithm differs from the optimum one;

min
: minimum value used by the simulated annealing algorithm to obtain a result that does not coincide with the optimum;

T
: time (in seconds) taken by the simulated annealing algorithm to obtain a result that does not coincide with the optimum;

$\Delta\%$
: average percentage difference.

6. Conclusions

In the present work we present an algorithm that uses the heuristic technique of *Simulated Annealing* to solve a scheduling problem. The followed solution method is of a non-deterministic type, and it is based on the probability of obtaining an optimal solution in relation to the current situation of the target function and the possible improvement due to a controlled movement in the space of the feasible solutions — linked to the concept of "temperature" of the algorithm — until it reaches the freezing point, that is an optimal acceptable one.

The choice made in our method prescinds from the initial solution and it rather takes into account the total computing time required, since it is more important to obtain an "almost" optimal feasible solution in a reasonable time rather than trying to arrive at the optimal solution in a theoretically unlimited time.

References

[1] E. H. L. Aarts and P. J. M. Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems.. Philips J. Res., 40(4):193–226, 1985.

n	m	OF	OF=SA	T_{min} (sec.)	T_{max} (sec.)	OF≠SA	min	T (sec.)	$\Delta\%$	max	T (sec.)	$\Delta\%$
10	3	1112.200	15	0.04	0.35	0	–	–	–	–	–	–
10	4	1312.600	13	0.06	0.33	2	1338.6	0.03	1.94	1353.6	0.01	3.02
10	5	2393.210	15	0.06	0.33	0	–	–	–	–	–	–
11	3	1170.000	13	0.05	0.64	2	1195.0	0.02	2.09	1195.0	0.08	2.09
11	4	1819.800	15	0.03	0.34	0	–	–	–	–	–	–
11	5	2433.900	15	0.05	0.31	0	–	–	–	–	–	–

Table 4: Computational tests

[2] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. The 8th IEEE international conference on advanced computing and communications (ADCOM 2000), 45–52, 2000.

[3] K. Akram, K. Kamal and A. Zeb. Fast simulated annealing hybridized with quenching for solving job shop scheduling problem. Applied Soft Computing, 49:510–523. Elsevier, 2016.

[4] D. C. Bissoli, W. A. S. Altoe, G. R. Mauri, and A. R. S. Amaral. A simulated annealing metaheuristic for the bi-objective flexible job shop scheduling problem. 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE), 1–6. IEEE, 2018.

[5] J. Błażewicz. Selected topics in scheduling theory. North-Holland Mathematics Studies, 132:1–59. North-Holland, 1987.

[6] J. Błażewicz, M. Dror and J. Weglarz. Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research, 51(3):283–300. Elsevier, 1991.

[7] J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. European journal of operational research, 93(1):1–33. Elsevier, 1996.

[8] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of optimization theory and applications, 45(1):41–51. Springer, 1985.

[9] B. Çaliş and S. Bulkan. A research survey: review of AI solution strategies of job shop scheduling problem. Journal of Intelligent Manufacturing, 26(5):961–973. Springer, 2015.

[10] S. Chakraborty and S. Bhowmik. An efficient approach to job shop scheduling problem using simulated annealing. International Journal of Hybrid Information Technology, 8(11):273–284, 2015.

[11] M. A. Cruz-Chávez, M. G. Martínez-Rangel, and M. H. Cruz-Rosales. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem. International Transactions in Operational Research, 24(5):1119–1137. Wiley Online Library, 2017.

[12] M. Eusuff, K. Lansey, and F. Pasha. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Engineering optimization, 38(2):129–154. Taylor & Francis, 2006.

[13] T. F. Gonzalez. Handbook of approximation algorithms and metaheuristics. Chapman and Hall/CRC, 2007.

[14] I. Heynderickx, H. De Raedt, and D. Schoemaker. Simulated anneal method for the determination of spin Hamiltonian parameters from ESR data. Journal of magnetic resonance, 70(1):134–139. Elsevier Science, 1986.

[15] W. A. Horn. Some simple scheduling algorithms. Naval Research Logistics Quarterly, 21(1):177–185. Wiley Online Library, 1974.

[16] A. S. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report. Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

[17] S. Jamali, F. Alizadeh, and S. Sadeqi. Task scheduling in cloud computing using particle swarm optimization. The Book of Extended Abstracts, 192, 2016.

[18] A.-Z. M. Kadhim, S. K. Ali, and M. M. Kassim. Solving Machine Scheduling Problem under Fuzzy Processing Time using the Simulated Annealing Method. Journal of Progressive Research in Mathematics, 14(1):2308–2317, 2018.

[19] A. R. Kan. Machine scheduling problems: classification, complexity and computations. Springer Science & Business Media, 2012.

[20] V. Kaplanoğlu. An object-oriented approach for multi-objective flexible job-shop scheduling problem. Expert Systems with Applications, 45:71–84. Elsevier, 2016.

[21] M. Kolonko. Some new results on simulated annealing applied to the job shop scheduling problem. European Journal of Operational Research, 113(1):123–136. Elsevier, 1999.

[22] J. Krishnaraj, S. Pugazhendhi, C. Rajendran, and S. Thiagarajan. Simulated annealing algorithms to minimise the completion time variance of jobs in permutation flowshops. International Journal of Industrial and Systems Engineering, 31(4):425–451. Inderscience Publishers (IEL), 2019.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680. American Association for the Advancement of Science, 1983.

[24] M. Lundy. Applications of the annealing algorithm to combinatorial problems in statistics. Biometrika, 72(1):191–198. Oxford University Press, 2015.

[25] M. Lundy and A. Mees. Convergence of an annealing algorithm. Mathematical programming, 34(1):111–124. Springer, 1986.

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092. AIP, 1953.

[27] F. A. Ogbu and D. K. Smith. The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem. Computers & Operations Research, 17(3):243–253. Elsevier, 1990.

[28] I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop scheduling. Omega, 17(6):551–557. Elsevier, 1989.

[29] C. N. Potts and L. N. Van Wassenhove. Approximation algorithms for scheduling a single machine to minimize total late work. Operations Research Letters, 11(5):261–266. Elsevier, 1992.

[30] C. Sel and A. Hamzadayi. A simulated annealing approach based simulation-optimisation to the dynamic job-shop scheduling problem. Pamukkale University Journal of Engineering Sciences, 24(4), 2015.

[31] N. Shivasankaran, P. S. Kumar and K. V. Raja. Hybrid sorting immune simulated annealing algorithm for flexible job shop scheduling. International Journal of Computational Intelligence Systems, 8(3):455–466. Taylor & Francis, 2015.

[32] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. Operations research, 40(1):113–125. INFORMS, 1992.

[33] M. P. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2(4):215–222. IEEE, 1983.