FLOWSHOP OPTIMISATION IN MATLAB – SIMULATED ANNEALING ALGORITHM APPROACH

Jaroslav Poživil

Department of Computer and Control Engineering, Prague Institute of Chemical Technology

1. INTRODUCTION

The purpose of this paper is to show that methods of artificial intelligence, simulated annealing algorithms in particular, are very effective at solving difficult, important real-world problems, specifically the optimization of serial multiproduct batch plant sequencing. We have found that Matlab realization of SA allows us to obtain optimum or good suboptimum solutions in acceptable times.

Batch processes are widely used in the chemical process industry and are of increasing importance due to a great emphasis on low-volume, high-value-added products (e.g., pharmaceuticals and specialty chemicals). Optimal planning and control of batch chemical plant offers immediate time- and money-savings with minimal investments into machinery. Our work focuses on plants where several different products are being manufactured using similar technology.

Multiproduct plant can be described as a periodical manufacturing of many batches of distinct products in a series of campaigns. In this paper, only a sub-category of multiproduct batch plants, a flowshop, is considered. The flowshop is a batch plant that consists of a fixed series of apparatuses and available apparatuses need not be assigned to tasks. Solving multiproduct batch plant production scheduling problem consists of two sub-problems. The first one is the problem of determining start-times and completion-times for all operations, i.e. creating detailed schedule for a known sequence of products. The second sub-problem, hierarchically a higher one, is that of a finding optimal sequence of products that satisfies contractual obligations. In the work presented in this paper, only permutation schedules (PS) are used. In this case, the sequence of products manufactured stays the same on all apparatuses, and the total number of configurations possible is (*n!*), where *n* is the number of products to be processed. Results are often used in form of Gantt charts (Fig. 1 shows simple example of such chart).

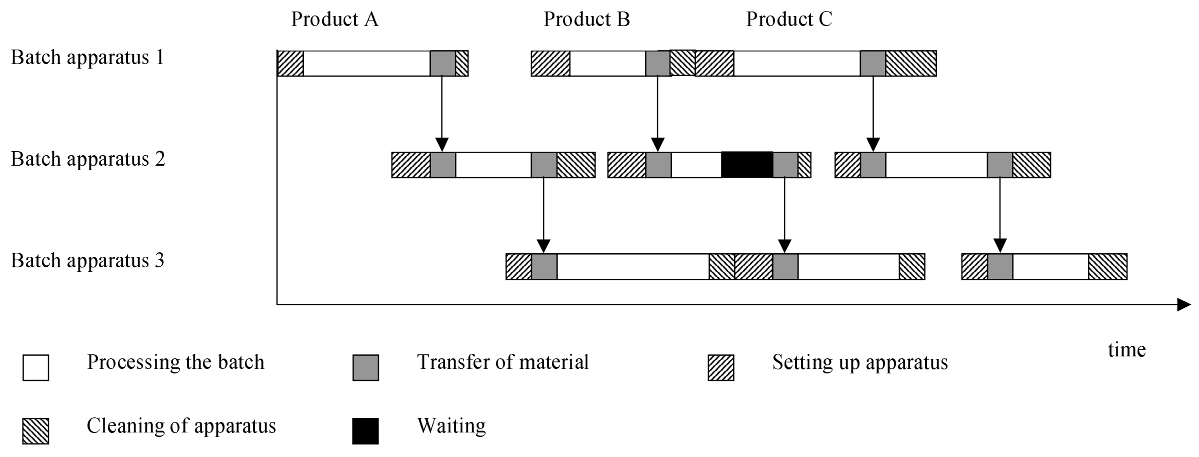


Fig. 1: Example of a Gantt chart

Finding optimal product sequence is a typical NP-complete problem. Computation time increases exponentially with increasing dimension of input dataset. The traditional method for solving these problems is mixed integer non-linear programming (MINLP). We don't believe it is well suited for solving problems with bigger dimension, such as those usually found in industry. The reasons for this are mainly long computation times and complexity of its formulations. The main difficulty is the size of resulting problems. For realistic problems, the MINLP problem may involve thousands (or tens of thousands) variables. Some of the alternatives to MINLP that have been found (see [1], [3]) are different variants of simulated annealing algorithm. The simulated annealing algorithm is a heuristic algorithm, and while this means that finding optimal solution is not guaranteed except in the case of infinite number of iterations, published works as well as our results show that the algorithm finds optimal or very good near-optimal solutions of problems solved.

2. PROBLEM DEFINITION

We focus on one of problems of batch processes scheduling studied in [5], on flowshops. Summarily, the scheduling problem for a flowshop can be stated as:

Given	set of *n* products to be produced; set of *m* processing units; set-up, transfer, processing and clean-up times; storage capacities; interstage policy.
Required	sequencing of products; assignment and timing of operations, for each unit (i.e. which task, if any, the unit performs at any time during the time horizon); and the flow of material through the network.
So as to *optimize*	a given objective criterion.

Work presented later in this paper uses objective criterion in the form of minimization of completion time. The completion time depends on the plant model used and on interstage policy used. This work uses relatively detailed representation of batch processes (and one that can even be used to describe semi-continuous operations) that breaks the operations down into several sub-tasks. Because some of the sub-tasks considered are sequence-dependent, i.e. they depend on the sequence of products, the resulting problem is MINLP formulation instead of MILP, and therefore is even more complicated than older, more simplified models, but this approach allows for more models realistic enough to be sufficiently applicable. The subtasks are: set-up, load and unload, processing, clean-up, waiting. Interstage policies that are considered in this work are the four most often used ones: unlimited intermediate storage, finite intermediate storage, no intermediate storage and zero wait.

Unlimited intermediate storage policy assumes that unlimited capacities for storing intermediate products are available between production steps. This means that when processing of batch *i* on apparatus *j* is finished, the apparatus *j* can be freed for processing batch *i+1* even if the apparatus *j+1* is still used simply by moving batch *i* into storage. The operation *i* on apparatus *j* is finished in time E_{ij}:

$$E_{ij} = \max\,[E_{i(j-1)}, E_{(i-1)j} + S_{(i-1)ij} + a_{i(j-1)}] + t_{ij} + a_{ij}\,, \qquad (1)$$

where $S_{(i-1)ij}$ is the time for setting-up empty apparatus *j* for processing batch *i* after batch *i-1*, $a_{i(j-1)}$ is the time to transfer product *i* from apparatus *(j-1)*, and t_{ij} is processing time of batch *i* on apparatus *j*. It is assumed that transfer time for moving batch *i* from apparatus *j* into apparatus *j+1* equals that of moving this batch from apparatus *j* into storage.

Finite intermediate storage policy places between production steps limited storage capacities. The number of the storage units between steps *j* and *j+1* is z_j. The equation for E_{ij} is then:

$$E_{ij} = \max\,[E_{i(j-1)} + t_{ij} + a_{ij},\ E_{(i-1)j} + S_{(i-1)ij} + a_{i(j-1)} + t_{ij} + a_{ij},\\ E_{(i-zj-1)(j+1)} + S_{(i-zj-1)(i-zj)(j+1)} + F(z_j)(S_{(i-zj)i0} + a_{(i-zj)j}) + a_{ij}]\,, \qquad (2)$$

where $F(z_j)$ is a function that equals 0 for z_j and otherwise equals 1. If all storage units are in use and apparatus *j+1* is also used, the batch must be stored for some time in apparatus *j*. The processing of batch *i* can be finished only if at least one of the storage units from the z_j units available is free, and this means that in apparatus *j+1* the processing of batch i-z_j must be started.

No intermediate storage policy assumes that no storage units are available, and therefore the intermediate products can only be stored inside the apparatus *j* when apparatus *j+1* is in use. The appropriate equations are:

$$E_{ij} = \max\,[E_{(i-1)j} + S_{(i-1)ij} + a_{i(j-1)},\ E_{(i-1)(j+1)} + S_{(i-1)i(j+1)} - t_{ij}] + t_{ij} + a_{ij}\,, \qquad (3.1)$$

in the case of the first apparatus, and

$$E_{ij} = \max\,[E_{i(j-1)},\ E_{(i-1)(j+1)} + S_{(i-1)i(j+1)} - t_{ij}] + t_{ij} + a_{ij}\,, \qquad (3.2)$$

in the case of all other apparatuses.

Zero wait policy requires that every batch must be transferred into next production step immediately after the completion of the current one, i.e. there are no storage units, and the intermediate product must not be stored in an apparatus. The computation of schedule is based on the computation of time E_{im} that represents the time of finishing of processing of batch *i* on apparatus *m*.

$$E_{im} = \max\,[(E_{(i-1)1} + S_{(i-1)i1} + \sum_{k=1}^{m} t_{ik} + \sum_{k=0}^{m} a_{ik}),\ (E_{(i-1)2} + S_{(i-1)i2} + \sum_{k=2}^{m} t_{ik} + \sum_{k=1}^{m} a_{ik}),\ \ldots$$

$$\ldots.\ (E_{(i-1)m} + S_{(i-1)im} + \sum_{k=m}^{m} t_{ik} + \sum_{k=m-1}^{m} a_{ik})] \qquad (4.1)$$

The remaining times are computed as follows:

$$E_{ij} = E_{im} - \sum_{k=j+1}^{m} t_{ik} + \sum_{k=j+1}^{m} a_{ik}\,. \qquad (4.2)$$

For more information on completion time algorithms, see ref. [2] & [4].

3. SIMULATED ANNEALING ALGORITHM

Simulated annealing algorithms are based on an analogy with the way metals cool and anneal. During slow cooling of heated metal the atoms begin to align themselves into crystals. During the cooling process transitions are accepted between low and high energy levels. The probability distribution describing this is Boltzmann distribution. Finally a pure crystal is formed, corresponding to minimum energy level allowed for the system. If a metal is cooled quickly and the substance is allowed to deviate from equilibrium, state of minimum energy is not reached and system ends in metastable, locally optimal structure. Optimization algorithm tries to simulate this physical process. Let E_1 be the objective function value of the initial solution and T_1 initial temperature. Basic version of the algorithm consists of the following two loops. In the inner loop, new solution E_2 is generated and if $E_2 \leq E_1$, the new solution is made the current solution and the process is repeated. In the case of $E_2 > E_1$ new solution is accepted/rejected randomly with probability based on Boltzmann distribution:

$$P(\Delta E) = \exp\left(\frac{E_1 - E_2}{K_B T_k}\right), \tag{5}$$

where K_B is Boltzmann constant and T_k current temperature. If the solution is accepted, it becomes current solution and the process continues, if the solution is rejected another solution is generated from the current one. In outer loop, the temperature is decreased accordingly to selected strategy of cooling.

To apply simulated annealing algorithm, following problem-dependent issues must be resolved:

a) an objective function to be minimized,

b) defining possible solutions based on given constraints,

c) choosing variant of algorithm and finding optimal parameter values.

Using the theory of Markov chains, it was shown that simulated annealing algorithm is guaranteed to converge to the set of optimal solutions, given an infinite number of iterations. However, this asymptotic behavior can be approximated in polynomial time. Iterations are stopped when defined termination criterion (pre-set temperature T_f is often used) is met, and such approximation algorithms are known to give near-optimal solutions.

The objective criterion can be created to include different requirements, and some of these are mentioned above. The objective criterion used in this work is the completion time, i.e. the time of the completion of the last operation on the last apparatus.

The simulated annealing algorithm can be said to be partially derived from local search algorithm. The algorithm starts at some point in solution space and searches only a part of the solution space known as a neighborhood of initial solution. The neighborhood is defined as a set of solutions that can be arrived at from current solution using some defined operations. Typical examples of neighborhood generation methods for permutation schedules are: swap of two elements (i.e. positions of two products in a sequence are swapped), swap of two neighboring elements, moving an element (i.e. some product's position in the sequence is changed), and reversal of sequence (i.e. positions of elements in part of the sequence are reversed). Both theory and practical tests shown us that best of these methods are swap of two elements and move of an element. No significant difference between systematic and random generation of the neighborhood was observed, and therefore systematic generation of neighborhood is to be advised.

In analogy to steepest vs. fastest descent variants of local search algorithm, two methods of exiting inner loop of SA algorithm exist. The first one, analogous to steepest descent, tries to reach the equilibrium as it searches the whole neighborhood of a current solution, and uses the best solution, before lowering temperature T_k. The second possibility is to exit the inner loop and to decrease the temperature as soon as a solution with lower objective function value is found.

At the start of the algorithm, the outer loop starts at temperature T_0. The value of the initial temperature can be estimated from the formula (6). If the Boltzmann constant K_B is set to 1 and the probability of accepting a move at the start of optimization is set to 90%, the resulting equation is:

$$T_0 = \frac{-\Delta C_{\mathrm{ij,max}}}{\ln 0{,}9} \approx 10\,\Delta C_{\mathrm{ij,max}}, \tag{6}$$

where $\Delta C_{\mathrm{ij,max}}$ is the maximum change of the value of objective criterion during a single transformation. The value of $\Delta C_{\mathrm{ij,max}}$ can be estimated by evaluating one or two hundred randomly generated transformations before the start of optimization.

The stopping temperature T_f should be selected as equal to or lower than the smallest observed change of value of objective criterion $\Delta C_{\mathrm{ij,min}}$.

The cooling that occurs in the outer loops can use one of several different algorithms. The simplest case is that of $T_0=0$, which means that the SA is reduced to local search algorithm. The three following algorithm are the most often used ones:

a) linear cooling

This method is often favored for its easy implementation. The starting temperature T_0, the stopping temperature T_f and the number of cooling steps *r* are defined, and the temperature is lowered by fixed value $\Delta = (T_0\text{-}T_f)/\mathrm{r}$.

$$T_{\mathbf{k+1}} = T_{\mathbf{k}} - \frac{T_0 - T_{\mathbf{f}}}{r} = T_{\mathrm{k}} - \Delta \tag{7}$$

The value of T_f can be set as a smallest observed difference between two neighboring solutions as observed during preliminary tests. The value of *r* is a compromise between accuracy (high values of *r*) and speed (low values of *r*).

b) exponential cooling

In this case, the new temperature is obtained by multiplying the current temperature by a constant α, $\alpha \in (0;1)$.

$$T_{\mathrm{k+1}} = \alpha\, T_{\mathrm{k}} \tag{8}$$

c) adaptive cooling

This method lowers the temperature accordingly to the results of statistical analysis of fluctuations of values of objective criterion at current temperature. First, the mean value $\overline{C}$ and the mean deviation $\sigma(C)$ are calculated for all the solutions generated in current inner loop. These values are then used to calculate new temperature using some formula. Most often used formula is probably the Aarts - van Laarhoven formula:

$$T_{\mathrm{k+1}} = T_{\mathrm{k}} \left[1 + \frac{\ln(1+\delta) T_{\mathrm{k}}}{3\sigma(C)} \right]^{-1} \tag{9}$$

The value of parameter δ is of high importance, as low values of δ ($\delta < 1$) slow down computation due to bad convergence, and high values of δ ($\delta > 1$) cause the algorithm to become stuck in non-optimal local extremes. Therefore the value of δ should be close to 1.

Determining the best method of cooling and its parameters is one of the most crucial problems when using SA algorithm, as these can vary with different problems and even with different numeric values of variables in solved problems.

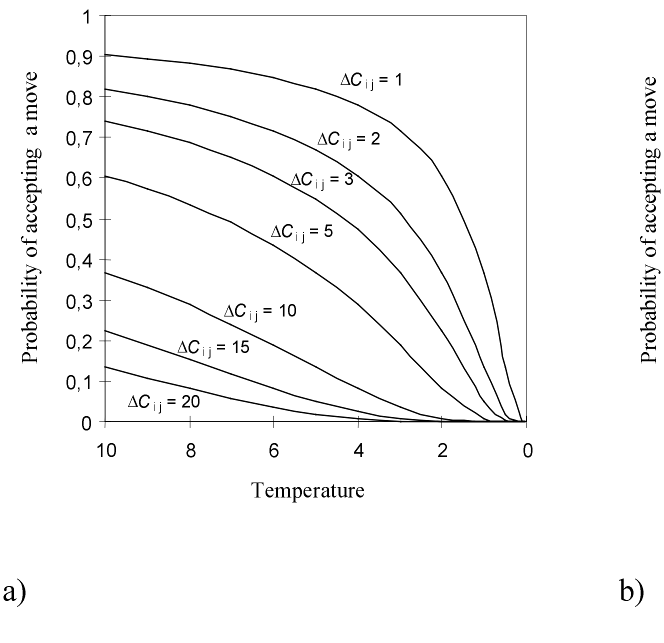
Different methods for deciding whether to accept new solution in the inner loop exist. The two criteria were used in our work and therefore should be mentioned. The first one is the

Metropolis criterion, which is the one used in basic variant of the SA algorithm as described above, the second one is the Glauber criterion. Both of the criteria are described in Tab. 1 describing the probability of accepting new solution.

	$C_{ij} < 0$	$C_{ij} \geq 0$
Metropolis	$P = 1$	$P = \exp(-\Delta C_{ij} / K_B T)$
Glauber	$P = \dfrac{\exp\left(-\Delta C_{ij} / K_B T\right)}{1 + \exp(-\Delta C_{ij} / K_B T)}$	

Tab. 1: Metropolis and Glauber criterion

The value of Boltzmann constant is usually set to one. The Metropolis criterion means that all new solutions that lead to the improvement of objective function value are accepted, and that as the temperature decreases the probability of accepting move that increases objective criterion value decreases as well. In the case of Glauber criterion, even the good moves can be rejected. The probability for accepting good move (i.e. one that improves objective criterion value) increases for solutions with higher improvement and when temperature is low, and the probability of rejecting bad move increases similarly. The probabilities for both criteria are shown in Fig. 2; the Fig. 2.a shows the probability of accepting bad move under Metropolis criterion, the Fig. 2.b shows acceptance probabilities for Glauber criterion.



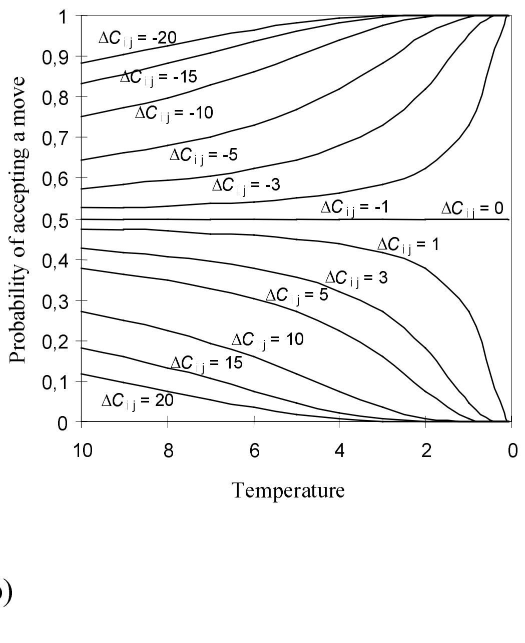


Fig. 2: Probability of accepting a move as a function of temperature; a) Metropolis criterion, b) Glauber criterion.

It is evident that the Metropolis criterion is somewhat faster and more direct, but does have bigger problems with local extremes. Our tests confirm this, as a minor difference in performance was observed when dealing with small problems (e.g. 10 products, 5 apparatuses), but no noticeable difference in performance was observed when bigger problems (e.g. 20 products, 10 apparatuses) were solved.

The SA algorithm is not guaranteed to find optimal solution in a fixed amount of steps. Computation termination criteria must be defined, both for the inner and the outer loop. The computations in the inner loop should be theoretically stopped when the system reaches the equilibrium. However, it is not possible to identify this state, and practical implementations of the algorithm usually end the inner loop after *L* steps, where *L* is some fixed, high enough value. The value of *L* is often set to the size of the neighborhood of a current solution y^i; in the case of swap of two elements transformation, the size of neighborhood is *n(n-1)/2* where *n* is the number of products. Some variants of the SA algorithm also exit the inner loop when solution that conforms to some criteria is found. The often used criteria for stopping the outer loop are the reaching of the stopping temperature T_f and the drop of the value of the derivation of a curve describing the value of objective criterion as a function of temperature bellow the pre-defined value ε (the curve is usually approximated by a 2nd degree polynomial).

4. COMPUTATIONAL RESULTS

We have tested the SA algorithm as a tool for solving sequencing problems in batch plants. The test problems were sets of randomly generated problems of different properties. Following are the details on test problems:

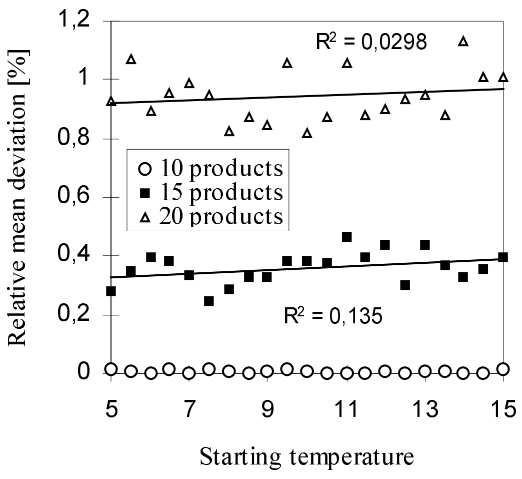
1. the plant operates as flowshop,
2. the plants operates under UIS, FIS, NIS or ZW policy,
3. the dimensions of the problems (*n*/*m*) are 5/10, 5/20, 8/4, 10/5, 10/15, 10/20, 15/5, 15/10, 20/5, 20/10,
4. neighborhood generation uses swap of two elements method,
5. randomly generated input data; processing times in range of 1-24; all other times: 1-4, number of storage units under FIS policy: 0-1.

Following are the results of testing several algorithm variants and parameter settings. It should be also noted that in cases of problems of large dimension (*n*=15, 20) it was not possible to find optimal solution by using complete enumeration method, and therefore other approach had to be selected. The optimal solution is assumed to be identical with the best solution of all the solutions obtained by repeated solving of the same problem by different algorithms.

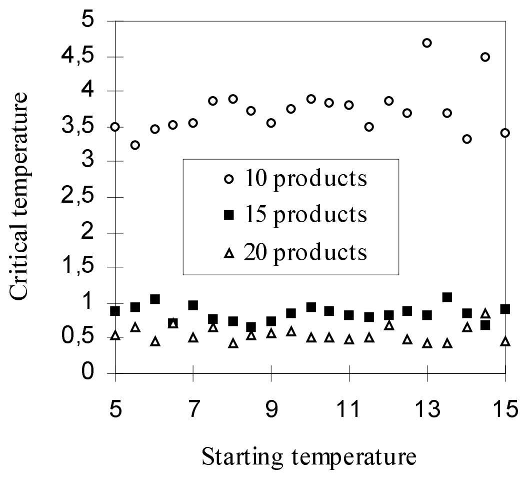
Starting and stopping temperature, number of iterations

The estimate of the starting temperature, as proposed in (5), is in fact more or less rule of thumb that does not take into account many of the properties of the problem solved. This means that in the case of scheduling problem as formulated in this work, this estimate is unnecessarily high. Should the estimate of $\Delta C_{ij,max}$=20 be made (the real value is usually even higher), the value of T_0 based on formula (5) should be 200. Our work shows that the performance of the algorithm is much better if the starting temperature is much lower, because the aforementioned estimate of T_0 results in long period of non-effective search at the start of the algorithm. Fig. 3.a shows the relative mean deviation of a solution for constant values of T_f=0.05 and *r*=50000, and Fig 3.b shows the average temperatures at which the optimal solution was found. It is evident that the optimal solutions are all found at temperatures lower than 5, and that the choice of starting temperature does not have significant impact at results. The observation that the algorithm performs better when the starting temperature is lower is probably due to the fact that the starting temperature is still high enough and then the fixed number of iterations means higher resolution and consequently more iterations in the proximity of the optimal solution.

The estimate of stopping temperature, i.e. a number equal to or smaller than $\Delta C_{ij,min}$, proved to be correct, as setting of this value to $\Delta C_{ij,min}$ (i.e. 1) was still too high. The probability of accepting a bad move would be too high, and the optimization would end prematurely, especially in cases of larger problems, as is evident in Fig. 3.b. The results of sensitivity analysis as displayed in Fig. 4 show that the algorithm is somewhat faster when the T_f is set at the lower end of the tested range.



a)



b)

Fig. 3: Sensitivity analysis: a) the distance from optimal solution as a function of starting temperature T_0; b) critical temperature as a function of starting temperature T_0.

The influence of the number of iterations (for T_0 and T_f constant) is shown in Fig. 5.a and Fig. 5.b. It should be evident that the decrease of relative mean deviation slows down as the number of iterations increases, and the compromise between computation time and accuracy must be reached, depending on the requirements and on computer hardware available. We think that r=70 000 is probably large enough value in cases of smaller problems, and the value of r=110 000 results in near-optimal solutions in cases of larger problems (i.e. n=20, and result less than 1% from optimum).

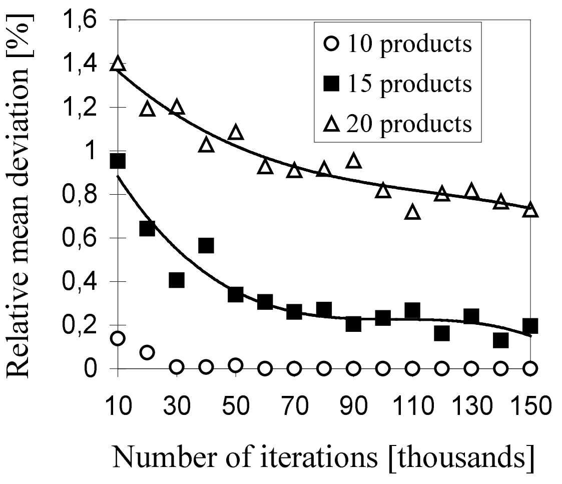


Fig. 4: Sensitivity analysis: critical temperature as a function of stopping temperature T_f.

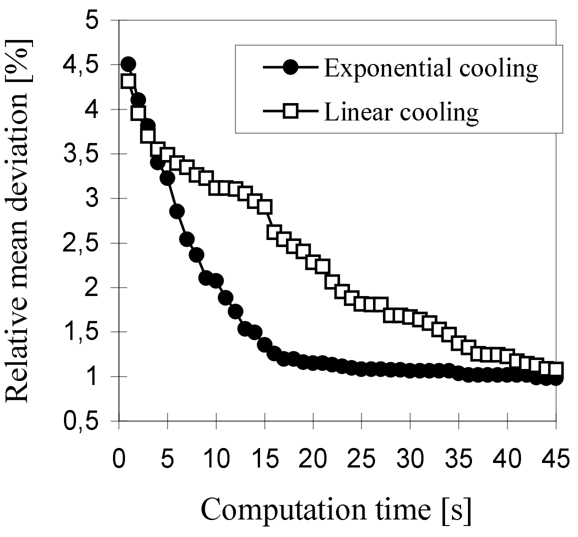
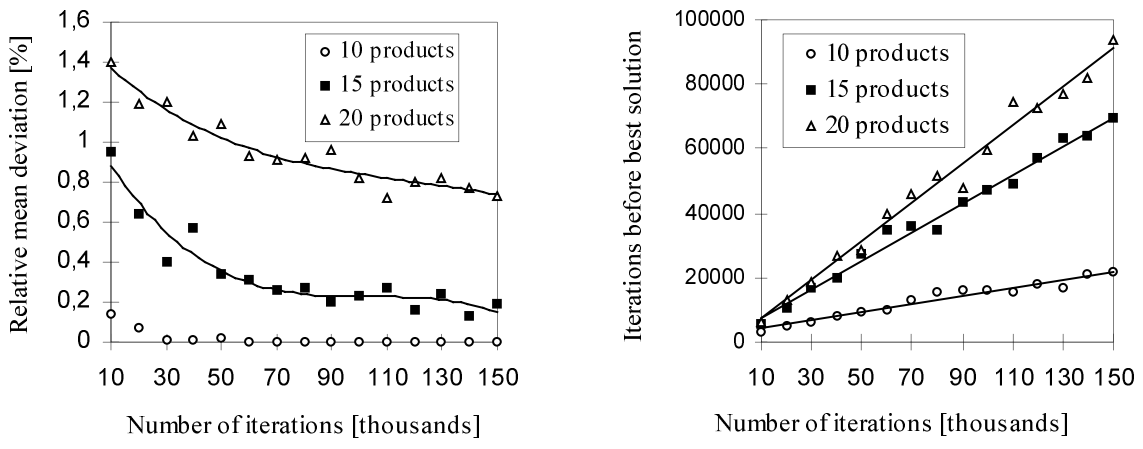


Fig. 6: The distance from optimum as a function of computation time.



a) b)

Fig. 5: a) distance from optimal solution as f function of a number of iterations; b) the number of iterations performed before best solution as a function of a total number of iterations in a run.

The result of these tests is the proposition to set the starting temperature T_0 to 5, the stopping temperature T_f to 0.05 and number of iterations to aforementioned values when problems of this type are solved, and when applying this SA algorithm to problems slightly more different to adjust these proposed values accordingly to principles and observations mentioned.

Lowering of temperature

Two different cooling strategies have been tested: linear cooling and exponential cooling. The results of the tests are shown in Fig. 6, and it should be clear that the exponential cooling offers better performance. This is probably due to the fact that as the exponential schedule spends less time in the initial search and more time in the lower-temperature area, there is more time for the system to reach the equilibrium.

The variant of the algorithm based on the fastest descent principle was also tested, but its performance was unsatisfactory. While being somewhat faster, the algorithm gave worse results, even when the computation termination criteria included zero derivation of the relative mean deviation decrease no significant change was observed.

5. CONCLUSIONS

We have created in Matlab a simulated annealing algorithm for solving scheduling problems. We have found that Matlab realization of simulated annealing algorithm allows us to obtain optimum or good suboptimum solutions in acceptable computation times. Matlab source code that implements this algorithm runs acceptable speed on PC-class computers. The batch processes model used in this work is detailed enough to suggest that the algorithm should be applicable to industrial problems with satisfactory results. The values of algorithm's parameters and its optimal version as found in our work are chosen so as to fit different problems of the solved type, and can be summarily listed in the following table.

Also, the simulated annealing algorithm should be flexible enough for it to be usable in other related areas, and to offer further improvement regarding batch process representation, scheduling constrictions etc.

Cooling schedule:	exponential cooling, steepest descent
Acceptance criterion:	Metropolis
Parameter values:	T_0=5, T_f=0.05, r=100 000, T_0 should not be greater than $\Delta C_{ij,max}$

REFERENCES

1. DAS H., CUMMINGS P. T., and LE VAN M. D.: Scheduling of serial multiproduct batch processes via simulated annealing, Computers chem. Engng., 1990, 14, 1351-1362
2. KIM M., JUNG J. H., and LEE I. B.: Optimal scheduling of multiproduct batch processes for various intermediate storage policies, Ind. Chem. Engng, 1996, 35, 4058-4066
3. KU H. M. and KARIMI I. A.: An evaluation of simulated annealing for batch process scheduling, Ind. Eng. Chem. Res., 1991, 30, 163-169
4. KU H. M. and KARIMI I. A.: Completion time algorithms for serial multiproduct batch processes with shared storage, Computers chem. Engng, 1990, 14, 46-69
5. STLUKA P.: Využití prvků umělé inteligence při rozvrhování vsádkových výrob, *(Use of artificial intelligence for scheduling of batch operations)*, Ph.D. thesis, VŠCHT Praha (1998) (in Czech)
6. POŽIVIL J., ŽĬÁNSKÝ M.: Use of simulated annealing algorithm for the flowshop sequencing, Hung. J. of Ind. Chem. 2000, 28, 259-265

Kontakt na autora:

Technická 1905, CZ-166 28 Praha 6, Czech Republic
E-mail: Jaroslav.Pozivil@vscht.cz
Tel.: +420 2 2435 4259, Fax: +420 2 2435 5053