Production Planning with Simulated Annealing

Karsten-Patrick Urban

Department of Production Theory and Information Science, University of Lüneburg, Germany, 1999/2003

E-mail: urban@uni-lueneburg.de

Combinatorial optimization is still one of the biggest mathematical challenges if you plan and organize the running of a business. Especially if you organize potential factors or plan the scheduling and sequencing of operations you will often be confronted with large-scaled combinatorial optimization problems. Furthermore it is very difficult to find global optima within legitimate time limits, because the computational effort of such problems rises exponentially with the problem size. Nowadays several approximation algorithms exist that are able to solve this kind of problems satisfactory. These algorithms belong to a special group of solution methods which are called *local search algorithms*. This article will introduce the topic of *simulated annealing*, one of the most efficient local search strategies. This article summarizes main aspects of the guest lecture *Combinatorial Optimization with Local Search Strategies*, which was held at the University of Ioannina in Greece in June 1999.

I. Problem Structure

Combinatorial optimization problems distinguish from linear programming problems (LP-problems) by the kind of the decision variables. Here you are confronted with discontinuous and not with continuous decision variables. Typical examples for combinatorial optimization problems are the facility layout planning or the sequencing of jobs on machines. To demonstrate the complexity of such planning problems consider the following production-planning-flow-shop-problem: If you have to sequence n jobs on m machines, there are $(n!)^m$ potential combinations. So, if you have 10 jobs which have to be sequenced on 3 machines you have the choice out of $4.778 \cdot 10^{19}$ combinations. If you relax this problem in that way that the jobs cannot overtake each other between the machines, there are still $n!$ (here: $10! = 3{,}628{,}800$) combinations.

II. Classification of Optimization Methods

There exist a lot of mathematical proceedings for solving combinatorial optimization problems. On the first level you distinguish between exact methods and heuristics. Exact methods always find at least one global optimum. The most trivial way to find a global optimum is to enumerate all possible combinations, e.g. all possible job sequences of a flow-shop-problem. More efficient are analytical methods. They are able to identify a global optimum without enumerating all combinations. Besides general approaches as the branch-and-bound-algorithm [Domschke et al. 1993] there also exist more specific algorithms that can only solve special problems. One example is the Johnson-Algorithm for the 2-machine-production-planning-flow-shop-problem. It always finds the job sequence with the lowest processing time [Johnson 1954].

As already mentioned the computational effort rises exponentially with the problem size. Therefore from the economical point of view the use of exact methods only makes sense for a small problem size. Large-scaled problems should be solved with heuristic methods. Their objective is to find a *good* solution (close to an optimum) within a reasonable period of time. Heuristics make use of special problem-specific and goal-directed rules which determine the proceeding of finding a solution. Only parts of the solution space are now considered for finding a solution. This reduces the computational effort a lot. On the other hand there is no guarantee to find global optima or to recognize solutions as global optima any more [Domschke et al. 1996].

Heuristics can be differentiated into construction and iterative heuristics. Construction heuristics are also called dirty or quick heuristics, because you find very fast a first feasible solution [Potts, Wassenhove 1991]. This solution is built with the help of *construction rules*. The quality of this solution is normally not very high [Reese, Urban 1999].

A solution of higher quality can be reached with iterative methods. They start with a feasible solution and search in the neighbourhood of this solution for a configuration with a better value of the objective function. So, before you can implement an iterative method you have to define how to generate a neighbour. The easiest way to generate a neighbour is a pairwise exchange of two elements of the initial configuration. But the establishment of a transition mechanism for generating the neighbourhood should be done from case to case, because the choice of this mechanism determines the efficiency of the algorithm [Hansmann 1997].

Iterative improvement (or hill climber) methods only accept neighbours as a new solution if the value of the objective function is better than that one of the current solution. You differentiate between first and best improvement. First improvement means that you compare the current solution only with one neighbour at the time. If you find a better configuration it replaces the current solution. Best improvement means that you compare a solution with all of its neighbours [Rixen 1997]. The best neighbour has to be chosen if it is better than the current solution. If no neighbour is found with a better function value, the algorithm ends. The problem is obvious: The algorithm can end in a local optimum. This local optimum can be far away from the global optimum. The solution quality depends on the initial configuration [Laarhoven, Aarts 1992].

This disadvantage of improvement methods can be solved with the help of local search algorithms. They also accept temporary worsening of the solution. So local optima can be overcome. For efficient work these algorithms need several control mechanisms, e.g. to prevent cycling or for defining a stop criterion [Domschke et al. 1996].

For completion it has to be mentioned that also combinations of construction and iterative heuristics or of improvement and local search algorithms are possible.

III. Simulated Annealing

The history of the local search algorithms in business administration is only about three decades old. The local search methods had proved that they lead to significant better results than construction heuristics if they are implemented carefully. Although the computational effort is much higher for local search algorithms than for quick heuristics, in relation to the solution quality and the computational effort of exact methods local search proceedings are the most efficient class of algorithms [Reese, Urban 1999].

One of the most popular and efficient representative of this class is simulated annealing. simulated annealing. As other famous local search techniques (e.g. tabu search, genetic algorithms) simulated annealing is originally developed outside of business administration. This is also reflected by the terms of its control mechanisms. These mechanisms have to be implemented very carefully to make simulated annealing as efficient as possible. In the following section the basic idea of simulated annealing will be described. Beside this basic form several variations and combinations with other heuristics can be found very often.

Analogy between Physical Process and Combinatorial Optimization

Originally simulated annealing was used for simulations of cooling processes of solids in statistical mechanics. The objective was to find the configuration of molecules with the lowest energy state, the ground state. Therefore the solid was heated up to its melting point and then cooled very carefully. After some time the system reaches the equilibrium, i.e. the minimum energy state, of the corresponding temperature. For reaching the equilibrium it can be necessary to increase temporary the energy level of the solid. So, e.g. crystals needs for their reorganization process a certain amount of energy. Therefore the temperature level should be held for a certain time period [Kuhn 1990].

If the system is cooled too fast, defects in the molecule configuration will be frozen. The system remains in a meta-stable state, i.e. in a local energy minimum (e.g. consider a crystal with several defects in its structure). The physical process shows analogies with combinatorial optimization: The ground state can be seen as the global optimum of an objective function, e.g. a global cost minimum. The meta-stable state complies with a local optimum, and the system states (configuration of the molecules) with the feasible combinations [Kirkpatrick et al. 1983, Bartholomeus, Weismantel 1989].

Physical System	**Combinatorial Optimization**
system state	configuration/combination
energy state	objective function value
ground state	global optimum
fast cooling	iterative improvement
slow cooling	simulated annealing

Fig. 1: Analogy between physical systems and combinatorial optimization problems

Metropolis Algorithm

The probability that the system have a certain system state *X* with the energy state E in its thermal equilibrium is given by:

$$Pr\{\, X = E \,\} = \frac{1}{Z(T)} \cdot e^{-\frac{E}{k_B \cdot T}} \tag{1}$$

Z (T):= regulating constant depending on the temperature *T*
k_B := Boltzmann-Constant
$e^{-\frac{E}{k_B \cdot T}}$:= Boltzmann-Factor

The amount of feasible system states decreases with a decreasing temperature. Approaching the zero point the system changes over to a final state. This cooling process can be simulated with the metropolis algorithm [Metropolis et al. 1953]: Starting with a feasible initial solution one molecule is replaced randomly. If the energy level decreases or stays the same ($\Delta E \leq 0$), the change will be done. An increasing energy ($\Delta E > 0$) level means that the change of the configuration will only be accepted with the probability $e^{-\frac{\Delta E}{k_B \cdot T}}$. The probability to accept a new configuration decreases by a high increase of the energy level and/or low temperatures.

Perturbation and Acceptance Probability

The metropolis algorithm is the base of simulated annealing. Instead of replacing molecules elements, e.g. jobs of a scheduling problem, are replaced. Any feasible solution which can be reached by one transition is a neighbour of the current solution. The probability to chose a configuration *Y* from the current configuration *X* is called *perturbation probability*. Theoretically it can be proven that the basic simulated annealing algorithm converges asymptotically to a global optimum. But a necessary condition for this convergence is a symmetric perturbation probability [Laarhoven, Aarts 1992][1]:

[1] If the perturbation probability is not symmetric other conditions have to be held for a convergence of the algorithm [Hajek, 1988].

$$p(X,Y) = p(Y,X). \tag{2}$$

If the configuration *Y* is a neighbour of the current solution the perturbation probability lies between 0 and 1, otherwise it is 0 and cannot be chosen from *X*:

$$p\,(X,Y) = \begin{cases} > 0\,, \text{if } Y \in N\,(X) \\ = 0\,, \text{if } Y \notin N\,(X) \end{cases}, \text{ with } \sum_{Y \in N} p(X,Y) = 1. \tag{3}$$

The decision to accept or to refuse the chosen neighbour as the new solution is done analogous to the metropolis algorithm. For a minimization problem this means if the value of the objective function of *Y* is lower than that one of *X* ($\Delta C = C\,(Y) - C\,(X) < 0$), *Y* replaces *X*, otherwise it is only accepted with a certain probability, depending on the increase ΔC and a *control parameter* T_k:

$$P\,(X,Y,T_k) = \begin{cases} 1 & \text{für } \Delta C \le 0 \\ e^{-\frac{\Delta C}{T_k}} & \text{für } \Delta C > 0 \end{cases}. \tag{4}$$

The control parameter T_k has the same function for the procedure of the method as the temperature of the metropolis algorithm, but no own economical meaning [Eglese 1990]. So it also strictly decreases during the proceeding. Therefore the probability to accept a configuration with a worse value of the objective function also decreases while running the algorithm.

The probability that a configuration *Y* replaces the configuration *X* results from the multiplication of perturbation and *acceptance probability*. Therefore a replacement of the current solution by a non-neighbour is excluded [Laarhoven, Aarts 1992]:

$$\Omega(X,Y,T_k) = \begin{cases} 0 & , Y \notin N(X) \wedge Y \neq X \\ p(X,Y) \cdot P(X,Y,T_k) & , Y \in N(X) \wedge Y \neq X \\ 1 - \sum\limits_{\Psi \in L, \Psi \neq X} p(X,\Psi) \cdot P(X,\Psi,T_k) & , Y = X \end{cases} \tag{5}$$

Formulation and Implementation of the Algorithm

Two different formulations of the simulated annealing algorithm are possible: The first one is the formulation as a homogenous algorithm. This means the control parameter is held fixed until an equilibrium is reached. After reaching an equilibrium the control parameter is reduced successively and the procedure is repeated for lower control parameter values as well. The algorithm ends after reaching a stop criterion by $\lim_{k \to \infty} T_k = 0$. For any fixed value of the control parameter a homogenous markov chain is generated. A markov chain is a series of configurations where a configuration *Y* only depends on the last accepted configuration *X*. The second kind of formulation of the method is that one as an inhomogenous algorithm. Then after each step the control parameter is reduced. So the proceeding is described as an inhomogenous markov chain . Theoretically both formulations converge asymptotically to a global optimum with a probability of one when certain conditions are held [Laarhoven, Aarts 1992]:

- *Homogenous Algorithm*: A with the problem size exponentially increasing amount of neighbours for reaching the stationary equilibrium has to be generated, and the limiting value of the control parameter is 0.
- *Inhomogenous Algorithm*: Convergence of the control parameter to the zero point may not be faster than $O\,([\log k]^{-1})$.

Both conditions cannot be held in a practicable implementation, because e.g. only a limited amount of neighbours can be taken into account. Therefore in a practicable implementation simulated annealing can only be a heuristic method. For the procedure of the algorithm a cooling scheduled is needed. The cooling schedule regulates the initial value of the control parameter, the stop criterion, the amount of iterations for each control parameter value and a function for the reduction of the control parameter value. You can differentiate between simple and more elaborate cooling schedules [Laarhoven, Aarts 1992].

The parameters of a simple cooling schedule are set externally. So the running of the algorithm occurs independently from the problem. The initial value of the control parameter is chosen in such a way that almost every neighbour will be accepted at the beginning of the proceeding, i.e. the acceptance probability $P\,(X,Y,T_0) = \min\,\{1;\ e^{-\frac{\Delta C}{T_0}}\} \cong 1$. One possibility to get an initial value of the control parameter is to chose an initial acceptance probability externally (e.g. 95 %) and calculate the average increase of the value of the objective function $\overline{\Delta C}^{(+)}$ for a certain amount of randomly chosen transitions:

$$T_0 = \frac{\overline{\Delta C}^{(+)}}{\ln(P^{-1})} \tag{6}$$

The algorithm ends when the solution has not changed for a certain amount of iterations or after running a preset amount of control parameter values [Laarhoven, Aarts 1992].

The number of iterations for each control parameter value can be described by the length of the single markov chains. The most trivial way for setting the number of iterations at each control parameter value is to chose an equal length of the markov chains depending (polynomially) on the problem size. An alternative is to fix a minimum number of transitions which have to be accepted for every control parameter value. By choosing this alternative a constant which limits the maximum number of iterations at each control parameter value should be established to prevent extremely long markov chains at low values of the control parameter [Laarhoven, Aarts 1992].

The value of the control parameter can be reduced quite simple with a reduction function of the following kind where the reduction factor should be close to one for guaranteeing a slow cooling [Laarhoven, Aarts 1992]:

$$T_{k+1} = \alpha \cdot T_k, \quad k = \{0; 1; 2; \ldots; K\}, 0 < \alpha < 1. \tag{7}$$

The advantage of an easy implementation of simple cooling schedules is faced with the disadvantage of a lower efficiency of the algorithm than using more elaborate cooling schedules. From time to time enormous empirical studies on a problem are necessary to determine the cooling parameters satisfactory [Kuhn 1990].

More elaborate cooling schedules run the algorithm problem-specific. The reduction of the control parameter value and the stop criterion depends on the operation of the proceeding and the values of the objective function. Several approaches for more elaborate cooling schedules exist. The following approach is only one proposal:

- *Initial value of the control parameter*: Standard deviation of the objective function value by a very high *temperature* [White 1984, Laarhoven, Aarts 1992]

$$T_0 \geq \sigma\ (T \to \infty) = \sqrt{< C^2 (T \to \infty) > - [< C(T \to \infty) >]^2}\ , \tag{8}$$

$< C\ (T\to\infty) >$ as the expected value of the objective function, $< C^2\ (T\to\infty) >$ as the expected square value of the objective function and $\sigma\ (T\to\infty)$ as standard deviation at a value of the control parameter of $T\to\infty$.

- *Stop Criterion:* The algorithm terminates when the difference between the average value of the objective function over a number of markov chains and the optimal solution reaches a lower limit [Laarhoven, Aarts 1992]:

$$\overline{\mathrm{C}}(\mathrm{T_K}) - \mathrm{C_{opt}} < \varepsilon. \tag{9}$$

- *Decrement rule for the control parameter*:

$$T_{k+1} = T_k \cdot \left(1 + \frac{\ln(1+\delta) \cdot T_k}{3\sigma(T_k)}\right)^{-1}, \tag{10}$$

$\sigma\ (T_k)$ as standard deviation of the objective function at T_k and δ as distance parameter.

This rule guarantees a slow decrease of the parameters. So the stationary equilibrium at (k+1) should be reached in a reasonable time [Laarhoven, Aarts 1992].

- *Length of markov chains*: The maximum amount of neighbours which can be reached from the current solution [Laarhoven, Aarts 1992].

$$\Lambda = \max\left\{ \left|L_X\right| : X \in L \right\}. \tag{11}$$

A modification: Threshold Accepting

A modification of simulated annealing is *threshold accepting*. Any randomly chosen neighbour which does not worse the value of the objective function more than a set threshold value Λ will be accepted as the new solution. Λ is a parameter which will be set to zero during the proceeding [Hansmann 1997]. The advantage of Threshold Accepting is that the computional effort is much lower than by simulated annealing, because no acceptance probability has to be calculated and no random numbers have to be generated. On the other hand it is a big disadvantage of this method that some configurations cannot be reached from another one, because a temporary worsening of the solution is only allowed up to the threshold value Λ.

IV. Example

An example shall demonstrate the basic procedure of the simulated annealing algorithm [Reese, Urban 1999]. Although the application of local search strategies is only recommendable for large-scaled problems because of the big implementation effort, the problem size of the example is held very small for an easier understanding.

Consider the following flow-shop-problem as shown in figure 2. The five jobs shall be scheduled in the same sequence on the three machines, i.e. no overtaking of the jobs is allowed. The objective is to find the job sequence with the shortest process time.

job / **machine**	**A**	**B**	**C**	**D**	**E**
1	14	26	1	52	5
2	24	6	2	18	15
3	31	23	5	14	2

Fig. 2: Process time of the jobs on each machine

For this scheduling problem there exist two optimal solutions: A-C-D-B-E and C-A-D-B-E. The corresponding process time for these solutions is 124 time units. The size of the solution space covers 120 (= 5!) job sequences. 76 knots are needed for finding one of the optimal solutions by using the constraint enumeration of Ignall and Schrage´s branch-and-bound-algorithm [Ignall, Schrage 1965]. A suboptimal solution results by solving the scheduling problem with the construction heuristic of Campbell, Dudek and Smith: The process time of the solution C-A-B-D-E is 128 time units.

Let the job sequence D-B-C-A-E be a randomly chosen initial solution, and the neighbourhood of the iterative search be defined as a pairwise exchange of neighbouring jobs. Then, the job sequences B-D-C-A-E, D-C-B-A-E, D-B-A-C-E, and D-B-C-E-A are the feasible neighbours of the initial solution. Because none of the neighbours has a shorter process time than the initial solution, a hill climber algorithm would get stuck in this local minimum. The simulated annealing algorithm can overcome this local minimum, and reach a better solution, even the global optimal solution.

control parameter value	**selected neighbour**	**processing time**	**acceptance probability**	**random number**	**current solution**
	D-B-C-A-E	150			D-B-C-A-E
$T_0 = 57$	D-B-C-E-A	156	0,90	0,19	D-B-C-E-A
	B-D-C-E-A	168	0,81	0,59	B-D-C-E-A
	B-C-D-E-A	167	1,00		B-C-D-E-A
	B-C-E-D-A	157	1,00		B-C-E-D-A
	B-E-C-D-A	157	1,00		B-E-C-D-A
$T_1 = 34$	E-B-C-D-A	157	1,00		E-B-C-D-A
	E-B-C-A-D	130	1,00		E-B-C-A-D
	E-B-A-C-D	130	1,00		E-B-A-C-D
	E-B-A-D-C	134	0,89	0,17	E-B-A-D-C
	E-B-D-A-C	161	0,45	0,42	E-B-A-D-C
$T_2 = 20$	E-A-B-D-C	134	1,00		E-A-B-D-C
	A-E-B-D-C	134	1,00		A-E-B-D-C
	A-E-B-C-D	130	1,00		A-E-B-C-D
	A-E-C-B-D	130	1,00		A-E-C-B-D
	A-C-E-B-D	130	1,00		A-C-E-B-D
$T_3 = 12$	A-C-B-E-D	130	1,00		A-C-B-E-D
	A-C-B-D-E	128	1,00		A-C-B-D-E
	A-C-D-B-E	124	1,00		A-C-D-B-E
	A-C-D-E-B	129	0,66	0,38	A-C-D-E-B
	A-D-C-E-B	130	0,92	0,93	A-C-D-E-B
$T_4 = 7$	C-A-D-E-B	129	1,00		C-A-D-E-B
	C-D-A-E-B	151	0,18	0,22	C-A-D-E-B
	C-A-D-B-E	124	1,00		C-A-D-B-E
	C-A-B-D-E	128	0,56	0,66	C-A-D-B-E
	C-A-D-E-B	129	0,49	0,51	C-A-D-B-E

Fig. 3: Exemplary solution steps of a simulated annealing implementation

Figure 3 shows exemplary the search process for an implementation of the simulated annealing algorithm. The initial acceptance probability in the example is set to 90 %, the average deviation of the objective function of 10 randomly chosen job sequences is 6 time units. Using equitation (6) leads to an initial temperature T_0 of 57. The number of markov chains and the length of the markov chains are externally set, and equal to the number of jobs. Equitation (7) is set as decrement rule of the control parameter, and α is set to 60 %. The solution of the local search is the job sequence C-A-D-B-E. The algorithm has found one of the optimal solutions.

References:

Bartholomeus, M. and R. Weismantel (1989): Simulated Annealing für die Plazierung von Standardzellen: Einführung, Implementierung und Ergebnisse, in: Rieder, U. (Editors), 14. Symposium on Operations Research of the Gesellschaft für Mathematik, Ökonomie und Operations Research e.V., Frankfurt am Main, pp. 171-190.

Campbell, H.G., Dudek, R.A. and M.L. Smith (1970): A Heuristic Algorithm for the n Job. m Machine Sequencing Problem, in: Management Science, Volume 16, pp. B-630-B-637.

Domschke, W., Klein, R. and A. Scholl (1996): Tabu Search - eine intelligente Lösungsstrategie für komplexe Optimierungsprobleme. In: Schriften zur Quantitativen Betriebswirtschaftslehre, Technische Hochschule Darmstadt, Nummer 4/96, pp. 1-8.

Domschke, W., Scholl, A. and S. Voß (1993).: Produktionsplanung - Ablauforganisatorische Aspekte. Springer: Berlin, Heidelberg, New York, Tokyo.

Eglese, R.W. (1990): Simulated Annealing: A tool for Operational Research, in: European Journal of Operational Research 46, pp. 271-281.

Hajek, B. (1988): Cooling Schedules for Optimal Annealing, in: Mathematics of Operations Research, Vol. 13, pp. 311-329.

Hansmann, K.W. (1997): Industrielles Management. 5., überarbeitete und wesentlich erweiterte Auflage, Oldenbourg: München, Wien.

Ignall, E. and L. Schrage (1965): Application of the branch and bound technique to some flow-shop scheduling problems, in: Operations Research, Volume 13, pp. 400-412.

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

Kirkpatrick, S.; C.D. Gelatt Jr. and M.P. Vecchi (1983): Optimization by Simulated Annealing, in: Science, Vol. 220, No. 4598, pp. 671-680.

Kuhn, H. (1990): Einlastungsplanung von flexiblen Fertigungssystemen, Physica-Verlag, Heidelberg.

Laarhoven, P.J.M. van and E.H.L. Aarts (1992): Simulated Annealing: Theory and Applications, Reprint, D. Reidel Publishing Company, Dordrecht.

Metropolis, N.; A.W. Rosenbluth; M.N. Rosenbluth; A.H. Teller and E. Teller (1953): Equation of State Calculations by Fast Computing Machines, in: Journal of Chemical Physics, 21, pp. 1087-1092.

Potts, C. N. and L.N. van Wassenhove (1991): Single Machine Tardiness Sequencing Heuristics. In: IIE Transactions, Volume 23, Number 4, pp. 346-354.

Reese, J. and K.-P. Urban (1999): Produktionsplanung mit Hilfe von lokalen Suchverfahren, in: wisu - das wirtschaftsstudium 3/99, pp. 318-324.

Rixen, I. (1997): Maschinenbelegungsplanung mit Evolutionären Algorithmen, DUV: Wiesbaden.

White, S.R. (1984): Concepts of Scale in Simulated Annealing, Proceedings of IEEE International Conference on Computer Design, Port Chester, November 1984.