The New Concepts in Parallel Simulated Annealing Method

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Institute of Engineering, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
email: wbo@ict.pwr.wroc.pl
[2] Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51-151 Wrocław, Poland
email: mwd@ii.uni.wroc.pl

Abstract. New elements of the parallel simulated annealing method are proposed to solve the permutation flow shop scheduling problem with the criterion of total completion time ($F^*||C_{Sum}$). This problem is more difficult to optimize than $F^*||C_{\max}$ (minimizing the makespan). Simulated annealing belongs to the artificial intelligence methods, which are commonly used to solve NP-hard combinatorial optimization problems. In the parallel algorithm, we propose a new acceptance probability function, multi-step technique, dynamic long-term memory and backtrack-jump. Computational experiments (for Taillard's benchmarks ta001-ta050, [8]) are given and compared with results yielded by the best algorithms discussed in literature [10]. We also present new referential solutions for ta051-ta080 instances (published on our benchmark page [1]), which so far have no solutions. *

1 Introduction

Finding optimal solutions to the NP-hard combinatorial problem is not generally realistic. For large-size problems, an exact algorithm may not be useful since it is computationally too expensive. Approximate algorithms, which do not guarantee finding an optimal solution, are a good alternative. Some popular metaheuristics are: simulated annealing, taboo search and genetic methods.

The simulated annealing (SA) is a stochastic heuristic method approach designed to find a near-optimal solution of the combinatorial optimization problem, motivated by the physical process of crystallization. This method was first introduced by Metropolis et al. [6], while Kirkpatrick et al. [4] and Cerny [2] applied the technique to optimization problems. It has gained popularity and there are many papers in literature on the subject with successfully implemented scheduling problems.

The simulated annealing algorithm starts from an initial solution. Each iteration randomly selects a solution from the neighborhood and evaluates its

* The work was supported by KBN Poland, within the grant No. T11A01624

objective function. If the solution is superior then it is accepted, if the solution is interior, it is accepted with some probability. This probability is calculated by an acceptance function depending on the parameter called temperature. The accepted element becomes the initial in the next iteration of the algorithm. Therefore, there are three essential steps of the SA method: (1) neighborhood, (2) acceptance function, (3) cooling scheme.

Basic problems which appear during designing of local improvement algorithms (simulated annealing and tabu search), are: (a) intensification of calculations – rapid attaining of local minimum, and (b) diversification of calculations – after reaching the local minimum, jumping into another 'good' region of the feasible solutions space.

We take under consideration the permutation flow shop scheduling problem described as follows. A number of jobs are to be processed on a number of machines. Each job must go through all the machines in exactly the same order and the job order is the same on every machine. Each machine can process at most one job at any point of time, and each job may be processed on at most one machine at any time. The objective is to find a schedule that minimizes the sum of completion time of the jobs.

In this paper we describe the parallel simulated annealing algorithm for the flow shop scheduling with the objective of minimizing the sum of completion times, and we propose many modifications of the classical simulated annealing method. The new elements of the method: are acceptance probability function, multi-step, dynamic long term memory and backtrack-jump. Computational experiments are compared with the results yielded by the best algorithms discussed in literature. New benchmarks,not available in literature, are published in [1].

2 Problem formulation

Each of n jobs from the set J={1,2,...,n} has to be processed on each of m machines from the set M={1,2,...,m} in succession. Job $j \in J$ consists of a sequence of m operations O_{j1}, O_{j2},...,O_{jm}. Operation O_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time p_{jk}. We want to find such a schedule that the sum of the job's completion times is minimal.

Let π =($\pi(1)$, $\pi(2)$,...,$\pi(n)$) be a permutation of jobs {1,2,...,n} and let Π be the set of all permutations. Each permutation $\pi \in \Pi$ defines a processing order of jobs on each machine. We wish to find a permutation $\pi^* \in \Pi$ that:

$$C_{sum}(\pi^*) = \min\{C_{sum}(\pi): \ \ \pi \in \Pi\}$$

where $C_{sum}(\pi) = \sum_{i=1}^{n} C_{i,m}(\pi)$ and $C_{i,j}(\pi)$ is the time required to complete job i on the machine j in the processing order given by the permutation π. In literature there are a few papers discussing this problem. Wang [9] shows some constructive algorithms based on the heuristic rule such as LIT and SPD. In the paper of Yamada et al. [10] a metaheuristic method is proposed to solve the problem. The representative neighborhood in this method incorporates the stochastic sampling and tabu search combined with genetic local search. The algorithm is applied to Taillard's benchmark problems t001-t050 from [8].

3 Simulated annealing method

A combinatorial optimization problem can be defined as a problem with well defined finite solution space Ω and objective function $F(\pi), \quad \pi \in \Omega$. The problem consists in finding an element π^* such that $F(\pi^*) = \min\{F(\pi): \quad \pi \in \Omega\}$.

For every element $\pi \in \Omega$ we call a ***move*** transformation $r \,:\, \pi \to r(\pi) \in \Omega$. Let $M(\pi)$ be a fixed set of moves on element $\pi \in \Omega$, and let $M(\Omega) = \bigcup_{\pi \in \Omega} M(\pi)$ be a set of all moves. We call a ***neighborhood*** $N(\pi)$ of element π a set of elements generated by moves from $M(\Omega)$, i.e. $N(\pi) = \{r(\pi): \quad r \in M(\Omega)\}$.

Simulated annealing is a randomized local improvement method, which accepts solutions depending on the value of the cost function. In each iteration of the SA algorithm a random perturbation is made to the current solution $\pi \in \Omega$, giving rise to the set $N(\pi)$ of neighbors. The neighbor $\beta \in N(\pi)$ is accepted as the next configuration with probability function $\Psi(\pi, \beta, T)$. The $\Psi(\pi, \beta, T)$ is known as *accepting function* and depends on control parameter T (*temperature*). Its value changes at suitably chosen intervals. In practice the function $\Psi(\pi, \beta, T)$ is chosen in such way that solutions corresponding to large increases in cost have a small probability of being accepted, whereas solutions corresponding to small increases in cost have a larger probability of being accepted.

Sequential simulated annealing algorithm sSA

Let π be any feasible solution, $N(\pi)$ – the neighborhood of π, π^* – the best known solution, L – the number of iterations for fixed value of temperature T.

***Step 0*:** $\pi^* \leftarrow \pi$;
***Step 1*:** <u>**for**</u> $k = 1, 2, \ldots, L$ <u>**do**</u>
Select at random $\beta \in N(\pi)$;
<u>**if**</u> $F(\beta) < F(\pi^*)$ <u>**then**</u> $\pi^* \leftarrow \beta$;
<u>**if**</u> $F(\beta) < F(\pi)$ <u>**then**</u> $\pi \leftarrow \beta$
<u>**else**</u> <u>**if**</u> $\Psi(\pi, \beta, T) >$ random([0,1)) <u>**then**</u> $\pi \leftarrow \beta$;
***Step 2*:** Modify the temperature;
***Step 3*:** If stop criterion is not true then go to Step 1.

Change of temperature follows according to the cooling scheme. Initial T_0 is determined experimentally. The most common acceptance function in SA algorithms is Boltzman function $\Psi(\pi, \beta, T) = \exp(-[(F(\beta) - F(\pi))/T])$, geometrical cooling scheme $T \leftarrow c \cdot T$ $(0 < c < 1)$ and moves: insert or swap.

3.1 Intensification and diversification of calculations

In this chapter we present a new acceptance function and cooling scheme. In order to intensify calculations we introduce:

a) backtrack jump – return to the neighborhood where improvement of the current solution takes place,
b) changes of temperature – enable on exact exploration of the promising region.

For better diversification of calculations we will apply:

a) multi-step – moving computations to an other remote region of solutions,
b) changes (increasing) of temperature enabling approval of considerably worse solutions.

Acceptance function and cooling scheme. If for randomly determined permutation $\beta \in N(\pi)$ there occur $C_{\max}(\beta) < C_{\max}(\pi)$, then β is the base solution in the next iteration. Contrarily, i.e. when $C_{\max}(\beta) \geq C_{\max}(\pi)$, the probability of acceptance in place of the base solution in the next iteration is determined by the acceptance function. We propose a new acceptance function:

$$\Psi_{\alpha,\lambda}(\pi,\beta) = \exp\left[\left(\frac{C_{Sum}(\beta) - C_{Sum}(\pi)}{C_{Sum}(\pi^*)}\right) \cdot \frac{\ln\lambda}{\alpha}\right] ,$$

which also depends on the best solution π^* determined until now. Parameters α i λ $(0 < \lambda < 1,\ \alpha > 0\ )$ act the role of changing the temperature in the classical acceptance function. We can intensificate or diversificate calculations changing these parameters (cooling scheme). By reducing them we intensificate calculations, whereas by enlarging them we make it possible to move away from the current local minimum (diversification of computations).

Backtrack-jump. Let π be the current base solution, T_π the temperature and π^* the best solution determined until now. By ***LM*** we denote long-term memory. On ***LM*** we will record certain attributes of the algorithm's iteration: base solution and temperature. If for randomly chosen permutation $\delta \in N(\pi)$, $F(\delta) < F(\pi)$ then we record on ***LM*** the pair (π, T_π), in other words $LM \leftarrow LM \cup (\pi, T_\pi)$.

If 'return condition' comes out (e.g. lack of improvement of the best solution after execution of a fixed number of iterations), then we get the pair (β, T_β) from the ***LM*** memory. In place of the current base solution of the algorithm we take permutation β, and T_β becomes the current temperature. In SA algorithms long-term memory ***LM*** is implemented using stack or queue.

Multi-step. If during a certain number of iterations the value of objective function is growing, then we execute a multi-step: generating distant permutation (in the sense of the number of moves) from the current base solution.

Let $\beta_s, \beta_{s+1},\ \ldots\ , \beta_t$ be a search trajectory, i.e. sequence of consecutive base solutions. If $F(\beta_s) < F(\beta_{s+1}) <, \ \ldots\ , < F(\beta_t)$ and $|t-s| > Lbp$, where Lbp is a fixed parametr, then we execute the multi-step. For fixed parameter k we generate permutation $\delta = r_k(r_{k-1}(, ..., r_1(\beta_t), ...,))$, where $r_1, r_2, ..., r_k$ are randomly generated moves. We take permutation δ as a base solution and $T_\delta \leftarrow T_0$.

4 Parallel simulated annealing pSA

The first parallelizations of the simulated annealing method were based on the crossing along a single trajectory through the solution's space (co called single-walk strategy). There were two ideas used in this strategy: move acceleration and parallel moves. Kravitz and Rutenbar [5] described one of the first examples of this type of parallelization of the simulated annealing metaheuristics.

Another method, crossing along a multiple trajectory through the solution's space (multiple-walk strategy), was proposed by Miki ed al. [7] and Czech [3]. This method is based on the execution of several threads (independent, semi-independent or cooperative) running simulated annealing at different temperatures. Such a model was the base of our researches.

New elements of the pSA method: intensification and diversification, multistep and long-term memory were implemented in parallel as follows. A master - slave model was used in implementation of the parallel algorithm. The master process keeps shared data, such as the best known solution and backtrack-jump heap. Slave processes $i = 1, 2, \ldots, p$ run its own simulated annealing threads with different temperatures $T^i = (\alpha^i, \lambda^i)$. If one process finds a new best solution π^*, then it sends it to the master process and runs an intencification procedure for next $S = 10$ iterations. Slave processes obtain up-to-date π^* from the master process every $K = 10$ iterations. If the process has no improvement of its own search procedure for $R = 20$ 'big' iterations (without modification of temperature), it runs the backtrack-jump procedure, consisting of getting new current solutions from the backtrack heap of the master process. Both, sequential and parallel algorithms, make n iterations without modification of the temperature inside one 'big' iteration.

5 Computational results

The algorithm was coded in Ada95 and run on 4-processors Sun Enterprise 4x400MHz, tested on the benchmark problems taken from the literature (see OR-Library [8] for more details). Main tests were based on 50 instances with 100, ..., 500 operations. Every instance of the test problems was executed 8 times, and the average and minimal result was used for comparing. The standard deviation of results was computed too – it was the measure of algorithm stability.

Results of tests are shown in Table 1. Starting solutions for the first process were taken from the quick approximate algorithm NEH, other processes start with random solutions. For all tested algorithms, the number of iterations was counted as a sum of iterations on processors. All algorithms, sequential and parallel, make 20.000 iterations, so the 4-processor implementations make 5.000 iterations on each of 4 processors.

Table 1. Relative distances to reference solutions [10].

$n \times m$	1 processor (sSA)			4 processors (pSA)		
	average	minimal	std. dev.	average	minimal	std. dev.
20×5	0.23%	0.05%	0.16%	0.08%	0.00%	0.08%
20×10	0.27%	0.04%	0.17%	0.05%	0.00%	0.05%
50×5	0.14%	0.01%	0.12%	0.04%	0.00%	0.04%
50×10	1.51%	1.25%	0.19%	1.25%	0.91%	0.20%
50×20	1.97%	1.56%	0.26%	1.54%	1.09%	0.29%
average	0.82%	0.58%	0.18%	0.59%	0.40%	0.13%

As we can see in Table 1 the average distance to reference solutions for sequential sSA and parallel pSA was at the level of 0.82% for 1-processor implementation and at the level of 0.59% for 4-processors implementation. All algorithms have the same number of iterations and comparable costs. Additionally, the results of the parallel algorithm in 8 runs was more stable: standard deviations of the result equaled to 0.13%, compared to 0.18% for the sequential algorithm. Another test was made for comparing algorithms in the matter of convergence's speed. Tests were made on the first 10 instances of the benchmarks [8], the average distance to reference solutions was used for comparison. As we can see in Figure 1 the parallel algorithm achieves better results than the sequential algorithm after the same number of iterations (as the sum of iterations on each processor).

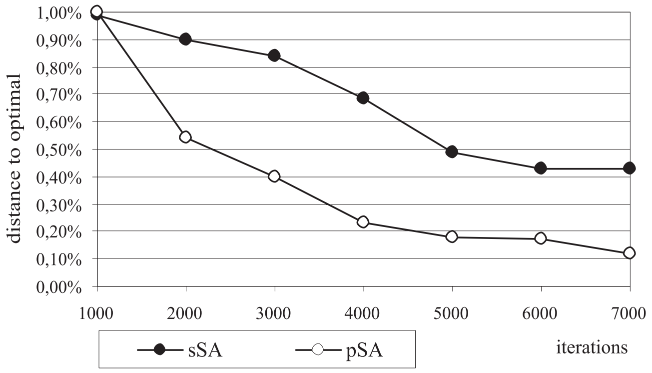

Fig. 1. Comparison of convergence for sSA and pSA algorithms.

6 Conclusions

We have discussed a new approach to parallel simulated annealing based on asynchronous multiple-walk strategy. As we can see parallelization increases the quality of the solutions obtained. The idea of the backtrack-jump and the multi-step was used. Computer experiments show, that the parallel algorithm is considerably more efficient and stable in comparison to the sequential algorithm.

References

1. Benchmark library: http://www.ii.uni.wroc.pl/~mwd/wobz.html
2. Černy V.: Thermodynamical approach to travelling salesman problem: An efficient simulation algorithm. J. Optim. Theory Appl. **45** (1985) 41–51
3. Czech Z.: Three parallel algorithms for simulated annealing. Lecture Notes in Computer Science **2328**, Springer Verlag (2002) 210–217
4. Kirkpatrick S., Gellat C.D., Vecchi M.P.: Optimization by simulated annealing. Science **220** (1983) 671–680
5. Kravitz S.A., Rutenbar R.A.: Placement by simulated annealing on a multiprocessor. IEEE Transactions on Computer Aided Design **6** (1998) 534–549
6. Metropolis N., A.W. Rosenbluth, A.H. Teller, E. Tellet: Equation of state calculation by fast computing machines. Jurn. Of Cem. Phys. **21** (1953) 1187–1191
7. Miki M., Hiroyasu T., Kasai M.: Application of the temperature parallel simulated annealing to continous optimization problems. IPSL Trans. **41** (2000) 1607–1616
8. OR-Library: http://mscmga.ms.ic.ac.uk/info.html
9. Wang C., Chu C., J. Proth J.: Heuristic approaches for $n|m|F|\sum C_i$ scheduling problems. European Journal of Operational Research (1997) 636–644
10. Yamada T., C.R. Reeves: Solving the C_{sum} Permutation Flowshop Scheduling Problem by Genetic Local Search. IEEE International Conference on Evolutionary Computation ICEG 98 (1998) 230–234