**Institute of Internet and Intelligent Technologies**
Vilnius Gediminas Technical University
Saulėtekio al. 11, 10223 Vilnius, Lithuania
*http://www.isarc2008.vgtu.lt/*

**The 25th International Symposium on Automation and Robotics in Construction**

**June 26–29, 2008**

ISARC-2008

# VARIABLE NEIGHBORHOOD SIMULATED ANNEALING METHOD AND APPLICATION FOR DESIGN

**Gražvydas Felinskas**
Šiauliai University
Department of Informatics
P. Visinskio str. 19, Šiauliai
e-mail: grazvis@splius.lt

**Leonidas Sakalauskas**
Šiauliai University
Department of Informatics
P. Visinskio str. 19, Šiauliai
e-mail: sakal@ktl.mii.lt

## ABSTRACT

In this paper we discuss about production scheduling. In manufacturing industry, efficient methods to solve resource constrained scheduling problems are needed. The aim is to find such schedule, which would meet the requirements of job priority relations, resource constraints, minimizing it by some criteria. In many cases this criterion is project's finishing time. We consider job scheduling and optimization algorithms related to resources, time and other constraints. In most cases, scheduling problems belongs to complexity class NP. In order to schedule and optimize jobs, we may apply heuristic methods. We explore application of Simulated Annealing (SA) method combined with variable neighborhood to schedule optimization. Computational results are given using data sets from the project scheduling problems library. Efficiency of the algorithm was tested on high performance computing machine.

## KEYWORDS

Resource Constrained, Schedule Optimization, Simulated Annealing, Variable Neighborhood.

## 1. INTRODUCTION

Production scheduling is an important part of the production planning of many manufacturing and construction companies. By scheduling it is possible to find the proper sequence to do the jobs and the proper schedule, when each operation of the job should be processed at each stage of the production or construction process.

Traditional scheduling methods, such as PERT and CPM [1], [2], are not enough for production scheduling, because they consider infinite resources. Scheduling with infinite resources may give results, which are not feasible. In manufacturing industry, efficient methods to solve resource constrained scheduling problems are needed. Input data for such problems are a set of jobs, their durations, priority rules (successors, predecessors) and necessary resources. The aim is to find such schedule, which would meet the requirements of job priority relations, resource constraints, minimizing it by some criteria. In many cases this criterion is project's finishing time. We consider job scheduling and optimization algorithms related to resources, time and other constraints. In most cases, scheduling problems belongs to complexity class NP [3].

Optimal solution can be found using full binary recombination [4]or using methods based on ideas of branch and bound methods [1], [5] etc. It is usually difficult or impossible to perform a full binary recombination in acceptable time, therefore, in order to schedule and optimize jobs, we may apply heuristic methods, based on priority rules [6], evolution process ideas [7], [8], local search [9], [10], [11], variable neighborhood search [12], [13], etc. We explore application of Simulated Annealing (SA) [14] method to schedule optimization because some investigations show that development of SA is a perspective trend to create efficient scheduling methods.

Computational results are given using data sets from the project scheduling problems library (PSPLib) [15], [16]. This library contains different problem sets for various types of resource constrained project scheduling problems as well as optimal and heuristic solutions. The data sets may be used for the evaluation of solution procedures for single-mode and multi-mode resource-constrained project scheduling problems (RCPSP) [13], [16], [17].

However the problem is that RCPSP are NP-complete and the solving is hard limited by available computer resources. Testing on a personal computer enabled us to solve tasks of the middle size, when dimensionality of real-life problems can exceed hundred or thousands tasks. Solving of large problems require the great number of iterations for each instance (5000-30000). Thus, the parallelization of the algorithm and computing in a high performance computer center were necessary. A lot of applications show that heuristic methods inherently suits for parallelization and can easily be implemented in robust computer code. The parallelization strategy consists of sharing the computation of identical instances of problems among processors and using control processor to manage the adaptation of the sample size.

The parallel computing programs were developed and computational results were got. All computational results were got at the high performance computing center CINECA, Italia (HPC-Europe project). The sample of application to construction design is given.

## 2. RESOURCE CONSTRAINED SCHEDULE OPTIMIZATION PROBLEM

### 2.1. Formulation of the Problem

Let's denote $J = \{0,1,...,n,n+1\}$ a set of jobs, $d_j, j \in J$ - duration of jobs. We can define priority relations in a set $J$ as a set of pairs $C = \{(i,j) \mid i \text{ must be executed before } j\}$. Let's denote a set of resources $K = \{1,..., m\}$. All resources are renewable and non-additive. Let's assume that amounts of resources $R_k > 0, k \in K$ are constant. Let's denote the starting moment of $j^{th}$ job by $s_j \geq 0$, and correspondingly $r_{jk} \geq 0$ - amount of $k^{th}$ resource, needed for performing this job. Started jobs must be performed without breaks. The finishing time of $j^{th}$ job $c_j = s_j + d_j$. The problem of schedule making may be reduced to the problem of finding a vector $s = (s_0, ..., s_{n+1})$ of jobs' starting time which meets priority and resource requirements and minimizes a certain objective function. Project finishing time is one of the most analyzed schedule optimality criteria which was applied in this paper. Let's denote $A(t) = \{ j \in J \mid s_j \leq t < c_j \}$ - a set of executable jobs at the time moment $t$, $T(s)$ – project finishing time, $T(s) = c_{n+1}$. After these definitions we can formulate the problem minimizing the objective function $T(s)$.

Find $\min_s T(s)$, subject to:

$$c_i \leq s_j, (i,j) \in C,$$

$$s_j \geq 0,\ c_j = s_j + d_j,\ j \in J, \quad \textbf{(1)}$$

$$\sum_{j \in A(t)} r_{kj} \leq R_k,\ k \in K,\ 0 \leq t \leq c_{n+1}. \quad \textbf{(2)}$$

The objective function defines the finishing time of the whole job project, inequality (1) describes priority relations, and the inequality (2) requires to pay heed to resource constraints.

### 2.2. Schedule Coding and Decoding

Efficiency of schedule optimization algorithm depends on solution coding [12], [13]. In this paper, we analyze job scheduling problems under resource constraints with solution coding in a shape of a priority list [5], [18]. The job priority list $b=(0,b_1,b_2,...,b_n,b_{n+1})$ can be determined by jobs' starting times vector $s$, where $s_{b_i} \le s_{b_j}$, if $i<j$. It is very important that this vector of starting moments must meet priority relations and resource constraints. On the other hand, for given priority list, we can find the vector of job starting times concerted with priority list and initial priority constraints. For this we use serial decoder detailed described in [19], [20].

At first, we must be sure that the priority list is concerted with priority constraints. Let's denominate the priority list from which we can find a vector of jobs' starting times concerted with priority relations as a feasible priority list. Using a set of priority relations we can check whether the solution is feasible or not. For any feasible job priority list, by applying the serial decoding procedure, we can determine jobs' starting times vector which may minimize a project finishing time, according to priority relations, resource constraints, and a given job priority list. Operating with job priority lists, the constructed algorithm must enable us to find such job priority list which corresponds to an optimal solution of the problem. (1), (2). Determination of admissibility of the job priority list is detailed described in [19]. This determination is based on full priority relations' matrix which describes all jobs which must be done corresponding to all chains of priority relations.

### 2.3. The Serial Priority List Decoder

This procedure computes the early starting moments of jobs, according to jobs' priority list concerted with priority relations, and resource constraints. In schedules obtained in such way, none of jobs can be started earlier than calculated time, without break of priority relations or resource constraints. We can call such schedules active ones [13]. The algorithm for determination of the active schedule is also detailed described in [18], [19].

## 3. SCHEDULE OPTIMIZATION ALGORITHM BASED ON SIMULATED ANNEALING AND VARIABLE NEIGHBORHOOD

### 3.1. Schedule Optimization Algorithm by the Method of Simulated Annealing

Let's consider simulated annealing (SA) algorithm based on the priority list and the serial decoding procedure [5], [18]. The main idea of algorithms such as SA, is the solutions' generation methods and special rules for accepting new randomly generated solutions [14]. New solutions are generated from the current solution environment, while calculating the values of the objective function. Usually depth (radius) of environment is decreasing through optimization procedure, starting from the fixed value. In order to move from the current solution to a new one we apply Metropolis rule, which allows, with a certain probability, to accept solutions with a worse objective function's value (accept worse solution with the probability $p = exp(\frac{T(b)-T^*(b)}{t_k})$, here $T(b)$ is current solution, $T^*(b)$ - new solution). Theoretical recommendations for choice of appropriate parameters of SA in continuous optimization, that gain the algorithm's convergence to global optimum, has been considered in [21], [21].

While constructing the SA algorithm for schedule optimization, we will consider two priority lists being neighbor if they can be obtained one from another by applying one elementary operation – shift or swap. These elementary operations with jobs in priority list are described in [19], [20].

### 3.2. Generating Random Depth of Solution Neighborhood

Typical rules for regulation of the algorithm parameters are *Neighborhood Depth* $\rho_k = \rho_0 / k^{\alpha}$, *Temperature Updating Function* $t_k = t_0 / k^{\beta}$, $0 < \alpha,\beta < 1$. We constructed special neighborhood depth generating algorithm, based on generation of stable Pareto values:

**Step 1.** Set initial $i := 1$ and $S := 0$.

**Step 2.** Generate *U1* and *U2*, uniformly distributed in [0, 1]. Then calculate

$$Z := \left( \frac{sin((\alpha - 1) \cdot U1 \cdot \pi)}{-ln(U2)} \right)^{\frac{1-a}{a}} \cdot \frac{\cdot sin(\alpha \cdot U1 \cdot \pi)}{\left( cos(\alpha \cdot \frac{\pi}{2}) \cdot sin(U1 \cdot \pi) \right)^{\frac{1}{\alpha}}}$$

($\alpha = 0.5$ or $\alpha = 0.75$)

$S := S + Z$.

**Step 3.** If $S \geq f(T)$, then $\rho_k := i$, else $i:=i+1$ and go to Step 2. Here $f(T)$ - monotonically depends on temperature.

Influence of the rules of such type on speed and exactness of convergence was discussed in the work by Yang [21].

*Let's write down the following SA algorithm:*

1) $k = 0$.

2) Let is performed $k$ steps of the algorithm. Let's state that we have the priority list $b$ where objective function value is $Z1 = T(b)$. We set the step parameters $\rho_k$ and $t_k$.

3) We randomly generate numbers $q$ and $l$, $l \neq q$, $1 \leq l, q \leq n$. With a probability $p = 0.5$, we perform whether the shift operation or the task swap operation.

4) If the priority list obtained by such way is not feasible, then we repeat the step (3) until we obtain the feasible list.

5) We repeat the steps (3) and (4) for $\rho_k$ times. Let's indicate the list obtained after performance of $\rho_k$ feasible elementary operations by $b'$.

6) We calculate the objective function $Z2 = T(b')$ for the priority list by applying the serial decoding procedure.

7) The priority list is being changed according to the Metropolis law:

if $\eta < exp((Z1-Z2)/t_k)$, then $b' = b$. $k = k+1$. If $k < k_{max}$, then we repeat the step (2).

## 4. PROJECT SCHEDULING PROBLEM LIBRARY PSPLIB AND APPLICATION FOR DESIGN

This library [15], [16] contains different problem sets for various types of resource constrained project scheduling problems as well as optimal and heuristic solutions. The data sets may be used for the evaluation of solution procedures for single- and multi-mode resource-constrained project scheduling problems. The instances have been generated by the standard project generator ProGen. Researchers may download the benchmark sets to evaluate their algorithms, and they may send their results to be added to the library. The main parameters are given in the following sections.

Kinds of solutions and instance sets, parameter settings, characterization of the benchmark instances are detailed described in [19], [20], [16]. There are a lot of RCPSP instances (input data sets, best known solutions) in the PSPLIB. More precisely, 480 samples for every problem with n=30, 60 and 90 jobs, and 600 samples for n=120.

Benchmark Instances from PSPLib includes this project information:

Number of jobs or tasks (including dummy) – 32, 62, 92, 122.

Horizon (total sum of all jobs' durations).

Number of renewable resources (typically 4). Also can be nonrenewable, doubly constrained.

Other project information – release date, due-date and other.

Precedence relations – number of successors and list of successors (jobs' numbers) for each job.

Jobs' durations and resource requests for each job.

Resource availabilities for each kind of them.

These instances were used for testing efficiency of the algorithms and such kind of projects we can usually meet in design. Of course, real-life single problem (big design project) can include thousands of jobs and their precedence relations, a lot of constrained resources. Testing algorithms' efficiency with different problem sets from PSPLib can make

us sure that constructed algorithms will be able to find good solution for any other similar problem with another initial data.

## 5. COMPUTATIONAL RESULTS

### 5.1. High Performance Computing

RCPSP are NP-complete and the solving is hard limited by available computer resources. Testing on a personal computer enabled us to solve tasks of the middle size, when dimensionality of real-life problems can exceed hundred or thousands tasks. Solving of large problems require the great number of iterations for each instance (5000-30000). Thus, the parallelization of the algorithm and computing in a high performance computer centre are necessary. A lot of applications show that heuristic methods inherently suits for parallelization and can easily be implemented in robust computer code. The parallelization strategy consists of sharing the computation of identical instances of problems among processors and using control processor to manage the adaptation of the sample size.

The developed heuristic optimization algorithms are not able to solve real life problems with hundreds and thousands variables in a reasonable computer time. Since the number of iterations and combinations of algorithms' parameters needed to solve the problem by our approach with admissible accuracy are very growing, it was very useful to test our algorithms on parallel high performance machine. We wrote the parallel code for our algorithms using MPI. Searching solution for each instance data set from PSPLib were independent processes, so it was not difficult to run benchmark tests for all instances from PSPLib with the large number of iterations on different processors. These benchmark tests were performed in High Performance Computing centre CINECA, Italia. Technical details of this parallel machine - Model: IBM CLX/1024; Architecture: IBM Linux Cluster 1350; Processor Type: Intel Xeon Pentium IV, 3 GHz, 512 KB cache; Number of Processors: 1024; Nodes: 512 (2 proc. on node); RAM: 1 TB (2 GB on node); Disk Space: 5.5 TB; Operating System: Linux SuSE SLES 8; Peak Performance: 6.1 Tflop/s; Available compilers: Intel (F90,F77,C,C++), PGI (F90,F77,C,C++), GNU (F77,C,C++); Parallel libraries: MPI (mpich/gm); Queuing system: LSF.

### 5.2. Computational Results of the simulated annealing with variable neighborhood

The created algorithm of Simulated Annealing was explored by applying the method of statistical modeling while using data sets and known solutions from the library PSPLib. Comparison of testing results of the developed algorithm show that the developed algorithm allows effective solving of RCPS problems (Table 1). The testing results show that efficiency of the algorithm can be increased by appropriately regulating environment depth and the parameter of Pareto models.

**Table 1.** Testing results of the simulated annealing with variable neighborhood. With $n = 30, 60, 120$, the tasks were performed for 1000, 5000, 10000 iterations each.

| | **1** | **2** | **3** |
|---|---|---|---|
| n = 30 | | | |
| it = 1000 | 448 of 480 | 93.33 | 0.21 |
| it = 5000 | 476 of 480 | 99.17 | 0.08 |
| *** it = 5000** | **480 of 480** | **100.00** | **0.00** |
| n = 60 | | | |
| it = 1000 | 128 of 480 | 26.67 | 4.42 |
| it = 5000 | 376 of 480 | 78.33 | 2.46 |
| *** it = 5000** | **480 of 480** | **100.00** | **0.00** |
| n = 120 | | | |
| it = 1000 | 1 of 600 | 0.17 | 18.27 |
| it = 5000 | 52 of 600 | 10.83 | 7.62 |
| *** it = 5000** | **511 of 600** | **85.17** | **2.34** |
| *** it = 10000** | **587 of 600** | **97.83** | **0.81** |

Table headings: 1 - Number of instances, when optimum or best known objective function value was found. 2 - Percentage of finding optimum or best known objective function value, %. 3- Average deviation from optimum or best known objective function value, %.

**Remark:** with "star" (*) are marked benchmark tests' results from high performance computing system. Also is given comparison to the results of early benchmark tests described with details in [4].

## 6. CONCLUSIONS

SA algorithm with Variable Neighborhood for optimizing schedules of jobs with resource constraints, based on schedule coding with the job priority list and the serial priority list decoding procedure were discussed in this paper

Minimization of the objective function, while applying several dynamic environments' depth adjustment methods, showed that change of solution environment's depth allows us to accelerate convergence to global optimum.

High performance computing using parallel algorithms lets us use very large numbers of iterations for problem's instances and in such way during acceptable calculation time, ensure us about efficiency of created algorithms.

While solving classes of various tasks, the developed algorithm proved his advantage at increased the number of iterations. In a case of separate practical tasks when there is no need to statistically compare a big amount of problems solutions, it is possible to increase the number of iterations again in order to find solutions as close to the optimum as possible. With increasing number of iteration convergence to optimum or best known objective function value can be achieved with high probability.

## REFERENCES


[1] Kelley, J. E., Jr. & Walker, M. R. (1959) Critical-path planning and scheduling, *Proceedings of Eastern Joint Computer Conference, Boston MA*, 160-173.

[2] Kelley, J. E., Jr. (1961) Critical-Path Planning and Scheduling: Mathematical Basis, *Operations Research*, Vol. 9, No. 3, 296-320.

[3] Lassaigne, R. & Rougemont, M., de (1999) Logic and Algorithms' Complexity, *Zara, Vilnius* (in Lithuanian).

[4] Savage, C. (1997) A Survey of Combinatorial Gray Codes, *Society of Industrial and Applied Mathematics Review* 39, 605-629.

[5] Brucker, P. & Knust, S. & Schoo, A. & Thiele O. (1998) A branch and bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research*, Vol. 107, No. 2, 272-288.

[6] Kolisch, R. (1996a) Efficient priority rules for the resource-constrained project scheduling problem, *Journal of Operations Management*, No.14, 179-192.

[7] Glibovec, N. N. & Medvidj, S. A. (2003) Genetic algorithms and their application to project scheduling problem solving, *Cybernetics and system analysis*, No.1, 95-108 (in Russian).

[8] Goldberg, D. E. (1989) Genetic Algorithm in Search, Optimization and Machine Learning, *Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.*

[9] Hoos, H. H. & Stutzle, Th. (2004) Stochastic Local Search. Foundations and Applications. Morgan Kaufmann / Elsevier.

[10] Kolisch, R. & Hartmann, S. (1999) Project scheduling: Recent models, algorithms and applications, *Kluwer, Amsterdam*, 147-178.

[11] Voss, St. (2001) Meta-heuristics: The State of the Art. Local Search for Planning and Scheduling, *LNAI* 2148, 1-23.

[12] Hansen, P. & Mladenovic, N. (1999) An introduction to variable neighborhood search, *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, 433-458, Kluwer.

[13] Kocetov, J. A. & Stoliar, A. A. (2003) Application of alternating environments to approximate solving of RCPSP, *Discrete analysis and operation research*, ser. 2, vol. 10, No 2, 29-55 (in Russian).

[14] Kirkpatrick, S. & Gelatt Jr., C. D. & Vecchi, M. P. (1983) Optimization by simulated annealing, *Science*, 220, 671-680.

[15] Kolisch, R. (1996c) PSPLIB - A project scheduling library, http://www.bwl.uni-kiel.de/Prod/psplib/

[16] Kolisch, R. & Sprecher, A. (1996) PSPLIB - A project scheduling library, *European Journal of Operational Research*, Vol. 96, 205-216.

[17] Klein, R. (2000) Scheduling of Resource-Constrained Projects, *Kluwer Academic Publishers*.

[18] Kolisch, R. (1996b) Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Op. Research*, Vol. 90, No. 2, 320-333.

[19] Felinskas, G. & Sakalauskas, L. (2006) Optimization of resource constrained project schedules by simulated annealing and variable neighborhood search, *Technological and economic development of economy*, Vol. XII, No. 4., 307-313.

[20] Felinskas, G. (2007) Investigation of Heuristic methods and application to optimization of resource constrained project schedules, Doctoral Dissertation, Vilnius.

[21] Felinskas, G. & Sakalauskas, L. (2003) Pareto type models in simulated annealing algorithms, *Lithuanian Mathematical Journal*, Vol.43, special volume, 573-578 (in Lithuanian).

[22] Yang, R. L. (2000) Convergence of the Simulated Annealing Algorithm for Continuous Global Optimization, *Journal Of Optimization Theory And Applications*, Vol. 104, No. 3, 691-716.