SYM-OP-IS 2009
36th Symposium on Operational Research
Mathematical Institute SANU - Belgrade

VNS FOR SCHEDULING INDEPENDENT TASKS TO IDENTICAL PROCESSORS

TATJANA DAVIDOVIĆ, STEFANA JANIĆIJEVIĆ
Mathematical Institute of Serbian Academy of Sciences and Arts, Belgrade, {tanjad,stefana}@mi.sanu.ac.rs

Abstract: *A heuristic approach, based on the Variable Neighborhood Search (VNS) meta-heuristic, to static scheduling of independent tasks to homogeneous multiprocessor systems (consisting of identical processors) is proposed. A mixed integer programming formulation is used within CPLEX optimization software to obtain optimal solution for the small size test instances. Preliminary experimental results show that using VNS optimal solutions are obtained within reasonable computational time.*

Keywords: *Static Scheduling, Independent Tasks, Homogeneous Multiprocessors, Variable Neighborhood Search.*

1. INTRODUCTION

In this paper we consider a well known problem of scheduling n independent tasks to homogeneous multiprocessor systems containing p identical processors. Although it is the simplest variant of scheduling problem it has been proven that it is NP-complete in the strong sense [1,7]. There are several exact algorithms in the literature with the goal to solve small to medium size problems to optimality [6,13]. A lot of approximation algorithms (heuristic and meta-heuristic ones) for solving this problem have been proposed (for example, [2,8,11.14]). Constructive heuristic implementing "largest processing time first" rule (LPT) proposed in [8] consists in two steps: i) tasks are sorted in decreasing order of their processing times and then ii) each task is assigned to the least loaded processor (ties are broken by the minimal index of task and/or processor). MultiFit algorithm [2] uses the fact that this scheduling problem can be formulated as the bin packing problem and produces better results than the LPT algorithm but with increasing the computational complexity. The Tabu Search approach has been proposed in [14].

In this paper we implement Variable Neighborhood Search (VNS) heuristic proposed by MladenoviĆ and Hansen [12] which has already been successfully applied to a variety of combinatorial and continuous optimization problems [9,10]. Among scheduling problems, VNS was applied to a Multiprocessor Scheduling Problem with Communication Delays [4].

The paper is organized as follows. Problem description is given in the next section. We present a mixed integer programming formulation which is used to obtain optimal solution for small examples. The outline of the VNS heuristic is given in the Section 3, while the implementation details connected to the scheduling problem are described in the Section 4. Section 5 contains experimental evaluation of our approach and Section 6 concludes the paper.

2. PROBLEM FORMLATION

The problem of scheduling independent tasks onto homogeneous multiprocessor systems consists in following: Given are a set of tasks $T = \{1,2,...,n\}$ and a set of identical processors $P = \{1,2,..., p\}$. We assume that the number of tasks n and number of processors p are input parameters. In addition, for all tasks, processing times l_i, i=1,é ,n are given in advance. Since all input data are known prior to the start of scheduling process, this variant of the scheduling problem is called *static*. The goal of the scheduling process is to assign all tasks to processors in such a way as to minimize the total completion time of all tasks (also referred to as *makespan*). Each task should be executed by a single processor without preemption and each processor can execute only one task at a time. Therefore the running time of each processor is determined as a sum of processing times of all tasks assigned to that particular processor. Makespan equals to the maximum over all processor's running times.

The considered scheduling problem could be graphically represented by Gantt diagram (Figure 1).

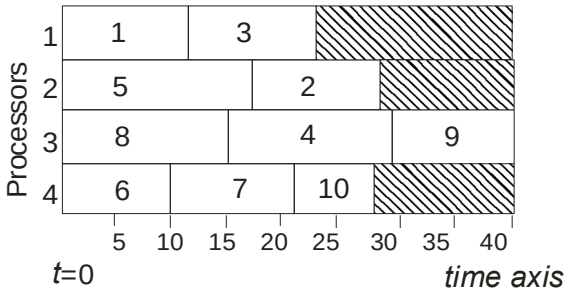

Figure 1: Gantt diagram of a schedule example

The horizontal axis in the diagram represents time. The rectangulars in the Gantt diagram represent tasks. The starting time of a task is determined by the completion times of all tasks already scheduled to the same processor. The total completion time (makespan) for the schedule shown in the Figure 1 equals 40 time units (the completion time of task 9 scheduled to processor 3). Any schedule that has completion time less than 40 time units is considered better. The goal is to discover the schedule of tasks to processors that has shortest completion time.

In order to present mathematical programming formulation of the problem, let us introduce the binary variables defined in the following way:

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is scheduled to processor } j, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The considered scheduling problem is formulated in the following way [13]:

Minimize

$$y \quad (2)$$

subject to

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad 1 \le i \le n \quad (3)$$

$$y - \sum_{i=1}^{n} l_i x_{ij} \ge 0 \qquad 1 \le j \le m \quad (4)$$

$$x_{ij} \in \{0,1\}, \qquad 1 \le i \le n, 1 \le j \le m \quad (5)$$

The objective function that should be minimized represents the total completion time of all tasks ï makespan y. Each task i should be scheduled to one and only one potential processor j (constraint (3)). The makespan y is computed as the maximum over all processorôs computation times, and processing time of a processor is defined as a sum of processing times of all tasks scheduled to that processor. This is described by the constraints (4). Constraint (5) shows binary nature of the variables x_{ij}.

This formulation is used within ILOG AMPL and CPLEX 11.2 optimization software to solve to optimality used test instances.

3. VARIABLE NEIGHBORHOOD SEARCH

In this section we give a brief overview of Variable Neighborhood Search (VNS) [9,10,12] meta-heuristic. VNS is designed for various combinatorial optimization problems. It can be described as follows. For a given optimization problem min $f(x)$, we first define the set of *solutions* S and the set of *feasible solutions* $X \subseteq S$. Let $x \in X$ be an arbitrary solution and $\mathcal{N}_k$, (k =1,é ,k_{max}), a finite set of pre-selected neighborhood structures. Then $\mathcal{N}_k(x)$ is the set of solutions in the k^{th} neighborhood of x. Steps in the basic VNS are:

Initialization. Find an initial solution $x \in X$; choose a stopping condition.

Repeat until the stopping condition is met:

1. Set $k = 1$;
2. Until $k = k_{max}$ repeat the following steps:
   a. Shaking. Generate a point x' at random from the k-th neighborhood of x, ($x' \in \mathcal{N}_k(x)$);
   b. Local search. Apply some local search method with x' as initial solution; denote with x'' the obtained local optimum;
   c. Move or not. If this local optimum is better than the incumbent, move there ($x = x''$), and continue the search with $\mathcal{N}_1$ ($k = 1$); otherwise, set $k = k+1$.

Usually, the initial solution is determined by some constructive scheduling heuristic and then improved by LS before the beginning of actual VNS procedure.

The stopping condition may be e.g. maximum *cpu* time allowed, maximum number of iterations, or maximum number of iterations between two improvements. Often successive neighborhoods $\mathcal{N}_k$, are nested, but it is not necessary to be always the case. Observe that point x' is generated at random in step 2a in order to avoid cycling, which might occur if any deterministic rule was used.

As a local optimum within some neighborhood is not necessarily one within another, change of neighborhoods can be performed during the local search phase too. This local search is then called Variable Neighborhood Descent (VND).

Basic VNS is very simple meta-heuristic and its only parameter is k_{max} - the preselected number of neighborhoods. For each particular problem solution representation, number and order of neighborhoods, and stopping condition should be defined in such a way to assure efficient execution of the search.

4. VNS IMPLEMENTATION DETAILS

In our implementation solution is defined as a ñdynamicò matrix $S_{p \times n}$ where the element s_{ji} represents the i-th task scheduled to a processor j. More precisely, for each processor, the corresponding row of the matrix S contains list of tasks scheduled to that processor. Since not all elements of the matrix S are relevant, we introduced an array o_j, j=1,é ,p, containing the number of tasks scheduled to processor j. The last part of the solution is array y_j, j=1,é ,p, whose elements represent running times of processors. Namely, y_j equals to the sum of execution times of all tasks scheduled to processor j.

We used two types of neighborhoods: *shift* and *interchange*. Shift neighborhood is realized by moving a task from one processor to another. This means that the complexity of the shift neighborhood is $O(np)$, more

precisely, each task (n) is transferred to any of the remaining (p-1) processors. Interchange neighborhood means that tasks from different processors exchange their positions. The worst case complexity of the interchange neighborhood is $O(n^2)$ since each pair of tasks (not scheduled onto same processor) can be considered for the position exchange.

In order to make our implementation more efficient, we used reduced neighborhoods. Namely, shift neighborhood moves only tasks from the most loaded processor to the least loaded one. Within the interchange neighborhood, only tasks from the most loaded and the least loaded processors exchange their positions. Moreover, since the order of tasks is not relevant in the case of independent tasks scheduling, moved tasks are placed at the end of the list of tasks for a given processor while a hole for task taken away from a processor is filled in with the last task allocated to that processor. These assumptions make easy to update data structures at each step of a local search move. Namely, for each neighborhood, the number of transformations of one solution to another one is constant, i.e. the complexity of a move from one solution to the other is O(1). For example, in shift neighborhood, to move task i (being the r-th one on the corresponding processor) from processor j to processor m, one should perform the following steps (we use C-like notation):

1. $s[m][o[m]++] = i$;
2. $s[j][r] = s[j][--o[j]]$;
3. $y[m] += l[i]$;
4. $y[j] -= l[i]$;

Finally, the makespan to be minimized is maximum y_j overall j.

5. EXPERIMENTAL EVALUATION

Our heuristic implementation is tested on small size examples with known optimal solutions. Optimal solutions were obtained by using ILOG AMPL and CPLEX 11.2 optimization software. Both programs, CPLEX and our VNS are running on Intel Core 2 Duo CPU E6750 on 2.66GHz with RAM=8 Gb under Linux Slackware 12, Kernel: 2.6.21.5, gcc version 4.1.2 Therefore we were able to compare execution times and solution quality for these programs. Test examples are generated using the random task graph generators proposed in [3]. Actually, we used examples generated for the more complex problem (multiprocessor scheduling problem with communication delays) and neglected task dependencies and communication times.

Experimental results are summarized in Table 1. The first column of Table 1 contains the name of an example file. Number of tasks is a part of the file name, while number of processors is set to 4 in all examples. In the second and third columns we present the optimal schedule length and CPU time required by CPLEX to find it, respectively. Column four contains the schedule length obtained by VNS. Corresponding CPU time (needed by VNS to find its best schedule) is given in the last column.

Table 1: Scheduling results for small task graphs

Example	opt	opt CPU time	VNS	VNS CPU time
It50_70	212	0.021	212	0.001
It50_80	196	0.028	196	0.001
It50_80_1	234	0.044	234	0.001
It50_80_2	337	0.020	337	0.001
It50_80_3	216	0.026	216	0.001
It50_80_4	278	0.018	278	0.001
It50_80_5	128	0.007	128	0.001
It50_80_6	167	0.023	167	0.001
It100_40_1	493	0.064	493	0.001
It100_40_2	782	0.111	782	0.001
It100_40_3	478	0.096	478	0.001
It100_40_4	483	0.075	483	0.001
It100_40_5	271	0.036	271	0.001
It100_40_6	340	0.011	340	0.001
It100_50_1	471	0.098	471	0.001
It100_60_1	465	0.010	465	0.001

As can be seen from Table 1, VNS was able to obtain optimal solution for all test instances within significantly smaller CPU time. Stopping criteria for the VNS was set to 1 *sec* of CPU time.

We compared the performance of VNS heuristic with the previously implemented Monte Carlo (MC) method [5]. The stopping criterion for MC was 10000 iterations requiring less than 1 *sec* to be completed. The results are presented in Table 2.

Table 2: Comparison of VNS and MC

Example	MC	MC CPU time	VNS	VNS CPU time
It50_70	212	0.007	212	0.001
It50_80	196	0.001	196	0.001
It50_80_1	234	0.001	234	0.001
It50_80_2	337	0.007	337	0.001
It50_80_3	216	0.000	216	0.001
It50_80_4	278	0.029	278	0.001
It50_80_5	128	0.000	128	0.001
It50_80_6	167	0.001	167	0.001
It100_40_1	493	0.006	493	0.001
It100_40_2	783	0.020	782	0.001
It100_40_3	478	0.028	478	0.001
It100_40_4	483	0.001	483	0.001
It100_40_5	271	0.000	271	0.001
It100_40_6	340	0.047	340	0.001
It100_50_1	471	0.001	471	0.001
It100_60_1	465	0.001	465	0.001

Although MC was also able to obtain optimal solution for most of the instances, it required several times longer execution time then VNS (except in several cases).

6. CONCLUSION

In this paper we present preliminary results of applying Variable neighborhood Search (VNS) heuristic to the problem of scheduling independent tasks onto homogeneous multiprocessor system. The obtained results

are very encouraging for continuing research on this topic. Possible extensions are related to testing larger instances, comparing with other heuristic methods, like Genetic Algorithms (GA), Tabu Search (TS), Multistart Local Search (MLS), and introducing new neighborhood structures.

REFERENCES

[1] Brucker, P., *Scheduling Algorithms*, Springer, 1998.

[2] Coffman Jr, E. G., Garey, M. R., Johnson, D. S. ñAn application of bin-packing to multiprocessor schedulingò, *SIAM J. Comput.*, 7 (1978), 1-17.

[3] DavidoviÍ, T., Crainic, T. G., ñBenchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systemsò, *Comput. & OR*, 33(8) (2006), 2155-2177.

[4] DavidoviÍ, T., Hansen, P., MladenoviÍ, N., ñPermutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delaysò, *Asia-pacific Journal of Operational Research*, 22(3) (2005), 297-326.

[5] DavidoviÍ, T., JaniÍijeviÍ, S., ñHeuristic Approach to Scheduling Independent Tasks on Identical Processorsò, in *Proc. Symp. on information technology, YUINFO 2008*, (on CD 115.pdf), Kopaonik, March 08-11, 2009.

[6] Dell'Amico, M., Martello, S., ñOptimal scheduling of tasks on identical parallel processorsò, *ORSA Journal on Computing*, 7 (1995), 191-200.

[7] Garey, M. R., Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[8] Graham, R. L., ñBounds on multiprocessor timing anomaliesò, *SIAM J. Applied Math.*, 17 (1969), 416-429.

[9] Hansen, P. MladenoviÍ, N., ñVariable neighbourhood searchò, in Glover, F., Kochenagen, G., editors, *Handbook of Metaheuristics*, pages 145-184. Kluwer Academic Publishers, Dordrecht, 2003.

[10] Hansen, P., MladenoviÍ, N., ñVariable neighbourhood searchò, in Burke, E. K., Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 211-238. Springer, 2005.

[11] Haouari, M., Gharbi, A., Jemmali, J. ñTight bounds for the identical parallel machine scheduling problemò, *Int. Trans. in Oper. Res.*, 13 (2006), 529-548.

[12] MladenoviÍ, N., Hansen, P., ñVariable neighborhood searchò, *Comput. & OR*, 24(11) (1997), 1097-1100.

[13] Mokotoff, E, ñAn exact algorithm for the identical parallel machine scheduling problemò, *Europ. J. Oper. Res.*, 152 (2004), 758-769.

[14] Thesen, A., ñDesign and evaluation of a tabu search algorithm for multiprocessor schedulingò, *J. Heur.*, 4(2) (1998), 141-160.