Proceedings of the World Congress on Engineering 2014 Vol II,
WCE 2014, July 2 - 4, 2014, London, U.K.

Branch and Bound Method for Scheduling Precedence Constrained Tasks on Parallel Identical Processors

N.S.Grigoreva *

Abstract—**The multiprocessor scheduling problem is one of the classic NP-hard optimization problems. This paper deals with the problem of scheduling tasks to parallel processors with the goal of minimizing execution time. The branch and bound algorithm produces a feasible IIT(inserted idle time) schedule for a fixed length T. In order to optimize over T we must iterate the scheduling process over possible values of T. The upper and lower bound on T is defined. New dominance criteria are introduced to curtail the enumeration tree. By this approach it is generally possible to eliminate most of the useless nodes generated at the lowest levels of decision tree. To illustrate the efficiency of our approach we tested it on randomly generated task graphs.**

Keywords: multiprocessor scheduling, parallel processors, Branch and Bound algorithm, IIT (inserted idle time), task graph

1 Introduction

The problem of minimizing the makespan while scheduling tasks to parallel identical processors is a classical combinatorial optimization problem. Following the 3-field classification scheme proposed by Graham *et al.* [1], this problem is denoted by $P|prec|C_{max}$. Branch and bound algorithm is presented for the scheduling problem with tasks have to be executed on several parallel identical processors. This problem relates to the scheduling problem [2,3],it has many applications, and it is NP-hard [4]. Branch and bound method [5,6] allows to obtain an exact solution or, when the number of iterations is restricted, a fairly good approximation of the solution. A new method for evaluating partial solutions, selecting the next task and new ways of reducing the exhaustive search was designed. To illustrate the effectiveness of this approach we tested it on randomly generated task graphs.

We consider a system of tasks $U = \{u_1 u_2, \ldots, u_n\}$. The execution time of each task $t(u_i)$ is known. Precedence constructions between tasks are represented by a directed acyclic task graph $G = \langle U, E \rangle$. E is a set of directed arcs, an arc $e = (u_i, u_j) \in E$ if and only if $u_i \prec u_j$. The expression $u_i \prec u_j$ means that the task u_j may be initiated only after completion of the task u_i. Set of tasks is performed on parallel identical processors, any task can run on any processor and each processor can perform no more than one task at a time. Task preemption is not allowed. The usual objective function is completion time of the scheduled task graph also referred to as makespan or schedule length.

For this model, we can consider two problems. In the first problem the number of processors m is known and the goal is to minimize the execution time - makespan. In the second one the makespan is known, the goal is to minimize the number of processors. To solve these two problems, we can apply the same algorithm.

Subtask of these two problems is the task of constructing a feasible schedule for the given number of processors and the given execution time. A schedule for a task set U is the mapping of each task $u_i \in U$ to a start time $\tau(u_i)$ and a processor $num(u_i)$. Length of schedule S is the quantity

$$T_S = \max\{\tau(u_i) + t(u_i) | u_i \in U\}.$$

To solve this problem we apply the branch and bound method in conjunction with binary search. Branch and bound method constructs a feasible schedule S of length T for m processors. The algorithm may be used in a binary search mode to find the smallest number of processors or the smallest makespan. First we propose an approximate IIT algorithm named CP/IIT (critical path/ inserted idle time). Then by combining the CP/IIT algorithm and $B\&B$ method this paper presents BB/IIT algorithm which can find optimal solutions to parallel scheduling problem.

2 Approximate algorithm

A lot of research in scheduling has concentrated on the construction of nondelay schedule. A nondelay schedule is a feasible schedule in which no processor is kept idle at a time when it could begin processing a task. An inserted idle time schedule (IIT) has been defined by J.Kanet and

*Manuscript accepted March, 21,2014. N.S.Grigoreva is with St.Petersburg State University, Department of Mathematics and Mechanics, St.Petersburg,Russia, Email: n.s.grig@gmail.com

ISBN: 978-988-19253-5-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

V.Sridharam [7] as a feasible schedule in which a processor is kept idle at a time when it could begin processing a task. We propose an approximate IIT algorithm named CP/IIT (critical path/ inserted idle time).

For each task u_i, we define the earliest starting time $v_{min}(u_i)$ and the latest start time $v_{max}(u_i)$. The earliest starting time is numerically equal to the length of the maximal path in the graph G from the initial vertex to the vertex u_i. The latest start time $v_{max}(u_i)$ of task u_i, is numerically equal to the difference between the length of the required schedule T_S and the length of the maximal path from the task u_i, to the final vertex. The latest start time $v_{max}(u_i)$ of task u_i is a priority of task. Let k tasks are put in the schedule and partial schedule S_k is constructed.

Let be $time_k[i]$ the time of the termination of the processor i after completion all its tasks. The approximate schedule is constructed by CP/IIT algorithm as follows:

1. Determine the processor l_0 such as $t_{\min}(l_0) = \min\{time_k[i] | i \in 1..m\}$
2. Select the task u_0, such as all its predecessors are included in the schedule and $v_{max}(u_0) = \min\{v_{max}(u_i) | u_i \notin S_k\}$
3. If $r_0 = v_{min}(u_0) - t_{min}(l_0) > 0$ then choose a task $u^* \notin S_k$, which can be executed during the idle time of processor without increasing the start time of the task u_0, namely $v_{min}(u^*) + t(u^*) \leq v_{min}(u_0)$.
4. If the task u^* is found, then we assign the task u^* to the processor l_0 otherwise we assign the task u_0 to the processor l_0.

In order to examine the effectiveness of CP/IIT algorithm we tested it on randomly generated task graph. The results are shown in Table 1.

3 Branch and bound method for constructing a feasible schedule $BB(U, T, m; S)$

For the formal description of the branch and bound method we must give a definition of partial solutions. It is convenient to represent the schedule as a permutation of jobs. For each permutation of tasks $\pi = (u_{i_1}, u_{i_2}, \ldots, u_{i_n})$, one can construct a schedule S_π as follows: to find the earliest time of the release of processors t_{min}, to find the processor that was released at this time, then the task is assigned to the processor at the earliest possible time, but not before t_{min}, then every permutation will uniquely identify the schedule S_π. Partial solution σ_k, where k the number of jobs will be regarded as a partial permutation $\sigma_k = (u_{i_1}, u_{i_2}, \ldots, u_{i_k})$, which is determined partial schedule.

Definition 1 *The solution $\gamma_n = (l_1, l_2, \ldots, l_n)$ is called the extension of partial solutions $\sigma_k = (q_1, q_2, \ldots, q_k)$, if $l_1 = q_1, l_2 = q_2, \ldots, l_k = q_k$.*

Definition 2 *A partial solution σ_k is called a feasible if there exists an extension of σ_k, which is a feasible schedule.*

For each task u_i, we define the earliest starting time $v_{min}(u_i)$ and the latest start time $v_{max}(u_i)$. In order to make the feasible schedule, it is necessary that each task $u_i \in U$, the start time of its execution $\tau(u_i)$ satisfies the inequality

$$v_{min}(u_i) \leq \tau(u_i) \leq v_{max}(u_i).$$

In order to describe the branch and bound method it is necessary to determine the set of tasks that we need to add to a partial solution, the order in which task will be chosen from this set and the rules that will be used for eliminating partial solutions.

3.1 Selection of task

Let I be the total idle time of processors in a feasible schedule S of length T_S for m processors, then $I = m \cdot T_S - \sum_{i=1}^{n} t(u_i)$. For a partial solution σ_k we know $r(u_i)$— idle time of processor before start the task u_i. Let I_k be the remaining pool of idle for a partial solution σ_k. Then $I_k = I - \sum_{i=1}^{k} r(u_i)$. We know the completion time of processors $time_k[1 : m]$. Denote $t_{min}(k) = \min\{time_k[i] | i \in 1 : m\}$ then $t_{min}(k)$ is the earliest time of ending all tasks were included in a partial solution σ_k.

At each level k will be allocated a set of tasks U_k, which we call the possible assignments. These are tasks that need to add to a partial solution σ_{k-1}, so check all the possible continuation of the partial solutions.

Definition 3 *Task $u \notin \sigma_k$ is called the ready task at the level k, if all its predecessors were included in the partial solution σ_k and the earliest starting time $v_{min}(u)$ satisfies the inequality $v_{min}(u) - t_{min}(k) \leq I_k$.*

Task selection procedure $Select(U_k, t_{min}(k); u_0)$

From the set U_k we choose task u_0 with the minimum latest start time. If before beginning this task processor will be idle we are trying to find a task that can be performed during idle time of processor.

1. Select the task u_0, such as $v_{max}(u_0) = \min\{v_{max}(u_i) | \, u_i \in U_k\}$.
2. If $r_0 = v_{min}(u_0) - t_{min}(k) > 0$ then choose a task $u^* \in U_k$, which can be executed during the idle time of processor without increasing the start time of the task u_0 namely, $v_{min}(u^*) + t(u^*) \leq v_{min}(u_0)$.

ISBN: 978-988-19253-5-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

3. If the task u^* is found, then we assign the task u^* to the processor otherwise we assign the task u_0 to the same processor.

3.2 Deleting of invalid partial solutions

The main way of reducing of the exhaustive search will be the earliest possible identification unfeasible solutions.

Definition 4 *Let the task* $u_{cr} \notin \sigma_k$ *is such as* $v_{max}(u_{cr}) = \min\{v_{max}(u) | u \notin \sigma_k\}$. *The task* $u_{cr} \notin \sigma_k$ *is called the delayed task for* σ_k, *if* $v_{max}(u_{cr}) < t_{min}(k)$.

Lemma 1 *Let delayed task* u_{cr} *for a partial solution* $\sigma_k = \sigma_{k-1} \cup u_k$ *exists, then:*

1. *The partial solution* σ_k *is unfeasible.*
2. *For any task* $u \in U_k$, *such that* $\max\{t_{min}(k-1), v_{min}(u)\} + t(u) > v_{max}(u_{cr})$ *a partial solution* $\sigma_{k-1} \cup u$ *is unfeasible.*
3. *If* $v_{max}(u_k) < t_{min}(k)$ *and* $t_{min}(k-1) + t(u_{cr}) > v_{max}(u_k)$, *then the partial solution* σ_{k-1} *is unfeasible.*

Another method for determining unfeasible partial solutions based on a comparison of resource requirements of tasks and total processing power. In this case we propose to modify the algorithm for determining the interval of concentration [8] for the complete schedule. We apply this algorithm to a partial schedule σ_k and determine its admissibility.

We consider time intervals $[t_1, t_2] \subseteq [t_{\min}(k), T_S]$. Let $MP(t_1, t_2)$ be the total time of free processors in time interval $[t_1, t_2]$ then

$$MP(t_1, t_2) = \sum_{i=1}^{m} \max\{0, (t_2 - \max\{t_1, time_k[i]\})\}.$$

For all task $u_i \notin \sigma_k$ we find minimal time of its begin: $v(u_i) = \max\{v_{\min}(u_i), t_{\min}(k)\}$. Let $L([t_1, t_2])$ be a length of time interval $[t_1, t_2]$.

Let $M_k(t_1, t_2)$ be the total minimal time of tasks in time interval $[t_1, t_2]$, then

$$M_k(t_1, t_2) = \sum_{u_i \notin \sigma_k} \min\{L(x_k(u_i)), L(y(u_i))\},$$

where

$$x_k(u_i) = [v(u_i), v(u_i) + t(u_i)] \cap [t_1, t_2],$$
$$y(u_i) = [v_{\max}(u_i), v_{\max}(u_i) + t(u_i)] \cap [t_1, t_2].$$

Let

$$est(\sigma_k) = \max_{[t_1, t_2] \in [t_{\min}(k), T_S]} \{M_k(t_1, t_2) - MP(t_1, t_2).\}$$

Lemma 2 *If* $est(\sigma_k) > 0$ *then a partial solution* σ_k *is unfeasible.*

The pseudo-code of Branch and bound method for constructing a feasible schedule $BB(U, T, m; S)$ is shown in Algorithm 1.

Algorithm 1 BB/IIT algorithm

Set $k := 1; time[i] = 0; i \in 1 : m$; $\sigma_0 = \emptyset$;
while $(k > 0)$ **and** $(k < n + 1)$ **do**
  Determine the processor l_0 such as $t_{\min}(l_0) = \min\{(time_k[i] | i \in 1..m)\}$
  Determine the task u_{cr} such as $v_{\max}(u_{cr}) = \min\{v_{\max}(u) | u \notin \sigma_{k-1}\}$;
  if $v_{\max}(u_{cr}) \leq t_{\min}(l_0)$ **then**
    Compute $EST = est(\sigma_{k-1})$;
    if $EST \leq 0$ **then**
      Select the task u_0, use procedure $Select(U_k, t_{min}(k); u_0)$
      Set the task u_0 on processor l_0 and create partial solution $\sigma_k = \sigma_{k-1} \cup u_0$
    else
      Perform step back and create partial schedule σ_{k-1}
  else
    There is delayed task u_{cr}. Delete all unfeasible partial solution by using Lemma 1
  end if
  end if
end while
if $k = 0$, **then**
  Makespan of optimal schedule is greater then T_S.
end if
if $k = n$, **then**
  We find feasible schedule $S = \sigma_n$ and its makespan is equal T_S
end if

4 Computation result

To test method and efficacy evaluations we conducted computational experiment. In the computational experiment, we intended to test the BB/IIT algorithm and to test the effectiveness of methods to remove invalid partial solutions. For branch and bound method to build a feasible solution, it had to accomplish the task in less than 60 seconds. If a feasible schedule S of length T for m processors was not received for 60 seconds, it was assumed that this does not exist and makespan T increased. This approach provided a schedule for all test

ISBN: 978-988-19253-5-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

problems, but whether or not the solutions obtained are exact or approximate remains an open question. Therefore, the quality of the solutions was estimated against to the lower estimate of the makespan LB. Lowbound LB of length optimal schedule is

$$LB = \max\{t_{cp}, \lceil \sum_{i=1}^{n} t(u_i)/m \rceil\},$$

where t_{cp} is the length critical path in task graph. To illustrate the effectiveness of our algorithm we tested it on two types graphs.

In the first group a set of task graph were generated using a random task graph generator with 50, 100,150 tasks. The processing time of the tasks has been chosen randomly from the interval [1:50]. In the second one we used Standard Task Graph Set which is available at http://www.kasahara.elec.waseda.ac.jp/schedule/.

In the first group we divided all tests on 5 series. There are 100 tests in series. We compare the length $Z0$ of initial solution (obtained by CP/IIT algorithm) and the length Z_{CP} of initial solution obtained by CP algorithm.

Comparative results are presented in Table 1. The first column of this table contains the name of series tests, in the second one m is a number of processors. The third one contains the relative error of initial solution $RZ = (Z0 - LB)/LB$ and the fourth one contains the relative error $RT = (T - LB)/LB$, where T is the length of schedule obtained by BB/IIT algorithm. Then the fifth column contains the relative error of initial solution $RCP = (Z_{CP} - LB)/LB$ and the the last one contains the relative error $RTCP = (T_{CP} - LB)/LB$ where T_{CP} is the the length of schedule obtained by B&B algorithm with select procedure CP.

Table I. Average data for any series of tests.

Series	m	RZ	RT	RCP	$RTCP$
s1	3	0.087	0.006	0.062	0.004
s2	4	0.062	0.013	0.095	0.011
s3	5	0.161	0.021	0.177	0.028
s4	5	0.128	0.057	0.248	0.143
s5	5	0.042	0.007	0.105	0.022
Average		0.096	0.021	0.137	0.042

Approximate solution with the error RZ of less then 10 percent in average were obtained by CP/IIT algorithm. The average relative error of schedules obtained by BB/IIT algorithm is equal 2 percent.

Table 2 shows the results (percent of optimal solutions found) for the first group of instances. The column N_{opt} shows the cases (in percents) where optimal schedules were obtained by BB/IIT method. The next column shows the number of cases (in percents) in which approximate solutions within the error of 0.05 were obtained, but optimal solutions could not be obtained because of CPU time limit. But an intermediate solution can be an optimal solution. The next two columns shows the number of cases in which $RT \in (0.05, 0.1]$ and RT greater then 0.1.

Table II. Results for the relative error of makespan of schedule.

Series	m	N_{opt}	$RT < 0.05$	$RT < 0.1$	$RT > 0.1$
s1	3	56	26	14	4
s2	4	70	19	11	0
s3	5	61	23	15	1
s4	5	57	29	11	0
s5	5	69	30	17	3
Average		62.6	23.4	13.6	1.6

It is seen from Table 2 that optimal solution were obtained for 63 percent (in average) of the cases tested. For 86 percent of the cases approximate solutions having error of less than 5 percent were obtained.

In the second group of tests we use tests from Standard Task Graph Set. Standard Task Graph Set is a kind of benchmark for evaluation of multiprocessor scheduling algorithms, where optimal decisions are known. We considered tests from Standard Task Graph Set with n= 50 and n=100, where n is the number of tasks. Optimal schedules were found by BB/IIT algorithm in 95 percent tests with n=50 and in 89 percent tests with n=100.

5 Conclusion

In this work we proposed a new branch and bound method for solving the multiprocessor scheduling problem of makespan minimization. We also presented a new approximate IIT (inserted idle time) algorithm. We found that the minimum execution time multiprocessor scheduling problem can be solved within reasonable time for moderate-size systems. With an increasing number of tasks, branch and bound method requires more time to obtain the optimal solution. Limiting the number of iterations seems justified and promising way to obtain a good approximate solution. Computer experiment confirmed efficiency of branch and bound method with this restriction.

References

[1] Graham,R.L.,Lawner E.L., Lenstra J.K.,and Rinnooy Kan. “Optimization and approximation in deterministic sequencing and scheduling: A survey” *Annals of Discrete Mathematics*,1979,V5,pp.287-326

ISBN: 978-988-19253-5-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[2] *Computer and job-shop scheduling theory*, Ed. by E.G.Coffman, John Wiley,1976.

[3] Brucker P. *Scheduling Algorithms*, Springer.1997.

[4] Ullman J.D. "NP-complete scheduling problems" *Journal Comput.System Sci.* 1975. 10. pp. 384–393.

[5] Kasahara H., Narita S. "Practical multiprocessor scheduling algorithms for efficient parallel processing", *IEEE Tranzactions on computers.1984*,V4. pp.22–33, No.11

[6] Grigoreva N.S., "Branch and bound algorithm for multiprocessor scheduling problem", *Vestnic SPbGU*.seria 10. 2009. Iss.1 pp.44–55.[in Russian]

[7] Kanet J.J.,Sridharan V., "Scheduling with inserted idle time: problem taxonomy and literature review",*Operation Research*.2000.V48.Iss.1. pp. 99–110.

[8] Fernandez E. and Bussell B., "Bounds the number of processors and time for multiprocessor optimal schedules",*IEEE Trans. on Computers*.1973.pp.745–751.

[9] Graham R.L., "Bounds for certain multiprocessing Anomalies",*Bell System Tecnical journal* 1966.V45.Issue 9.pp.1563–1581.

ISBN: 978-988-19253-5-0
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)