Optima Localization in Scheduling Multi-Processor Jobs

Alexander Gordeev[1,2], Alexander Kononov[1,2], and Polina Kononova[1,2]

[1] Sobolev Institute of Mathematics,
4, Akad. Koptyug avenue, 630090, Novosibirsk, Russia.
[2] Novosibirsk State University, 2, Pirogova str., 630090, Novosibirsk, Russia
agordeevw@gmail.com,
alvenko@math.nsc.ru,
polik83@gmail.com

Abstract. The paper considers a multi-processor tasks scheduling problem with dedicated processors. The main target is determining the tight optima localization intervals for different sub-problems of the basic problem. Based on the ideas of a computer-aided technique developed in the paper by Sevastyanov and Chernykh [15] for shop scheduling problems, we elaborated a similar method for a different-type problem (see above). For small values of the number of processors (from 3 to 5) a computer program coped with the main target. The extremum values of the optimum (in terms of a standard lower bound taken for 1) computed for several sub-problems of the basic problem are listed in Table 1 (page 7). In parallel with those sub-problems, so called *prime-versions* of those problems were investigated, in which only jobs requiring more than one processor are allowed. An interesting phenomenon was discovered: nearly all (but one) prime-sub-problems appeared to be polynomial-time solvable (with optima coinciding with their lower bounds – such instances and classes of instances were called in [10, 11] *normal*). As a by-product of those results, a family of linear-time approximation algorithms was designed (for the sub-problems listed in Table 1) with ratio performance guarantees taken from Table 1.

Keywords: multiprocessor jobs, branch-and-bound algorithm, optima localization

1 Introduction

We consider adjacent single mode multiprocessor jobs. Set $\mathcal{J} = \{J_1, ..., J_n\}$ of jobs and $\mathcal{M}$ of processors are given. Each job has processing time τ_j and may require more than one processor at the same time. We assume that the set of processors required by a job is given and fixed. Additionally, we assume that processors are the vertices of a given graph $G = (\mathcal{M}, E)$ and the required set of processors must induce a connected subgraph of G. By $\mu_i \subseteq \mathcal{M}$ denote the set of processors required by job J_i. Let $\mathcal{J}^A$

In: A. Kononov et al. (eds.): DOOR 2016, Vladivostok, Russia, published at http://ceur-ws.org

denote the set of jobs requiring processors in set A, i.e. $\mathcal{J}^A = \{J_i : \mu_i = A\}$. We will say that jobs in set $\mathcal{J}^A$ are of type A.

A schedule σ is an assignment of each job J_i to an interval of length τ_i, such that for any two jobs J_i and J_j with $\mu_i \cap \mu_j \neq \emptyset$, their intervals do not overlap. It is required to find a schedule which minimizes the makespan. For the given graph G we denote the above problem by $\Pi(G)$; OPT stands for the optimum.

For any G, $\Pi(G)$ is a special case of $P|fix_j|C_{\max}$ [4, 7] The difference in the two problems is that an instance of $P|fix_j|C_{\max}$ may contain a job of any type, while an instance of $\Pi(G)$ contains only the jobs with the adjacent set of processors in G. In turn, if G is a complete graph K_m with m vertices, $\Pi(G)$ is the same as $P|fix_j|C_{\max}$ with m processors. Restrictions on the allowable types of jobs are associated with the location of resources in space. The requirement for contiguity of resources occurs in parallel computing using certain network topologies [7], in warehouse management[5, 9], in the domain of harbor logistic [2, 12, 14] and other applications[8, 13]. We note that $P|fix_j|C_{\max}$ is known more than fourty five years and was considered in many scheduling papers including a section in textbook "Scheduling for Parallel Processing" by Drozdowski [7].

However, the problem Π has been given much less attention. Duin and van der Sluis [9] consider the problem Π when the given graph G is a path. They call this problem as ARS-R (Adjacent Resource Scheduling). Duin and van der Sluis presented a polynomial time exact algorithm for the case of three processors and proved NP-hardness for the case of four processors. They also observe that ARS-R is solvable in linear time if each job requires at most two processors. This result can be easily extended for the case when the given graph G is a circuit with even number of vertices. Indeed, let m processors are numbered consecutively. We divide the set of all jobs into three sets $\mathcal{J}_1$, $\mathcal{J}_2$ and $\mathcal{J}_3$ such that $\mathcal{J}_2$ contains all single-processor jobs, $\mathcal{J}_1$ contain all jobs of type $\{m+1, 1\}$ and all jobs of type $\{i, i+1\}$, where i is even, and $\mathcal{J}_3$ contain the remaining jobs. We note that jobs of different types may be executed in parallel. An optimal schedule is constructed by scheduling first all jobs in $\mathcal{J}_1$ in an arbitrary order, then all jobs in $\mathcal{J}_2$ in an arbitrary order, and, finally, all jobs in $\mathcal{J}_3$ in an arbitrary order. It is easy to check that the makespan is equal to the maximal processor load.

In our paper we present the results on optima localization in terms of lower bounds for different classes of the problem Π. Two jobs J_i and J_j are called incompatible if $\mu_i \cap \mu_j \neq \emptyset$. Incompatibility of the jobs can be represented as an incompatibility graph. Note that in the incompatibility graphs jobs are represented as nodes. The total processing time of incompatible jobs is a lower bound on OPT. Denote this lower bound by $LB(G)$. Notice that a calculation of $LB(G)$ is equivalent to determining a clique in the incompatibility graph, which is a hard combinatorial problem itself. However, for some special classes of graphs a lower bound can be found in polynomial time. For example, if G is acyclic then a lower bound is easily computable and equal to the maximal processor load.

For the given graph G we wish to find the minimal functional $\rho(G)$ such that the inequality $OPT \leq \rho(G)LB(G)$ holds for any instance of $\Pi(G)$. We say that a schedule is *primitive* if all jobs of the same type are scheduled consecutively. Dell'Olmo et al. [6] showed that $\rho(K_3) = \frac{5}{4}$. Actually, they proved that the length of the best primitive

schedule is bounded by $\frac{5}{4}LB$ and this bound is tight. It is obvious, that there exists an instance I with at most one job of each type in which $\frac{OPT(I)}{LB(G)}$ attains its maximum. Since any schedule of jobs from I is primitive, we obtained that $\rho(K_3) = \frac{5}{4}$.

Our research is unusual since we use a computer to get the theoretical results. It is based on the method proposed by Sevastianov and Tchernykh in [15]. They considered the open shop scheduling problem $O3||C_{max}$ with three processors and the makespan minimization criterion. The maximum of the maximal processor load and the maximal job length is a trivial lower bound $\tilde{C}$ for the optimal makespan. Sevastianov and Tchernykh proved that the length of the optimal schedule belongs to the interval $[\tilde{C}, \frac{4}{3}\tilde{C}]$ and this interval is tight. They also found the tight optima localization interval $[\tilde{C}, \frac{5}{3}\tilde{C}]$ for the three-processor assembly line problem. Both above results imply linear time approximation algorithms as by-product.

Our paper is organized as follows. We introduce the necessary definitions and the preliminary results in the second section. In the third section we present a branch-and-bound algorithm which calculates $\rho(G)$ and find an instance $I \in \Pi(G)$ for which $\frac{OPT(I)}{LB(J)}$ attains its maximum. In the forth section we list values of $\rho(G)$ for different graphs G obtained by our algorithm.

2 Preliminaries

Consider a graph G. Denote the set of valid job types by $\mathcal{A}(G)$. Remind that we wish to determine the value of $\rho(G)$. So we need to prove that $OPT(I) \leq \rho(G)LB$ for any instance I of $\Pi(G)$ and find an instance I' such that $OPT(I') = \rho(G)LB$.

Lemma 1. *For any graph G the supremum of $OPT(I)/LB(G)$ over all instances I, $I \in \Pi(G)$ is attained on an instance which contains at most one job of each type and does not contain jobs required all processors.*

Proof is straightforward.

In what follows in this section, we deal with instances that satisfy Lemma 1. We call such instances reduced instances.

As an example, let G be a path with four vertices, i.e. $G \equiv P_4$, see Fig. 1.

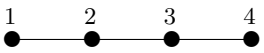

Fig. 1. G is a path with four vertices, $G \equiv P_4$

We have nine non-trivial job types:

$$\mathcal{A}(P_4) = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{2,3\}, \{3,4\}, \{1,2,3\}, \{2,3,4\}\}.$$

Thus if $G \equiv P_4$ each reduced instance contains at most nine jobs. Without loss of generality we assume that each instance has exactly one job of each valid type. It follows that an incompatibility graph H is the same for all instances of $\Pi(G)$. Let $A \in \mathcal{A}(G)$

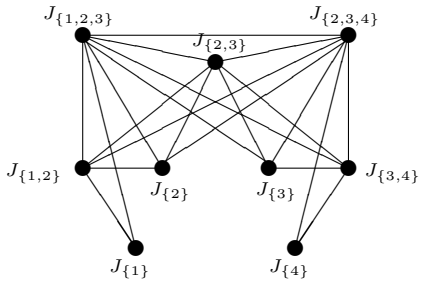

Fig. 2. The incompability graph H

we denote by J_A and τ_A the job of type A and its processing time, correspondingly. The incompatibility graph for $\Pi(P_4)$ is shown in Fig. 2.

Let $Q \subseteq H$ be a maximal clique in H, then $\sum_{J_A \in V(Q)} \tau_A$ is a lower bound on OPT, where $V(Q)$ is the set of vertices of Q. Generally speaking the number of maximal cliques grows exponentially in the number of vertices for most graphs. However in our research we consider graphs with a few number of vertices and we can find all maximal cliques in reasonable time. We use the algorithm by Tanaka et al. [16] to generate all maximal cliques of a graph. Moreover, if G is acyclic graph on m vertices, the corresponding incompatibility graph H has exactly m maximal cliques that determine m lower bounds on OPT. For example, if $G \equiv P_4$ we get the following lower bounds:

$$OPT \geq l_1 = \tau_{\{1\}} + \tau_{\{1,2\}} + \tau_{\{1,2,3\}},$$

$$OPT \geq l_2 = \tau_{\{2\}} + \tau_{\{1,2\}} + \tau_{\{2,3\}} + \tau_{\{1,2,3\}} + \tau_{\{2,3,4\}},$$

$$OPT \geq l_3 = \tau_{\{3\}} + \tau_{\{2,3\}} + \tau_{\{3,4\}} + \tau_{\{1,2,3\}} + \tau_{\{2,3,4\}},$$

$$OPT \geq l_4 = \tau_{\{4\}} + \tau_{\{3,4\}} + \tau_{\{2,3,4\}}.$$

Thus, $LB(P_4) = \max\{l_1, l_2, l_3, l_4\}$. At this point we assume that $LB(G) = 1$ and measure all job processing times in these units.

We define the weight of each vertex J_A in the incompatibility graph H as τ_A. This graph is called a *constraint graph* [1, 7]. It has been observed in [1] that any feasible schedule imposes orientation on the edges of the constraint graph. And vice versa, any acyclic orientation of the edges in the constraint graph induces a feasible schedule. Acyclic graph obtained by orientation of the edges in the constraint graph is called a network [15]. Although the number of instances is infinite the number of different networks is finite. Our goal is to show that for any instance of $\Pi(G)$ there exists an orientation of the edges such that a schedule specified by this orientation meets $OPT(I) \leq \rho(G)LB$, where $\rho(G)$ does not depend on the instance. The makespan of the schedule specified by a network $\overrightarrow{H}$ and an instance I is equal to the length of a *critical path* in this directed graph, which, in turn, must be a *complete* path in the digraph. For example, the network shown in Fig.3 has four complete paths.

It is obvious that different complete paths are critical for different instances. In the next section we propose a branch and bound algorithm to find $\rho(G)$ by breaking up the set of instances of $\Pi(G)$ into smaller subsets, in which instances have the same

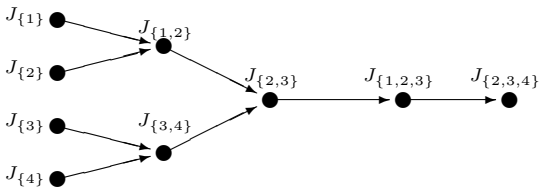

Fig. 3. The network obtained for some orientation of edges in the constraint graph H of $\Pi(P_4)$, the transitive arcs are omitted

critical paths in the selected set of constraint graphs. The algorithm calculates upper and lower bounds on $\rho(G)$ over each subset of instances and use the bounds to discard certain subsets from further consideration. The algorithm starts with $\rho(G) = 1$ and search the worst instance for which $\rho(G)$ has the maximal value. It terminates when each subset has either produced a solution with the same $\rho(G)$, or has been shown to contain no better solution than the best one found so far by the algorithm.

3 Branch and bound algorithm

In this section we present a branch and bound algorithm to compute $\rho(G)$ and to find an instance in which $\frac{OPT(I)}{LB(G)}$ attains this value. The algorithm constructs a search tree T. Each vertex v of T contains some subset of instances of $\Pi(G)$. Algorithm calculates an upper bound on $\rho(G)$ for this subset. In the calculation of the upper bound algorithm finds the instance, in which the optimal makespan is a lower bound on $\rho(G)$.

3.1 Branching

The root v_0 contains all instances of $\Pi(G)$. Initialize $\lambda = 1$ and $\nu(v_0) = \infty$, where λ is a current lower bound on $\rho(G)$ and $\nu(v)$ is an upper bound on $\rho(G)$ in the vertex v. Take an arbitrary network $\overrightarrow{H}_0$ and consider a set of complete paths of $\overrightarrow{H}_0$. We remove paths of length less than or equal to λ. The network on Fig.3 contains two such paths: $(J_{\{2\}} \to J_{\{1,2\}} \to J_{\{2,3\}} \to J_{\{1,2,3\}} \to J_{\{2,3,4\}})$ and $(J_{\{3\}} \to J_{\{3,4\}} \to J_{\{2,3\}} \to J_{\{1,2,3\}} \to J_{\{2,3,4\}})$.

Let $\mathcal{K}(\overrightarrow{H}_0) = \{p_1, p_2, \ldots, p_{|\mathcal{K}(\overrightarrow{H}_0)|}\}$ be the set of non-trivial paths in the network $\overrightarrow{H}_0$. We construct $|\mathcal{K}(\overrightarrow{H}_0)|$ subsets, one for each path $p_i \in \mathcal{K}(\overrightarrow{H}_0)$. The i-th child of v_0 contains instances of $\Pi(G)$ that have the critical path p_i in $\overrightarrow{H}_0$.

The branching on other vertices in the search tree is similar to previous one. Let v' be a vertex of the search tree such that $\nu(v') > \lambda$. Let V be the set of vertices in the v_0-v'-path in the search tree. Each vertex v in V corresponds to a critical path in some network chosen during the branching procedure of its parent. By $\mathcal{K}_v$ denote the set of such paths. We choose a network $\overrightarrow{H}_v$, such that the network $\overrightarrow{H}_v$ doesn't contain complete paths from $\mathcal{K}_v$. In the next subsection we explain how to choose $\overrightarrow{H}_v$ correctly. Finally, we use the non-trivial complete paths in $\overrightarrow{H}_v$ to create children of v.

3.2 Upper and lower bounds

Let $\mathcal{K}_v = \{p_1, \ldots, p_N\}$, and let $\delta_{iA} = 1$ if a job $J_A \in p_i$ and $\delta_{iA} = 0$ otherwise. Then the length of p_i is equal to $\sum_{J_A \in \mathcal{J}} \delta_{iA}\tau_A$. Let η be the value of $\rho(G)$ on the set of instances in the vertex v. Then for any $p_i \in \mathcal{K}_v$ we have $\sum_{J_A \in \mathcal{J}} \delta_{iA}\tau_A \geq \eta$.

We note that any clique in H forms a (not necessary complete) path in $\overrightarrow{H}$. Let $\mathcal{K}_{LB}$ be a set of paths corresponding to maximal cliques in H. Then we have $\sum_{J_A \in \mathcal{J}} \delta_{kA}\tau_A \leq 1$ for all $p_k \in \mathcal{K}_{LB}$.

In order to obtain an upper bound $\nu(v)$ on $\rho(G)$ in the vertex v, we solve the following linear program (LP).

$$\eta \to \max$$

$$\sum_{A \in \mathcal{A}(G)} \delta_{kA}\tau_A \leq 1, \;\; p_k \in \mathcal{K}_{LB} \tag{1}$$

$$\sum_{A \in \mathcal{A}(G)} \delta_{iA}\tau_A \geq \eta, \;\; p_i \in \mathcal{K}_v \tag{2}$$

$$\tau_A \geq 0, \;\; J_A \in \mathcal{J}.$$

The rational variables τ_A represent processing times of jobs. Inequalities (1) and (2) must hold for any instance $I \in v$. Thus an optimal solution of LP find an instance I', such that any critical path $p_i \in \mathcal{K}_v$ has a length greater than or equal to η and this value is the maximum possible. We set $\nu(v) = \eta$.

If $\nu(v) \leq \lambda$ we terminate branching in the vertex v. Otherwise, we construct an optimal schedule σ^* for instance I'. If the number of vertices in the graph G is small we can solve an instance I' in reasonable time. Let λ^* be the length of the optimal schedule to the instance I'. First, if $\lambda < \lambda^*$ we update the lower bound λ, $\lambda := \lambda^*$. Second, if $\eta = \lambda^*$ we terminate branching in the vertex v. Finally, let $\eta > \lambda^*$. Let a network $\overrightarrow{H}_v$ correspond to the schedule σ^*. Since $\eta > \lambda^*$, the network $\overrightarrow{H}_v$ does not contain paths from the set $\mathcal{K}_v$. We use the network $\overrightarrow{H}_v$ to branch the vertex v.

The algorithm terminates when $\nu(v) \leq \lambda$ for all leaves in current search tree T. Moreover, $\nu(v) = \lambda$ for at least one leaf $v \in T$. Finally, we obtain $\rho(G) = \lambda$.

4 Results

To present our results we introduce the following notation. We denote the complete graph on n vertices by K_n. By P_n and C_n we denote the path on n vertices and the cycle on n vertices, correspondingly. We also consider the star $K_{1,n}$ on $n+1$ vertices and the special graphs shown in Fig. 4.

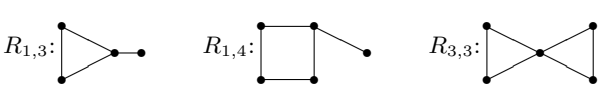

Fig. 4. The graphs $R_{1,3}$, $R_{1,4}$, $R_{3,3}$

A problem $\Pi(G)$ is called a *prime-version* and denoted by $\bar{\Pi}(G)$ if only jobs requiring more than one processor are allowed. Let $\bar{\rho}(G) = \max_{I \in \bar{\Pi}(G)} \frac{OPT(I)}{LB(G)}$. For each selected graph G we have considered two problems: $\Pi(G)$ and $\bar{\Pi}(G)$. The results obtained by the branch and bound algorithm are presented in the Table 1. The rows in Table 1 have the following meaning. Row 1 lists the value of the parameter G in the problem $\Pi(G)$. In rows 2 and 3 we present the values of $\rho(G)$ and $\bar{\rho}(G)$, correspondingly.

	P_4	P_5	C_3	C_4	$K_{1,3}$	$K_{1,4}$	K_4	$R_{1,3}$	$R_{1,4}$	$R_{3,3}$
$\rho(G)$	8/7	1,25	1,25	1,25	4/3	4/3	4/3	4/3	1,5	4/3
$\bar{\rho}(G)$	1	1	1	1	1	1	1	1	1,5	1

Table 1. Values of $\rho(G)$ for the selected graphs

In addition to the results presented in Table 1, we note that $\bar{\rho}(P_6) = 1$. A comparison of results presented in Table 1 shows that prime-version $\bar{\Pi}(G)$ is easier than the original problem $\Pi(G)$. In particular, $\bar{\rho}(G) = 1$ means that for any instance I of $\bar{\Pi}(G)$ we have $OPT = LB(G)$. This observation implies a simple exact linear time algorithm for the corresponding problem. Replace all the jobs of the same type by an aggregated job. Set the processing time of the aggregated job equal to the sum of processing times of original jobs. A new instance has a constant number of jobs. We can enumerate all active schedules and choose the best one. In the same way, we obtain a $\rho(G)$-approximation algorithms for the corresponding problems in the case $\rho(G) > 1$.

A naive enumeration procedure can be improved by using the search tree T constructed by the branch and bound algorithm when this algorithm calculates $\rho(G)$. Let h be the height of the search tree T. It is sufficient to consider at most h schedules chosen for branching in each vertex in a path from the root v_0 to some leaf of T.

Acknowledgements

We would like to thank Sergey Sevastianov and the anonymous referees for their helpful comments on an earlier draft of this work.

This research is supported by the Russian Science Foundation grant 15-11-10009.

References

1. Bianco, L., Dell'Olmo, P., Speranza, M.G.: Nonpreemptive scheduling of independent tasks with prespecified processor allocations. Naval Research Logistics. 41, 959–971 (1994)
2. Bierwirth, C., Meisel, F.: A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operations Research. 202, 615-627 (2010)
3. Blazewicz, J., Dell'Olmo, P., Drozdowski, M., Speranza, M. G.: Scheduling multiprocessor task on three dedicated processors. Information Processing Letters. 41, 257–280 (1992)
4. Chen, Bo, Potts, C.N., Woeginger, G. J.: A Review of Machine Scheduling: Complexity, Algorithms and Approximability. In: Handbook of Combinatorial Optimization, D.-Z. Du and P. M. Pardalos(Eds), Vol. 3, P. 21–169. Kluwer Academic Publisher, Amsterdam (1998)

5. Chun, H.N.: Scheduling as a multidimensional Placement Problem. Engineering Applications of Artificial Intellegence. 9, 261-273 (1996)
6. Dell'Olmo, P., Speranza, M.G., Tuza, Z.: Efficiency and effectiveness of normal schedules on three dedicated processors. Discrete Mathematics. 164, 67–79 (1997).
7. Drozdowski, M.: Scheduling for parallel processing. Computer communications and networks. Springer. 386 (2009)
8. Dyckhoff, H.: A topology of cutting and packing problems, European Journal of Operational Research. 44, 145–159 (1990)
9. Duin, C.W., Sluis, E.V.: On the complexity of adjacent resource scheduling, Journal of Scheduling. 9, 1, 49–62 (2006)
10. Kononov, A., Sevastianov, S., Tchernykh, I.: Polynomially solvable classes of the open shop problem on the base of different machine loads, in: The Third Workshop on Models and Algorithms for Planning and Scheduling Problems, Cambridge, U.K., April 7-11, P. 41 (1997)
11. Kononov, A., Sevastianov, S., Tchernykh, I.: When difference in machine loads leads to efficient scheduling in open shops, Annals of Operations Research, 92, 211-239 (1992) DOI:10.1023/A:1018986731638
12. Lim, A.: The berth planning problem, Operations Research Letters, 22(2-3), 105-110, (1999)
13. Lodi, A., Martello, S., Monaci, M.: Two-dimencional packing problem: A survey, European Journal of Operational Research, 141, 241–252 (2002)
14. J.J. Paulus, J. Hurink. Adjacent-Resource Scheduling. Why spatial resources are so hard to incorporate. Electronic Notes in Discrete Mathematics, 25: 113-116, 2006.
15. Sevastianov, S.V., Tchernykh, I.D.: Computer-aided Way to Prove Theorems in Scheduling. In: Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, Geppino Pucci (Eds.) ESA 1998, LNCS, Vol. 1461, pp. 502–513, Springer, Heidelberg (1998)
16. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science, 363 28–42 (2006)