Total Completion Time Minimization for Scheduling with Incompatibility Cliques

Klaus Jansen[1], Alexandra Lassota[2], Marten Maack[3], and Tytus Pikies[4]

[1] kj@informatik.uni-kiel.de, [2] ala@informatik.uni-kiel.de, [3] mmaa@informatik.uni-kiel.de, [4] tytpikie@pg.edu.pl

[1,2,3] Department of Computer Science, Faculty Of Engineering, Kiel University, 24098 Kiel, Germany
[4] Dept. of Algorithms and System Modeling, Gdańsk University of Technology, 80-233 Gdańsk, Poland

The Model

There is a set of jobs given. The jobs form a cluster graph. There is a set of machines given. No two jobs in a clique are allowed to be assigned to the same machine. How to schedule the jobs, respecting this additional requirement, to minimize the total completion time?

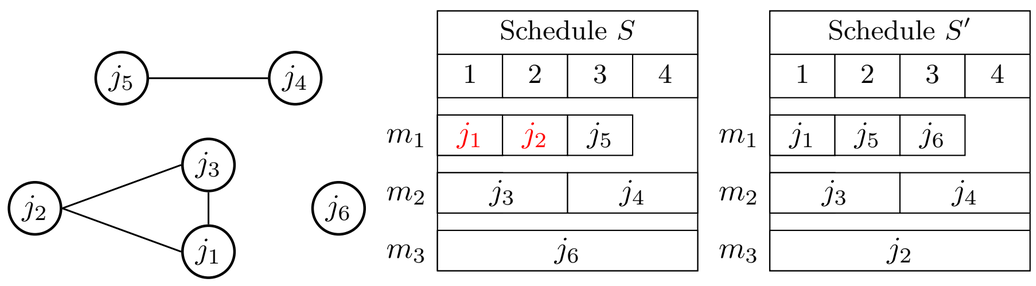

Fig. 1: A sample instance: the set of jobs $\{j_1, \ldots, j_6\}$, forming a cluster graph, the set of machines $\{m_1, m_2, m_3\}$. The schedule S_1 is unfeasible due to the fact that j_1, j_2 are from the same clique.

For an overview of the topic see: (Klaus Jansen, Alexandra Lassota, Marten Maack, Tytus Pikies: Total Completion Time Minimization for Scheduling with Incompatibility Cliques. ICAPS 2021: 192-200).
See also: (Tytus Pikies, Krzysztof Turowski, Marek Kubale: Scheduling with Complete Multipartite Incompatibility Graph on Parallel Machines. ICAPS 2021: 262-270) for an overview of the works in a similar model.

A Sample Application

- Consider a task system under difficult conditions like high electromagnetic radiation, or with an unstable power supply. Due to the environmental conditions, users prepare tasks in groups and want the jobs in a given group to be scheduled on different processors. That assures that even if a few processors fail, another processor will be able to execute at least part of the jobs.
- Due to the instability, the system even might stop working completely and in this case all jobs that are done only partially have to be scheduled again.
- The sum of completion times criterion tends to reduce the mean number of unfinished jobs at each moment in the schedule. For this reason, one could like to minimize the sum of completion times of the jobs respecting the additional reliability requirement given by the groups.

Polynomial Time Algorithms

- An optimal polynomial time algorithm for $P|G = \mathit{cliques}|\sum C_j$ was constructed. Interestingly, the division into cliques does not increase the total completion time, provided that a solution still exists.
- For $R|\text{cliques}, M(j), (p_k^i)_{k\in[b],i\in M}|\sum C_j$ an optimal polynomial time algorithm was constructed. We assume that the machines are unrelated, but the jobs forming each clique can be seen as copies.

Hardness Results

- APX-hardness of $P|\text{cliques}, M(k), p_i \in \{p_1 < p_2 < 2p_1\}|\sum C_j$. Here we assume that a clique k can be scheduled only on $M(k)$.
- APX-hardness of $R|\text{cliques}, p_i^j \in \{p_1 < p_2 < p_3\}|\sum C_j$.
- NP-hardness of $R|2 \text{ cliques}, p_j^i \in \{p_1 < p_2 < p_3\}|\sum C_j$.

FPT Results

- $P|\text{cliques}, M(k)|\sum C_j$ is FPT w.r.t. b.
- $R|\text{cliques}|\sum w_j C_j$ is FPT w.r.t. m, $p_{\max}$, ϑ.
- $R|\text{cliques}|\sum w_j C_j$ is FPT w.r.t. b, $p_{\max}$, ϑ.

Where b is the number of cliques, m is the number of machines, $p_{\max}$ is the maximum processing time, ϑ is the number of job kinds.

Methods

- Matching and flow methods:

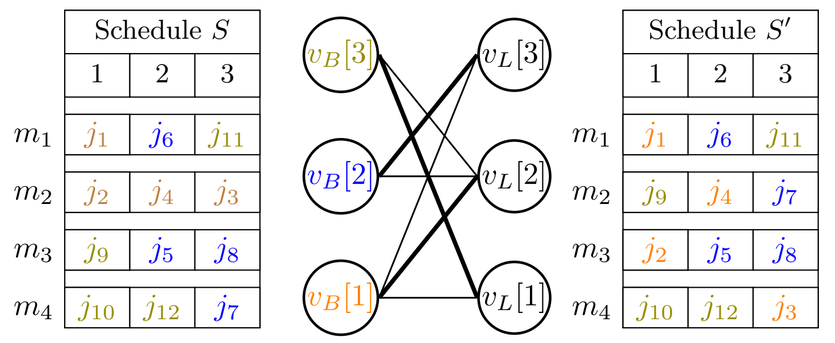

Fig. 2: An illustration of an application of the method developed for $P|G = \text{cliques}|\sum C_j$. Let the set of cliques be given by $\{j_1, j_2, j_3, j_4\}$, $\{j_5, j_6, j_7, j_8\}$, $\{j_9, j_{10}, j_{11}, j_{12}\}$ and let $i = 2$ (which means that m_1 has already a set of compatible jobs assigned). Notice how using a matching in the constructed graph the jobs can be exchanged to make the jobs assigned to m_2 compatible.

- n-fold IPs: For the first result the Integer Program we propose:

$$\min \sum_{\ell,k\in[b]} \sum_{s=1}^{y_{k,\ell}} p_{k,s}$$

$$\sum_{C\in\mathcal{C}} x_{C,i} = 1 \qquad \forall i \in M \qquad (1)$$

$$\sum_{i\in M} \sum_{C\in\mathcal{C}(k,\ell)} x_{C,i} = y_{k,\ell} \qquad \forall k \in [b], \ell \in [b] \qquad (2)$$

$$y_{k,b} = n_k \qquad \forall k \in [b] \qquad (3)$$

Despite the fact that the program has not a super constant number of variables and has not a linear objective function, it can be solved efficiently by exploiting convexity of the objective function and using n-fold IPs.