High-Quality Algorithms for Makespan Minimization

Description

Job scheduling problems arise in various application areas, for example manufacturing systems and data centers. The makespan minimization problem on identical parallel machines (also denoted by $P||C_{\text{max}}$) asks for a distribution of $n$ jobs to $m$ machines. Each job has a processing time and the goal is to minimize the completion time of the latest job.

Makespan minimization is a classical NP-complete problem and there are several known approximation algorithms, such as the Longest Processing Time (LPT) algorithm. For small instances, techniques such as ILP or SAT-encoding allow to find an optimal solution. In addition, makespan minimization permits efficient approximation schemes (EPTAS). However, despite recent advances in the practicability of PTAS approaches\(^1\) it is still expensive to run a PTAS with higher precision than simple approximation algorithms. In practice, the results might be worse for some instances while requiring a high running time. Thus, the engineering of practical high-quality algorithms is still an open question.

The goal of this thesis is to combine ideas from approximation algorithms, PTAS and local search heuristics to achieve high quality in practice (note that the focus is not on worst case guarantees). A promising approach is to use a portfolio of different solvers that can run in parallel and exchange information about machine configurations with small span. The algorithm then maintains a set of configurations that can be recombined to find better solutions. This allows to explore the solution space heuristically, hopefully resulting in significantly better running times than PTAS algorithms. In addition to the generation, selection and combination of machine configurations, there are multiple related areas that could be explored. These include heuristics for improving known solutions, reducing the size of the search space and solving a small set of machines with high quality. The results should be evaluated on a comprehensive benchmark set and compared with competing algorithms.

Prerequisites

- Interest in optimization algorithms and parallel algorithms
- Good programming knowledge in either (modern) C++ or Rust
- Willingness to do a detailed experimental evaluation

---