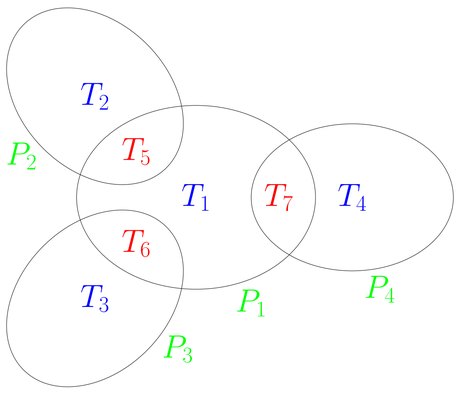
Exact Algorithms for Scheduling Programs with Shared Tasks

Doctorant:Théo Nazé, Encadrants:Imed Kacem, Giorgio Lucarelli, Camel Tanougast

Laboratoire LCOMS, Université de Lorraine

{theo.naze,imed.kacem,giorgio.lucarelli,camel.tanougast}@univ-lorraine.fr

Problem Presentation



- We consider n programs, k tasks (the T_i), and c shared tasks.
- Each task has a processing time.
- A program is successfully performed if all of its tasks are processed by one machine.
- If two programs sharing a non-empty subset of tasks are scheduled on the same machine, these shared tasks have to be performed only once.

Objective and Methods

Our objective is to schedule the program present in our input data on two parallel homogeneous machines, in order to minimize the makespan, that is to say the completion time of the last performed task. This is an NP-Hard problem (cf Partition). This objective has not been previously studied in this context. We present exact algorithms using branching techniques.

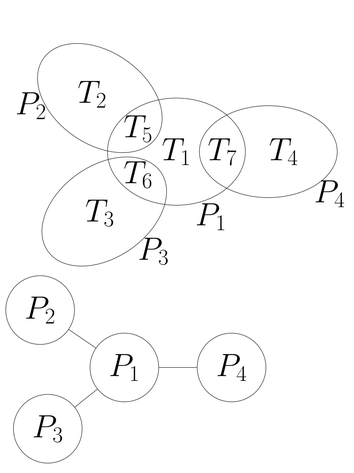
Applications

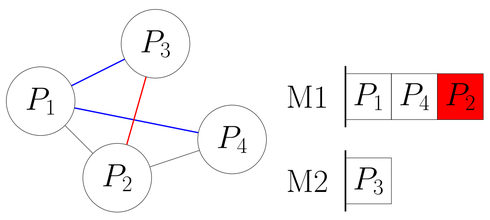
- Problem known in the literature under the names of VM Packing and Pagination
- Correspond to the problem of stocking virtual machines on physical server (the virtual machines are the programs , the memory pages are the tasks, and the physical server are the machine of our scheduling problem).
- The objectives previously studied are the maximization of the number of hosted virtual machines given a fixed number of physical servers, and the minimization of the number of physical servers used given a fixed number of virtual machines.
- Applications in parallel computing.

Trivial Algorithms

- A first basic exact algorithm branches on programs, and runs in $\mathcal{O}^*(2^n)$ steps, as each program should be scheduled on one of the two machines.
- The best exact algorithm for the Knapsack problem, runs in $\mathcal{O}^*(2^{\frac{N}{2}})$ steps. We adapted this algorithm to be run on partial schedules of our problem.
- By branching on the shared tasks, a trivial algorithm solved Pagination in $\mathcal{O}^*(3^c 2^{\frac{n}{2}})$. We enumerate the ways to assign the shared tasks to the machines, and apply the modified knapsack algorithm to complete the schedules.

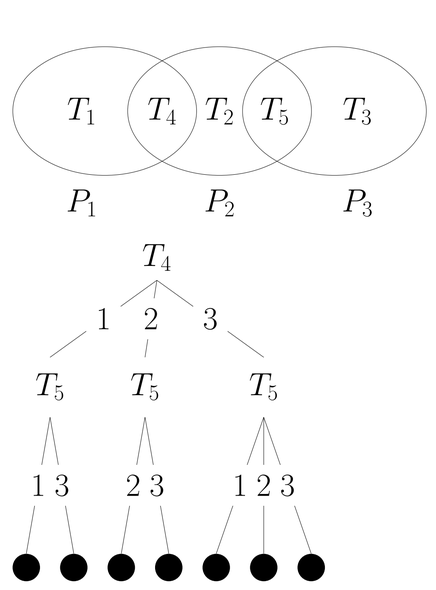
Algorithm branching on links





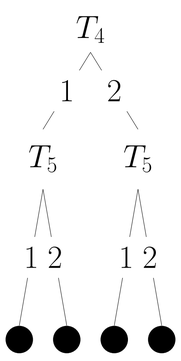
Here, the program P_1 is scheduled with the program P_4, the program P_3 is not scheduled with the program P_1, and the program P_2 is not scheduled with the program P_3. By performing this branching on every C connected component in our input data and applying the modified knapsack algorithm, we get a $\mathcal{O}^*(2^{n-\frac{C}{2}})$ algorithm.

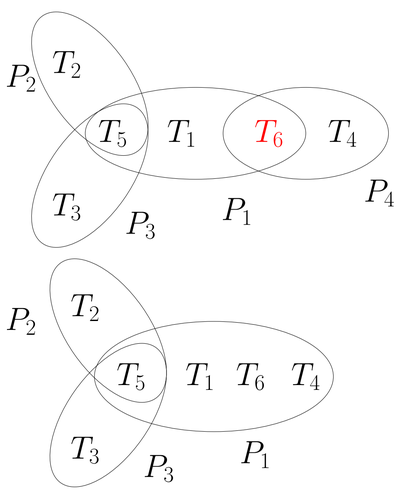
Ternary shared tasks branching algorithm



By assigning the shared tasks of our input data on the first, the second, or on both machines, we show that some schedules are unfeasible. This is the case here if the shared task T_4 have to be performed on the first machine, while the shared task T_5 have to be performed on the second one, or if T_4 have to be performed on the second machine while T_5 have to be performed on the first.
By establishing a lower bound on the number of shared task assignments producing unfeasible schedules for any given input data, we demonstrate that this way of proceeding yields a $\mathcal{O}^*(7^{\frac{c}{2}}2^{\frac{n}{2}})$ step algorithm.

Binary shared tasks branching

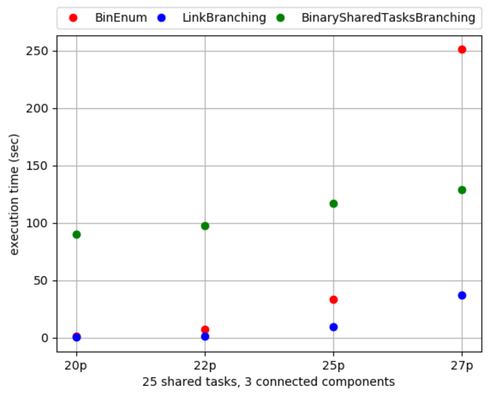


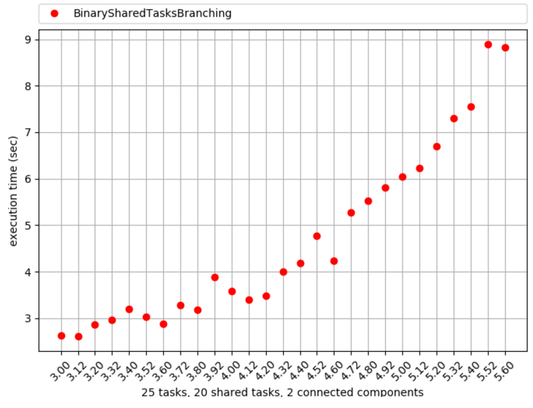


For this approach, we consider that each shared task can either be assigned to both machines, or to only one without specifying which one. The first interest is that the tree enumerating the shared task assignments creates two branches at each step.
The shared tasks adjacent to exactly two programs (the shared tasks of degree two) have a property allowing us to reduce the complexity of the associated algorithm. Indeed, like illustrated with those two hypergraphs, if a shared task of degree two have to be scheduled on one machine, then the two neighboring programs will also be scheduled on this machine. And if a shared task of degree two have to be scheduled on both machines, then the two neighboring programs will be scheduled on different machines. The number of elements given to our modified knapsack algorithm is then reduced. This technique yields a $\mathcal{O}(2^c 2^{\frac{n}{2}})$ step algorithm. More involved techniques allow us give an algorithm running in $\mathcal{O}(1.5^c 2^{\frac{2}{n}})$ steps.

Experiments

We implemented all these algorithms in the C language, and conducted experiments on different instances of our problem. We focused on three main parameters while building our test instances: the number of programs (20 to 35), the number of shared tasks (10 to 35), and the number of connected components (1 to 6) present in our input data. Twenty different instances have been tested for a given combination of those parameters.





Conclusion and Perspectives

We have investigated the NP-complete problem of Pagination , and presented different exact algorithms with worst-case time-complexity guarantees, using various branching techniques. We are currently working on improving these algorithms, and are also studying the field of parameterized complexity with the purpose of establishing a fixed-parameter algorithm and obtaining a kernel for Pagination.