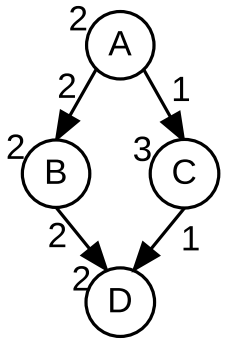
Column generation algorithm for task scheduling in parallel systems

Jake Bowden, Sergei Ogai, **Oliver Sinnen**

Parallel and Reconfigurable Computing lab
Department of Electrical, Computer, and Software Engineering

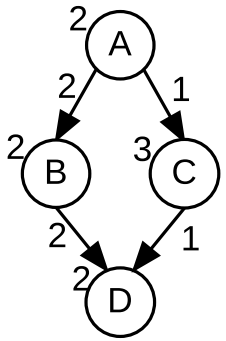
Task scheduling with communication delays

Task Graph



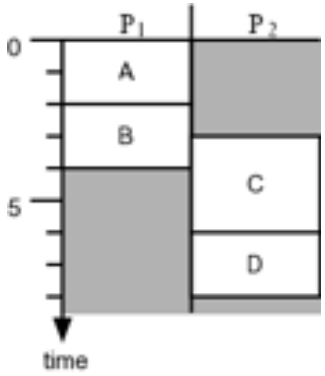
Task scheduling with communication delays

Task Graph



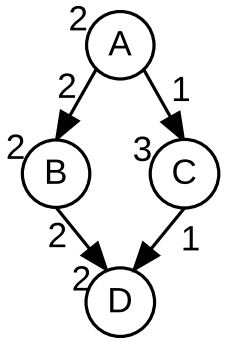
$\Rightarrow$

Schedule on homogeneous processors



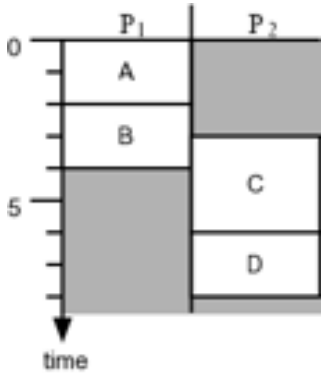
Task scheduling with communication delays

Task Graph



$\Rightarrow$

Schedule on homogeneous processors



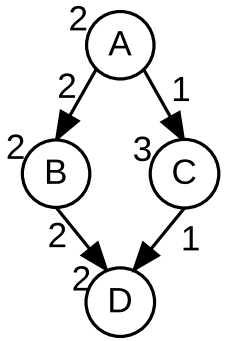
Objective: Minimise Makespan

$\Rightarrow$ Strong NP-hard optimisation problem

$\Rightarrow$ Usually heuristics used (e.g. List scheduling)

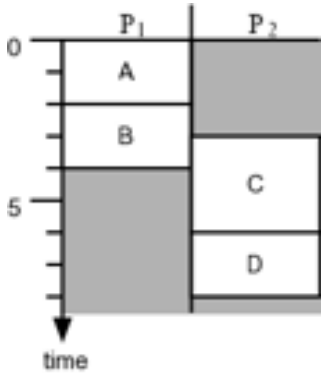
Task scheduling with communication delays

Task Graph



$\Rightarrow$

Schedule on homogeneous processors



Objective: Minimise Makespan

$\Rightarrow$ Strong NP-hard optimisation problem

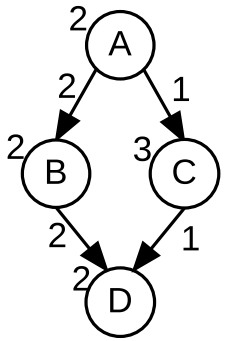
$\Rightarrow$ Usually heuristics used (e.g. List scheduling)

Optimal scheduling approaches – small instances (up to 30 tasks)

- State-space search (BnB, A*)
- Mixed Integer Linear Programs (MILP)

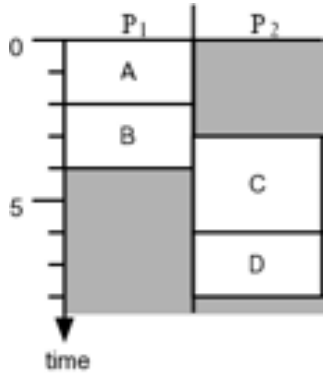
Task scheduling with communication delays

Task Graph



$\Rightarrow$

Schedule on homogeneous processors



Objective: Minimise Makespan

$\Rightarrow$ Strong NP-hard optimisation problem

$\Rightarrow$ Usually heuristics used (e.g. List scheduling)

Optimal scheduling approaches – small instances (up to 30 tasks)

- State-space search (BnB, A*)
- Mixed Integer Linear Programs (MILP)

Here: Column Generation approach

Preliminary work: simplified scheduling problem

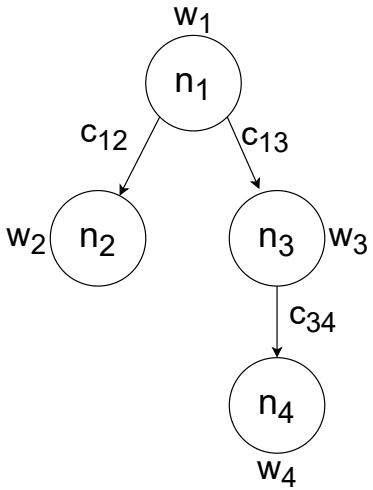
Content

Overview

Simplified Scheduling Problem

Full model: task graph $G = \{V, E, W, C\}$

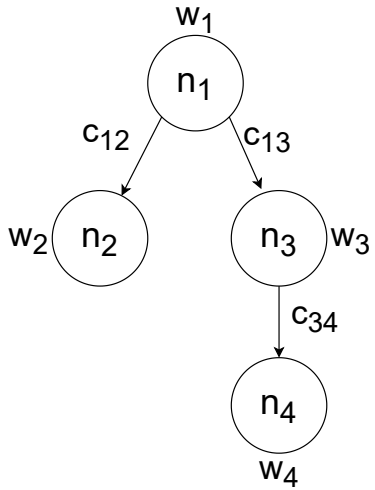
- tasks: $n_i \in V$
- edges: $e_{ij} \in E$ between n_i and n_j, precedence constraints/communication
- task weights: $w_i \in W$, work
- edge weights: $c_{ij} \in C$, communication costs



Simplified Scheduling Problem

Full model: task graph $G = \{V, E, W, C\}$

- tasks: $n_i \in V$
- edges: $e_{ij} \in E$ between n_i and n_j, precedence constraints/communication
- task weights: $w_i \in W$, work
- edge weights: $c_{ij} \in C$, communication costs



Simplified model: set of tasks

- tasks: $n_i \in V$
- task weights: $w_i \in W$, work
- no edges $\rightarrow$ no dependences

Simplified scheduling problem

Original scheduling problem

- allocation problem (task to processor)
- ordering problem (execution order of tasks)

Simplified scheduling problem

Original scheduling problem

- allocation problem (task to processor)
- ordering problem (execution order of tasks)

Simplified problem

$\Rightarrow$ only allocation problem

- independent tasks – order does not matter

Simplified scheduling problem

Original scheduling problem

- allocation problem (task to processor)
- ordering problem (execution order of tasks)

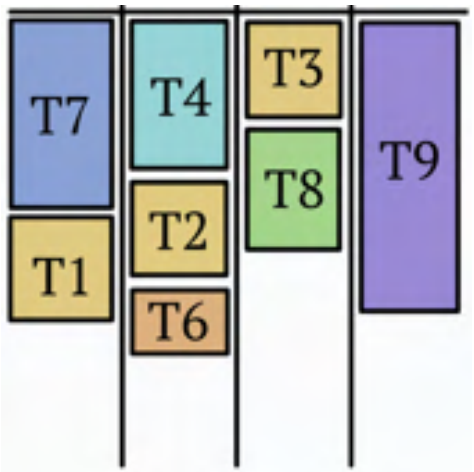
Simplified problem

$\Rightarrow$ only allocation problem

- independent tasks – order does not matter

Target System

- Set of homogeneous processors P
- No task preemption, no overlap on same processor

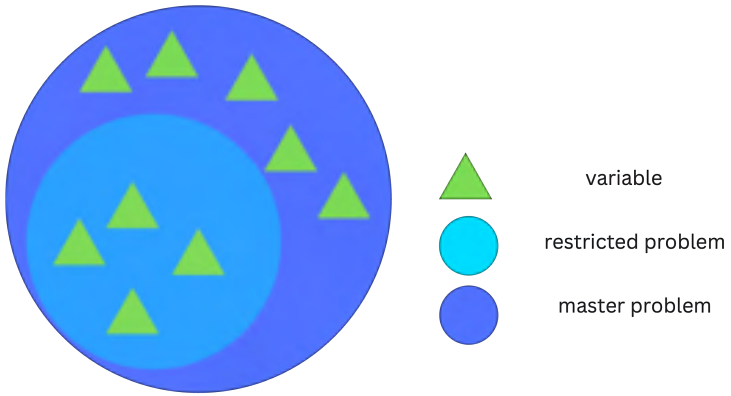


Overview

Column generation

Column generation approach

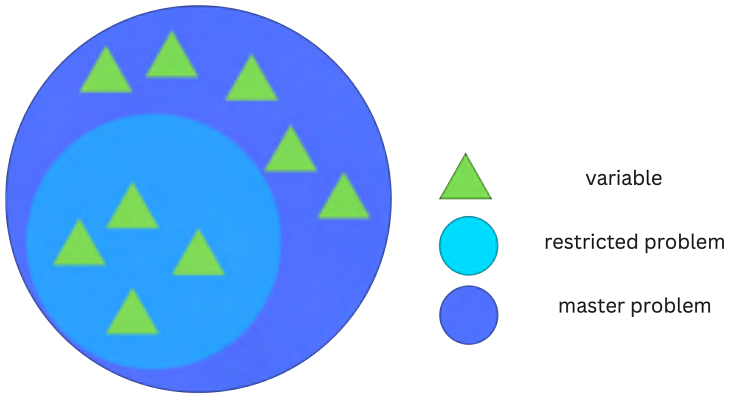
- Master problem: Formulate problem so valid solution can be found with limited set of variables
- Restricted master problem: Solve that restricted problem
- Subproblem: Can this solution be improved with more variables (columns)?
  - If **yes**: add columns and repeat
  - If **no**: solution is optimal also for Master problem



Column generation

Column generation approach

- Master problem: Formulate problem so valid solution can be found with limited set of variables
- Restricted master problem: Solve that restricted problem
- Subproblem: Can this solution be improved with more variables (columns)?
  - If **yes**: add columns and repeat
  - If **no**: solution is optimal also for Master problem



Why?

- Can be more efficient than other more direct formulations
- Subproblem needs to be solvable efficiently

(Column - variables in Linear Program)

Cutting stock problem

Well known problem for Column Generation

- Stock (e.g. sheet) of width W
- Different demanded items of different width
- w_i width of item i
- d_i demand (quantity) for item i

Objective: Minimise number of stocks used

Cutting stock problem

Well known problem for Column Generation

- Stock (e.g. sheet) of width W
- Different demanded items of different width
- w_i width of item i
- d_i demand (quantity) for item i

Objective: Minimise number of stocks used

Cutting pattern example

27	25	17	17	

$\Rightarrow$ cutting pattern P is a column!

Overview

Related work

Some previous work address similar scheduling problem.

[1] J. M. van den Akker, J. A. Hoogeveen, and J. W. van Kempen. "Using column generation to solve parallel machine scheduling problems with minmax objective functions". In: *Journal of Scheduling* 15 (2010), pp. 801–810. DOI: 10.1007/s10951-010-0191-z.

[2] Z.L. Chen and W.B. Powell. "Solving Parallel Machine Scheduling Problems by Column Generation". In: *INFORMS Journal on Computing* 11.1 (1999), pp. 78, 94. DOI: 10.1287/ijoc.11.1.78.

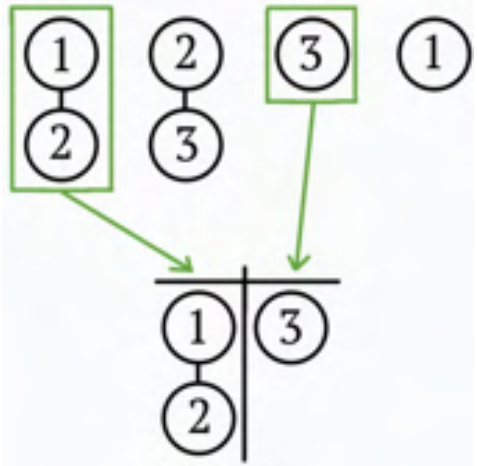
Not solving our original problem, but similar ideas used here

Proposed column generation

Using cutting stock approach

In fact, using binary cutting stock problem

- each item (i.e. task) only needed once



Restricted Master Problem (RMP)

- Column: single processor schedule s
  - i.e. tasks allocated to one processor
- Selected columns must cover all tasks

$$\min : \sum_{s \in S} x_s$$

$$\sum_{s \in S} x_s a_{js} \geq 1 \qquad \forall j \in J$$

$x_s = 1$ if single processor schedule (column) s was selected, 0 otherwise

$a_{js} = 1$ if task j is part of schedule s, 0 otherwise

Subproblem – Pricing problem

Exponential size: set S of all possible single processor schedules

Subproblem

Finding new single processor schedules

$$\max : \sum_{j \in J} \pi_j a_j$$

$$\sum_{j \in J} w_j a_j \leq L$$

π_j dual variables from Restricted Master Problem
$a_j = 1$ if task j is selected for single processor schedule, 0 otherwise
L is (freely set) upper limit for execution time

If objective value $\sum_{j \in J} \pi_j a_j > 1$, adding found single processor schedule to RMP is beneficial
Otherwise RMP solution is already optimal

Branch and Price

Approach so far has no integer guarantee
E.g. solutions uses 0.8 of one column and 0.2 of another column etc.

Branch and Price

Approach so far has no integer guarantee
E.g. solutions uses 0.8 of one column and 0.2 of another column etc.

Branch and Bound

- branching on 0/1 choices for variables not viable

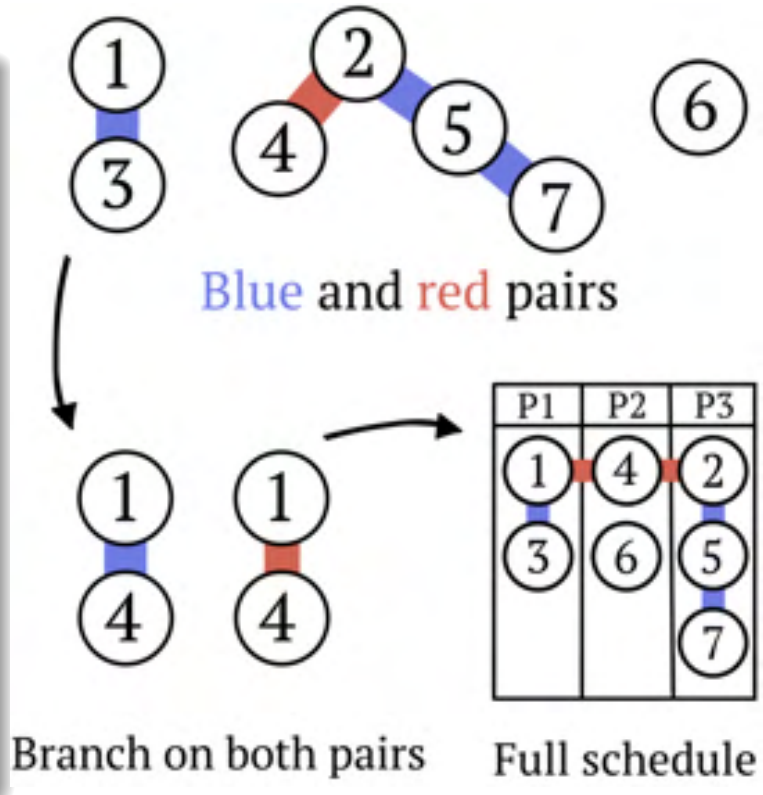
Branch and Price

Approach so far has no integer guarantee
E.g. solutions uses 0.8 of one column and 0.2 of another column etc.
Branch and Bound

- branching on 0/1 choices for variables not viable

Branching strategy

- Select task pairs from fractional columns
- Branch (blue): pair needs to be on same processor
- Branch (red): pair needs to be on different processors
- Add decision (either red or blue) as constraint to RMP
- Re-optimise RMP



Binary search

OK, but how does this all help us?

- Minimises number of processors
- Not makespan!

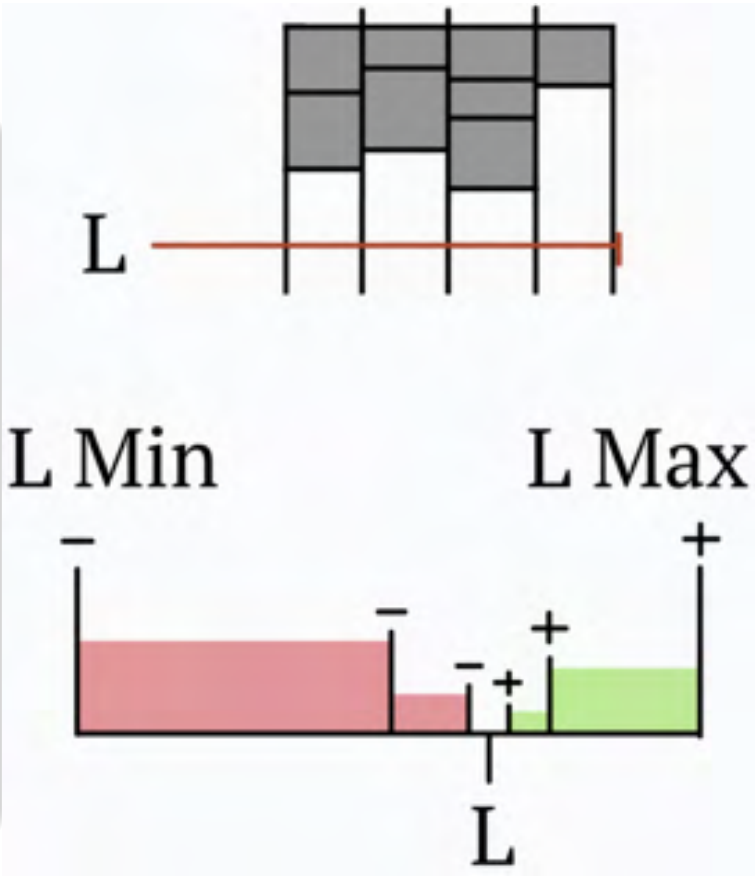
Binary search

OK, but how does this all help us?

- Minimises number of processors
- Not makespan!

Binary Search

- Compute L_{min} (e.g. lower bound) and L_{max} (e.g. heuristic solution
- Select $L = L_{\text{min}} + \lceil \frac{L_{\text{max}} - L_{\text{min}}}{2} \rceil$
- Run Column Generation Approach
  - If #singleProcessorSchedules $\leq$ #processors: $L_{\text{max}} = L$
  - Otherwise $L_{\text{min}} = L$
- Repeat



Overview

Evaluation

Implementation

- Algorithm implemented in Java
- Solver: GUROBI
- Also implemented simple, direct MILP

$$\min : L$$

$$\sum_{p \in P} y_{jp} \geq 1 \qquad \forall j \in J$$

$$\sum_{j \in J} w_j y_{jp} \leq L \qquad \forall p \in P$$

Evaluation

Implementation

- Algorithm implemented in Java
- Solver: GUROBI
- Also implemented simple, direct MILP

$$\min : L$$

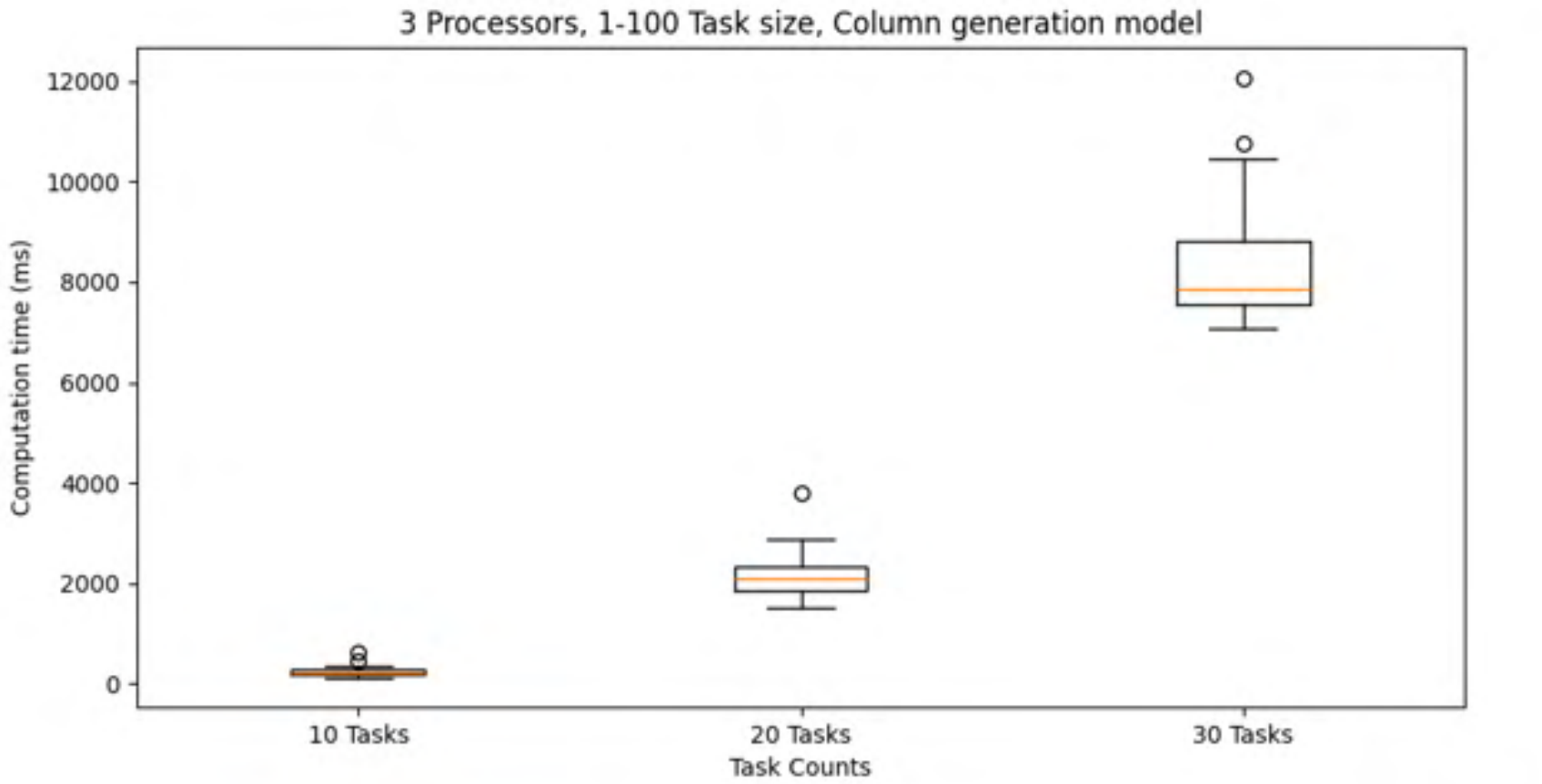
$$\sum_{p \in P} y_{jp} \geq 1 \qquad \forall j \in J$$

$$\sum_{j \in J} w_j y_{jp} \leq L \qquad \forall p \in P$$

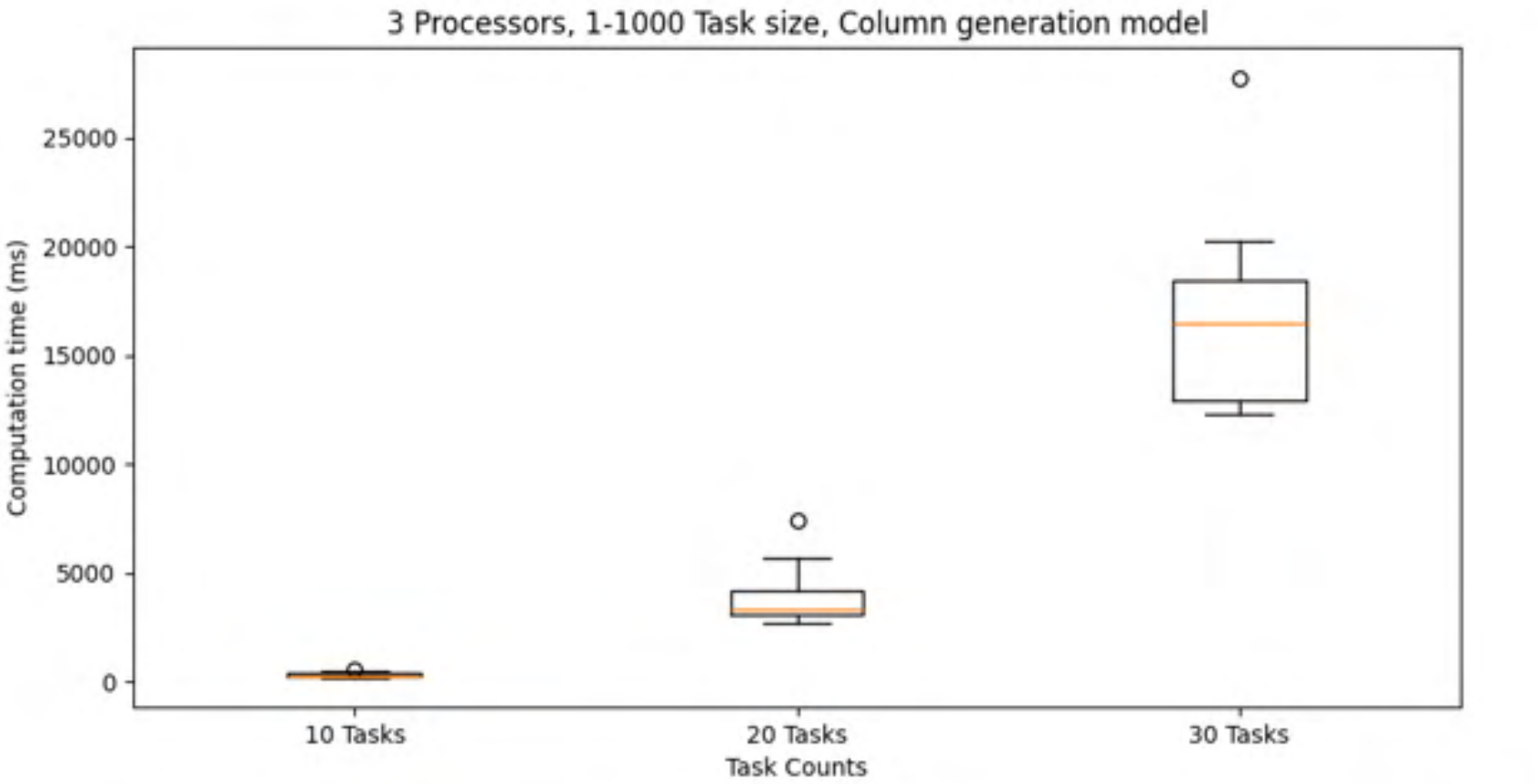
Workload

- Small task sets generated: 10, 20, 30
- Random weights, uniformly from 1-100 or 1-1000
- Different numbers of processor: 3, 5, 10

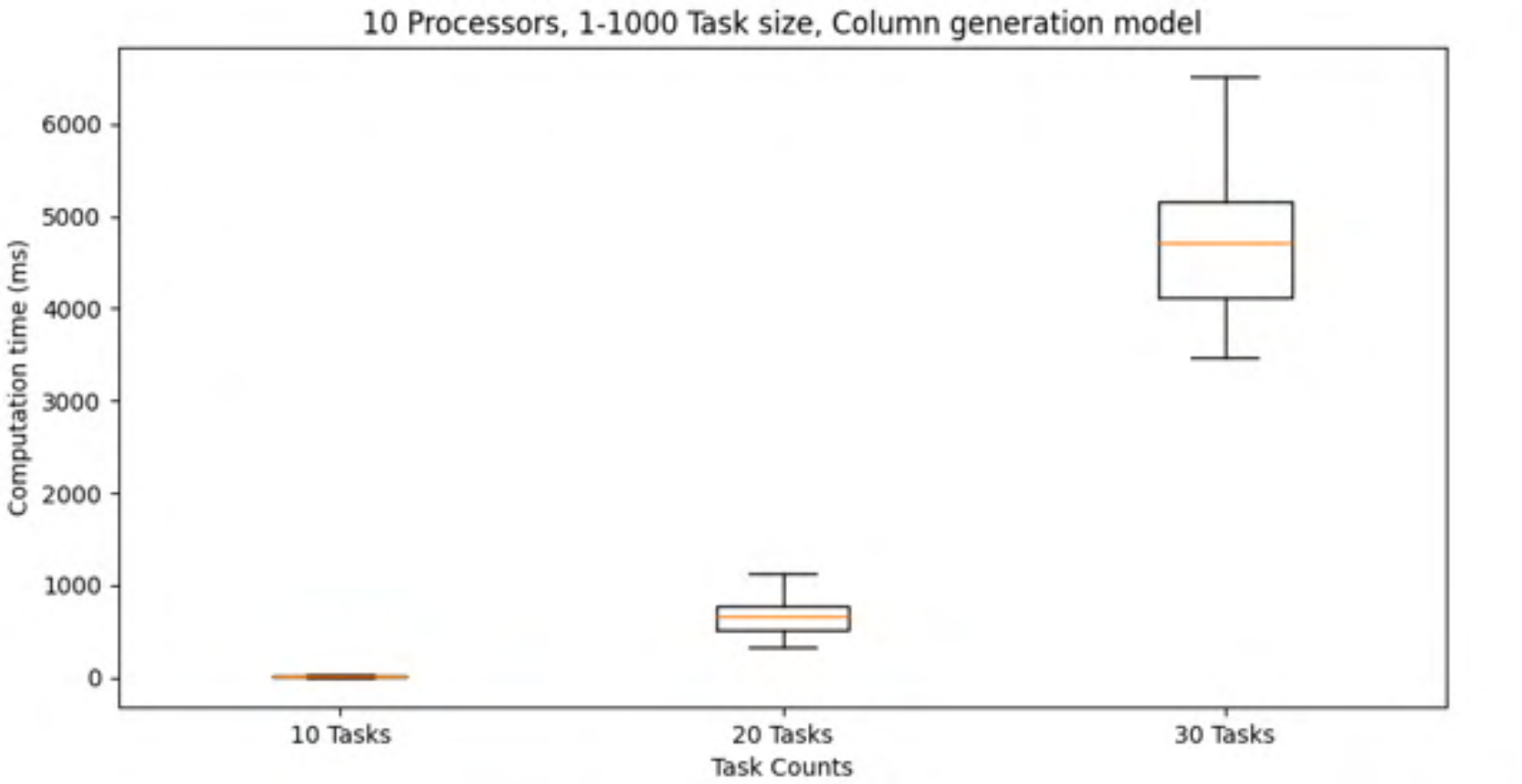
Results – 3 processors, 1-100 weights



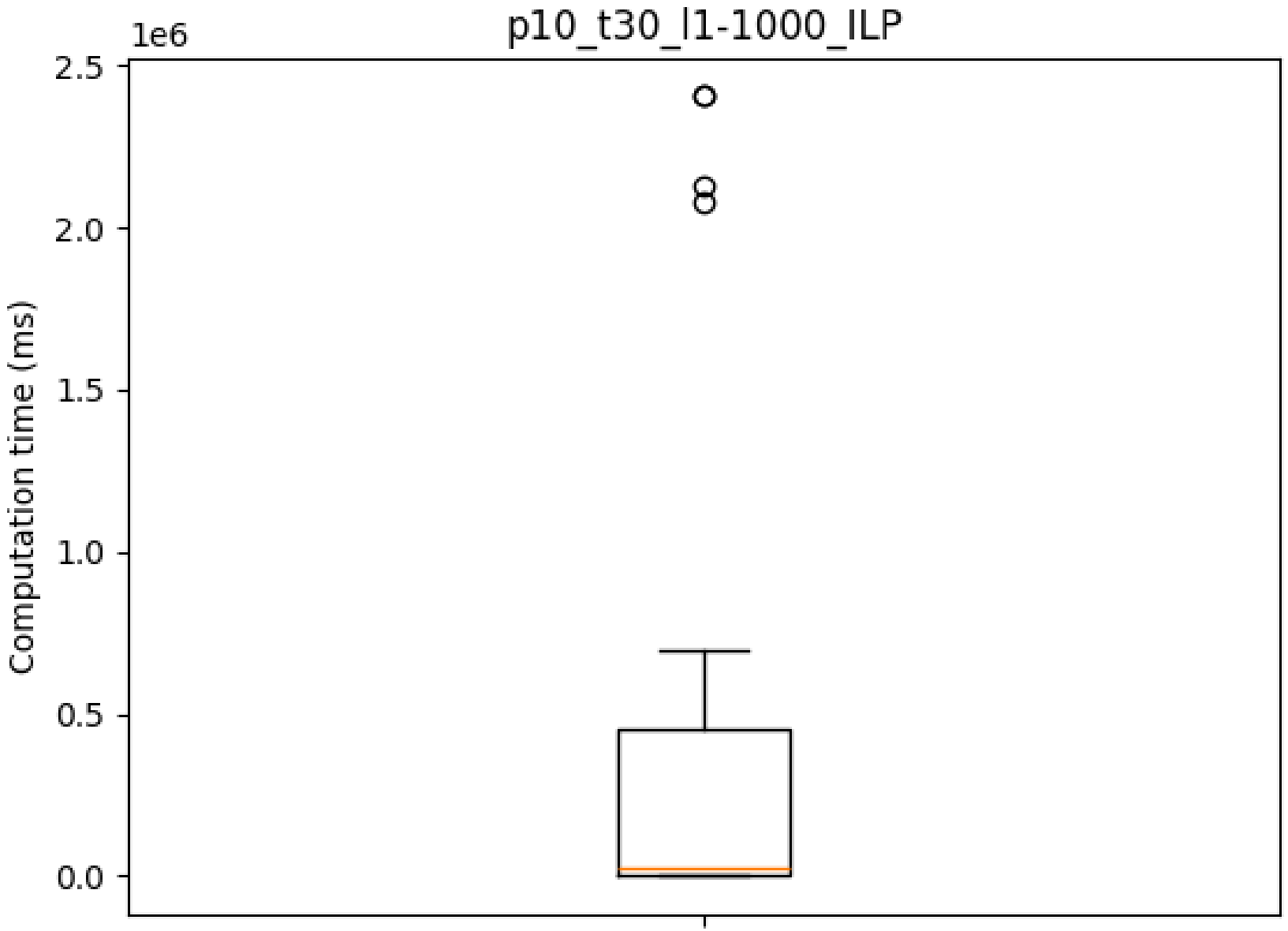
Results – 3 processors, 1-1000 weights



Results – 10 processors, 1-1000 weights



Simple MILP – 10 processors, 1-1000 weights



Conclusion

- Working Column Generation formulation for independent tasks
- Slower than simple MILP in most cast
- BUT, no extreme outliers in our experiments
- Full potential not reached

Conclusion

- Working Column Generation formulation for independent tasks
- Slower than simple MILP in most cast
- BUT, no extreme outliers in our experiments
- Full potential not reached

Future

- Dynamic programming approach for subproblem
- Use multiplicity of tasks – realistic in practice
- Add dependences
- Add communication costs