UNIVERSITY OF LIVERPOOL

LEVERHULME TRUST

The Leverhulme Research Centre for Functional Materials Design

Fast and Safe Scheduling of Robots

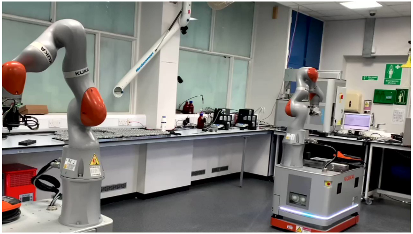
Duncan Adamson, Nathan Flaherty, Igor Potapov and Paul G. Spirakis

BCTCS 2025

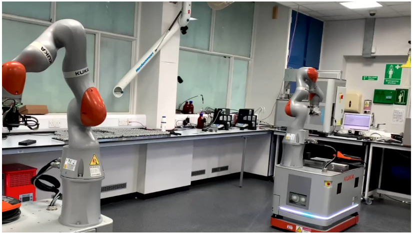
diamond Hartree Centre Imperial College London ICNS King Abdullah University of Science and Technology University of Southampton

Contents

Motivation

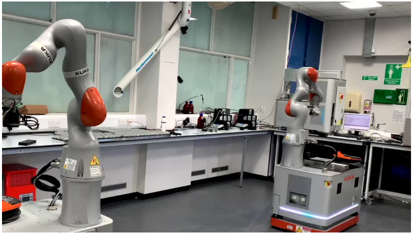


Motivation



- Many labs around the world have mobile robots to automate chemistry experiments.

Motivation



- Many labs around the world have mobile robots to automate chemistry experiments.
- We are interested in routing these robots in order to complete tasks based around the lab as fast as possible.
- Our model can also be relevant to manufacturing and logistics.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

[4] A sequence alternating between walks and tasks.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

[3] At no point in time do two robots share the same vertex or traverse the same edge

[4] A sequence alternating between walks and tasks.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

[2] For each task $t = (v_t, d_t) \in T$ some robot remains on v_t for d_t timesteps

[3] At no point in time do two robots share the same vertex or traverse the same edge

[4] A sequence alternating between walks and tasks.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

[1] One with a shortest time span[5]

[2] For each task $t = (v_t, d_t) \in T$ some robot remains on v_t for d_t timesteps

[3] At no point in time do two robots share the same vertex or traverse the same edge

[4] A sequence alternating between walks and tasks.

k-Robot Scheduling

The Setting:

- A graph $G = (V, E)$, of n vertices, which represents a discretisation of a lab space.
- k robots (agents) $R_1, ..., R_k$ starting on vertices $sv_1, ..., sv_k$.
  - Each edge takes 1 timestep to traverse
- A set, T, of m tasks, $t_1, ..., t_m$, where t_i is a pair consisting of a location $v \in V$ and duration $d_{t_i} \in \mathbb{N}$.
- **GOAL:** Find a fastest[1] task-completing[2] collision-free[3] set of schedules[4] for the k robots.

[1] One with a shortest time span[5]

[2] For each task $t = (v_t, d_t) \in T$ some robot remains on v_t for d_t timesteps

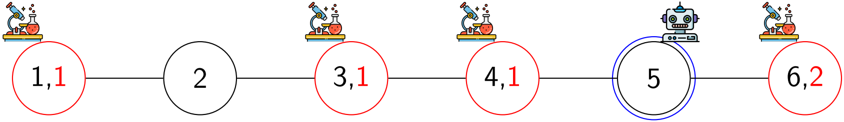
[3] At no point in time do two robots share the same vertex or traverse the same edge

[4] A sequence alternating between walks and tasks.

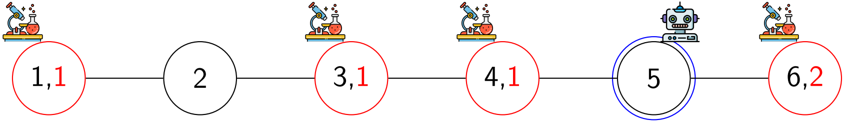
[5] Total number of time steps taken by all schedules in the set

Contents

One robot on a path

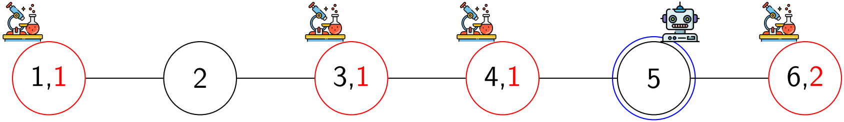


One robot on a path



- $C_1(P, T, sv) := \min(|sv - v_{t_1}|, |sv - v_{t_m}|) + v_{t_m} - v_{t_1} + \sum_{i \in [m]} d_i.$

One robot on a path



- $C_1(P, T, sv) := \min(|sv - v_{t_1}|, |sv - v_{t_m}|) + v_{t_m} - v_{t_1} + \sum_{i \in [m]} d_i.$
- For the example above:

$$C_1(P_6, \{(1,1), (3,1), (4,1), (6,2)\}, 5) = min(5-1, 6-5) + 6 - 1 + 5 = 11$$

2-Partition Algorithm

- For two robots, R_L and R_R, starting on vertices sv_L and sv_R, we wish to split the tasks into two where R_L completes $t_1, ..., t_q$ and R_R does $t_{q+1}, ..., t_m$.

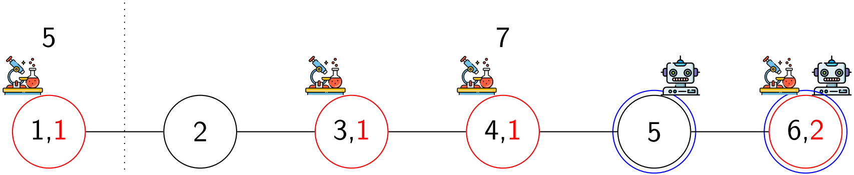
2-Partition Algorithm

- For two robots, R_L and R_R, starting on vertices sv_L and sv_R, we wish to split the tasks into two where R_L completes $t_1, ..., t_q$ and R_R does $t_{q+1}, ..., t_m$.
- We determine the value of q by finding the value which minimises

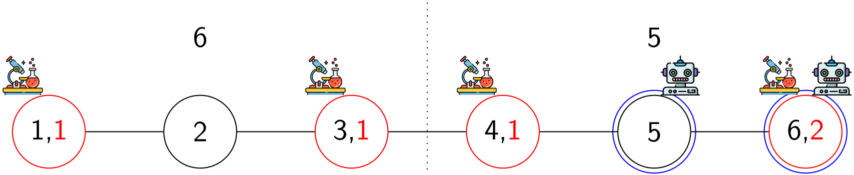
$$\max(C_1(P_{\max(sv_L, i_q)}, (t_1, , \ldots, t_q), sv_L), C_1(P_{1,\min(i_{q+1}, sv_R), m}, (t_{q+1}, \ldots, t_m), sv_R))$$

.

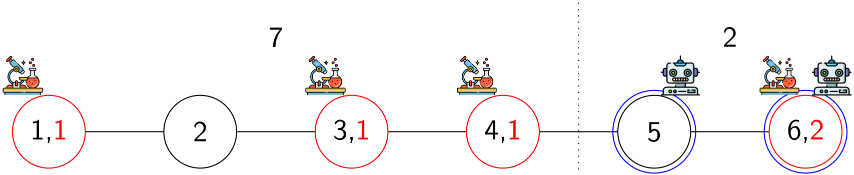
Partition Example



Partition Example



Partition Example



k-Partition Algorithm

Sketch of the $O(kmn)$ dynamic programming algorithm:

- Let $S[c, \ell]$ be the time taken for the first c robots to complete the first ℓ tasks.

k-Partition Algorithm

Sketch of the $O(kmn)$ dynamic programming algorithm:

- Let $S[c, \ell]$ be the time taken for the first c robots to complete the first ℓ tasks.
- $S[1, \ell] = C_1(P, (t_1, \ldots, t_\ell), sv_1)$.

k-Partition Algorithm

Sketch of the $O(kmn)$ dynamic programming algorithm:

- Let $S[c, \ell]$ be the time taken for the first c robots to complete the first ℓ tasks.
- $S[1, \ell] = C_1(P, (t_1, \ldots, t_\ell), sv_1)$.
- $S[c, \ell] := \min_{r \in [1, \ell]} \max(C_1(P, (t_{r+1}, \ldots, t_\ell), sv_c), S[c-1, r])$

k-Partition Algorithm

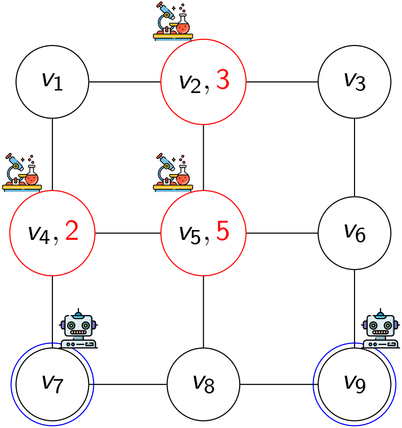
Sketch of the $O(kmn)$ dynamic programming algorithm:

- Let $S[c, \ell]$ be the time taken for the first c robots to complete the first ℓ tasks.
- $S[1, \ell] = C_1(P, (t_1, \dots, t_\ell), sv_1)$.
- $S[c, \ell] := \min_{r \in [1,\ell]} \max(C_1(P, (t_{r+1}, \dots, t_\ell), sv_c), S[c-1, r])$
- When $k = 2$ this is the same as the 2-Partition Algorithm.

Contents

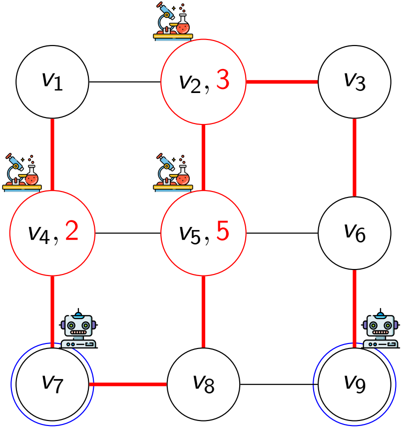
Grid graphs

A naïve way of approximating $k-$ Robot Scheduling on a grid is by finding an Hamiltonian Path and executing the partition algorithm on that path.



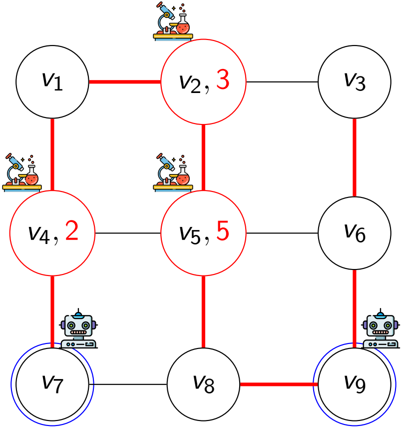
Grid graphs

A naïve way of approximating $k-$ ROBOT SCHEDULING on a grid is by finding an Hamiltonian Path and executing the partition algorithm on that path.



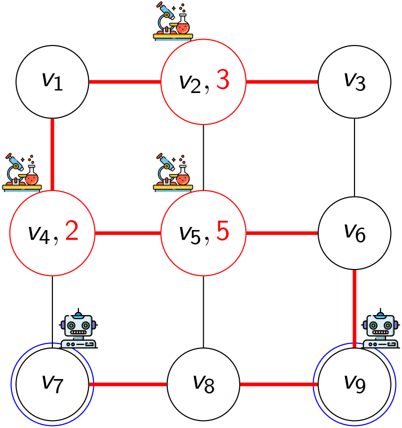
Grid graphs

A naïve way of approximating k – ROBOT SCHEDULING on a grid is by finding an Hamiltonian Path and executing the partition algorithm on that path.



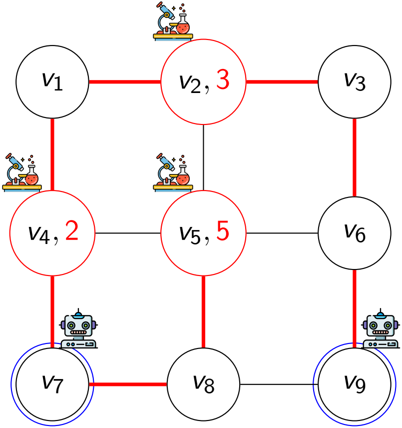
Grid graphs

A naïve way of approximating $k - \textsc{Robot Scheduling}$ on a grid is by finding an Hamiltonian Path and executing the partition algorithm on that path.



Grid graphs

A naïve way of approximating $k-$ Robot Scheduling on a grid is by finding an Hamiltonian Path and executing the partition algorithm on that path.



Algorithm for Cycles

We then adapt the partition algorithm for use on cycles by "*flattening*" the cycle.

Algorithm for Cycles

We then adapt the partition algorithm for use on cycles by *"flattening"* the cycle.

Lemma

Given an instance of k-ROBOT SCHEDULING *on a cycle* $G = (V, E)$ *with a set of equal duration tasks* $T = \{t_1, \ldots, t_m\}$ *and robots* $r_1, \ldots, r_k$*, there exists a fastest collision-free task-completing schedule* C *such that there exists some edge* $e \in E$ *that is not traversed by any robot in* C*.*

Sketch Proof

- We assume the contrary for contradiction.

Sketch Proof

- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1(mod\ k)}$ must share some path $v_x, ..., v_{y(mod\ n)}$ for $y \geq x$ where $v_{y(mod\ n)}$ contains task $t_{j_{i+1}}$.

Sketch Proof

- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1(mod\ k)}$ must share some path $v_x, ..., v_{y(mod\ n)}$ for $y \geq x$ where $v_{y(mod\ n)}$ contains task $t_{j_{i+1}}$.
- If for all i, r_i instead completes task t_{j_i} the total travel time would decrease.

Sketch Proof

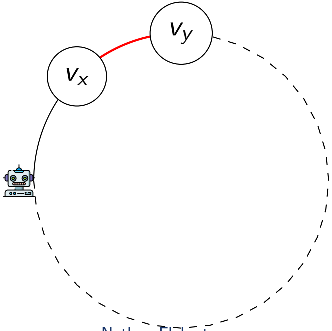
- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1(mod\ k)}$ must share some path $v_x, ..., v_{y(mod\ n)}$ for $y \geq x$ where $v_{y(mod\ n)}$ contains task $t_{j_{i+1}}$.
- If for all i, r_i instead completes task t_{j_i} the total travel time would decrease.
- Therefore the solution we started with was not optimal - a contradiction.

Sketch Proof

- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1(mod\ k)}$ must share some path $v_x, ..., v_{y(mod\ n)}$ for $y \geq x$ where $v_{y(mod\ n)}$ contains task $t_{j_{i+1}}$.
- If for all i, r_i instead completes task t_{j_i} the total travel time would decrease.
- Therefore the solution we started with was not optimal - a contradiction.

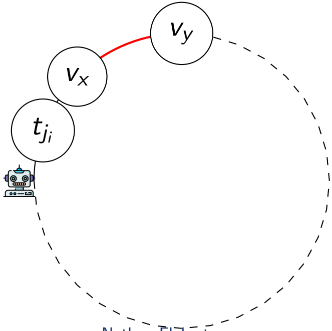
Sketch Proof

- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1(mod\ k)}$ must share some path $v_x, ..., v_{y(mod\ n)}$ for $y \geq x$ where $v_{y(mod\ n)}$ contains task $t_{j_{i+1}}$.
- If for all i, r_i instead completes task t_{j_i} the total travel time would decrease.
- Therefore the solution we started with was not optimal - a contradiction.



Sketch Proof

- We assume the contrary for contradiction.
- For all i r_i and $r_{i+1 (mod\ k)}$ must share some path $v_x, ..., v_{y (mod\ n)}$ for $y \geq x$ where $v_{y (mod\ n)}$ contains task $t_{j_{i+1}}$.
- If for all i, r_i instead completes task t_{j_i} the total travel time would decrease.
- Therefore the solution we started with was not optimal - a contradiction.



Algorithm for Cycles

We then adapt the partition algorithm for use on cycles by *"flattening"* the cycle.

Lemma

Given an instance of k-ROBOT SCHEDULING on a cycle $G = (V, E)$ with a set of equal duration tasks $T = \{t_1, \ldots, t_m\}$ and robots $r_1, \ldots, r_k$, there exists a fastest collision-free task-completing schedule C such that there exists some edge $e \in E$ that is not traversed by any robot in C.

The idea of our algorithm on a cycle is to iterate through all $e \in E$ such that we minimise $S[k, m]$ when applying the partition algorithm on $P = (V, E \setminus \{e\})$, this takes time $O(kmn^2)$.

Contents

Problem Formulation - Variables

- $x_{r,v,t} \in \{0, 1\}$ is 1 if robot r is at vertex v at timestep t
- $TC_{i,t} \in \{0, 1\}$ is 1 if task i is complete by timestep t
- $AC_t \in \{0, 1\}$ is 1 on the smallest timestep t where all tasks are complete.

Problem Formulation

$$\text{Maximise} \sum_{t\in[\tau]} AC_t$$

Subject To:

$$\sum_{v\in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau] \tag{1}$$

$$x_{r,v,t} \leq \sum_{N(v)\cup\{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2,\tau] \tag{2}$$

$$\sum_{r\in[k]} x_{r,v,t} \leq 1 \qquad \forall v \in V, t \in [\tau] \tag{3}$$

$$x_{r,v,t} + x_{r,v',t-1} + x_{r',v,t-1} + x_{r',v',t} \leq 3 \qquad \forall r \in [k], r' \in [k] \setminus \{r\}, (v,v') \in E, t \in [\tau] \tag{4}$$

$$TC_{i,t} \leq TC_{i,t-1} + \max_{r\in[k]} \left(\sum_{j\in[t-d_i,t]} x_{r,v_i,j}/d_i \right) \qquad \forall i \in [T], t \in [\tau] \tag{5}$$

$$TC_{i,t} \geq TC_{i,t-1} \qquad \forall i \in [T], t \in [\tau] \tag{6}$$

$$AC_t \leq \sum_{i\in[|T|]} TC_{i,t}/|T| \qquad \forall t \in [\tau] \tag{7}$$

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

- Ensures robots only move to adjacent vertices

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

- Ensures robots only move to adjacent vertices

$$\sum_{r \in [k]} x_{r,v,t} \leq 1 \qquad \forall v \in V, t \in [\tau]$$

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

- Ensures robots only move to adjacent vertices

$$\sum_{r \in [k]} x_{r,v,t} \leq 1 \qquad \forall v \in V, t \in [\tau]$$

- Ensures no collisions on vertices

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

- Ensures robots only move to adjacent vertices

$$\sum_{r \in [k]} x_{r,v,t} \leq 1 \qquad \forall v \in V, t \in [\tau]$$

- Ensures no collisions on vertices

$$x_{r,v,t} + x_{r,v',t-1} + x_{r',v,t-1} + x_{r',v',t} \leq 3 \qquad \forall (v, v') \in E, \forall r, \forall t.$$

Integer Program Constraints

$$\sum_{v \in V} x_{r,v,t} = 1 \qquad \forall r \in [k], t \in [\tau].$$

- Ensures each robot exists on every timestep

$$x_{r,v,t} \leq \sum_{N(v) \cup \{v\}} x_{r,v,t-1} \qquad \forall r \in [k], t \in [2, \tau]$$

- Ensures robots only move to adjacent vertices

$$\sum_{r \in [k]} x_{r,v,t} \leq 1 \qquad \forall v \in V, t \in [\tau]$$

- Ensures no collisions on vertices

$$x_{r,v,t} + x_{r,v',t-1} + x_{r',v,t-1} + x_{r',v',t} \leq 3 \qquad \forall (v, v') \in E, \forall r, \forall t.$$

- Ensures no collisions on edges

Integer Programming Constraints

$$TC_{i,t} \leq TC_{i,t-1} + \max_{r\in[k]}\left(\sum_{j\in[t-d_i,t]} x_{r,v_i,j}/d_i\right) \quad \forall i \in [T], t \in [\tau]$$

- Handles task completion.

$$TC_{i,t} \geq TC_{i,t-1} \quad \forall i \in [T], t \in [\tau]$$

- Ensures a task which is complete remains complete.

$$AC_t \leq \sum_{i\in[|T|]} TC_{i,t}/|T| \quad \forall t \in [\tau]$$

- Ensures AC is 1 iff all tasks have been completed

Contents

Partition Algorithm vs Gurobi Optimizer

- We generated 12,767,374 instances of Robot Scheduling.

Partition Algorithm vs Gurobi Optimizer

- We generated 12,767,374 instances of Robot Scheduling.
- For each we ran the Gurobi Optimizer on the integer programming formulation as well as the partition algorithm.

Partition Algorithm vs Gurobi Optimizer

- We generated 12,767,374 instances of Robot Scheduling.
- For each we ran the Gurobi Optimizer on the integer programming formulation as well as the partition algorithm.
- It produced an optimal solution in 11,061,661 cases (86.6%)

Partition Algorithm vs Gurobi Optimizer

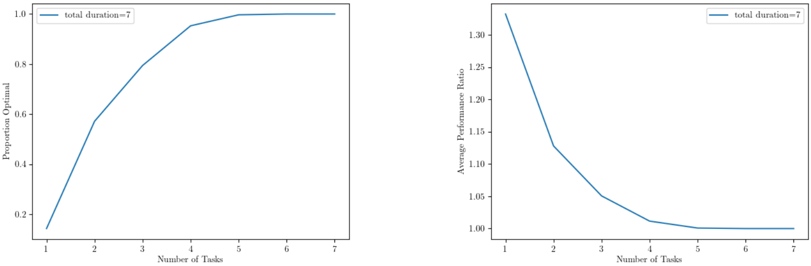


Figure 1: Comparison of the performance ratio of the Partition Algorithm to the theoretical optimal results given by our integer programming model. In this case, the tasks had a sum of durations of 7, while the number of tasks varies across the *x*-axis.

Experimental Results

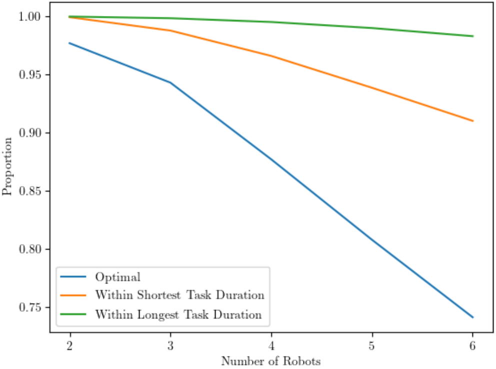


Figure 2: Comparison of proportion of output from the partition algorithm with optimal timespan, and timespan within the size of a single task from optimal for fixed $n = 7$. The plot shows the proportion decreasing as the number of robots increase

Runtime

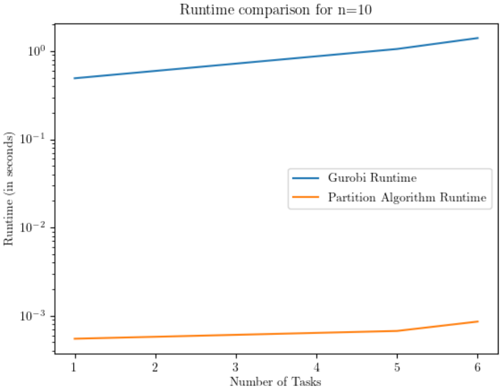


Figure 3: logplot showing average runtime of Gurobi solving the linear program and the partition algorithm solving the same instances.

Gurobi vs Partition Algorithm for Grid Graphs

Gurobi vs Partition Algorithm for Grid Graphs

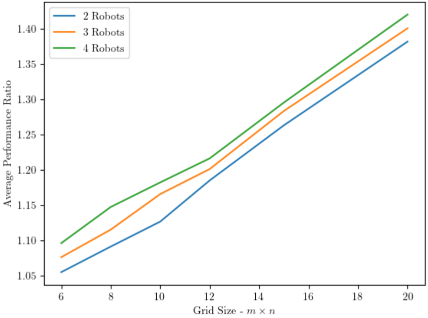


Figure 4: A plot showing the average performance ratio over all generated instances increasing as the grid size increases for 2,3 and 4 robots. In general Performance ratio is lower for fewer robots.

Gurobi vs Partition Algorithm for Grid Graphs

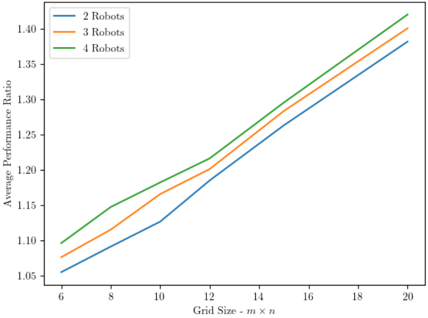


Figure 4: A plot showing the average performance ratio over all generated instances increasing as the grid size increases for 2,3 and 4 robots. In general Performance ratio is lower for fewer robots.

Overall - partitioning produced an optimal result 66% of the time.

Contents

Thank You!