Brief Announcement: Collision-Free Robot Scheduling[*]

Duncan Adamson
Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK

Nathan Flaherty
Leverhulme Research Centre for Functional Materials Design, University of Liverpool, UK

Igor Potapov
Department of Computer Science, University Of Liverpool, UK

Paul G. Spirakis
Department of Computer Science, University Of Liverpool, UK

Abstract

Robots are becoming an increasingly common part of scientific work within laboratory environments. In this paper, we investigate the problem of designing *schedules* for completing a set of tasks at fixed locations with multiple robots in a laboratory. We represent the laboratory as a graph with tasks placed on fixed vertices and robots represented as agents, with the constraint that no two robots may occupy the same vertex, or traverse the same edge, at the same time. Each schedule is partitioned into a set of timesteps, corresponding to a walk through the graph (allowing for a robot to wait at a vertex to complete a task), with each timestep taking time equal to the time for a robot to move from one vertex to another and each task taking some given number of timesteps during the completion of which a robot must stay at the vertex containing the task. The goal is to determine a set of schedules, with one schedule for each robot, minimising the number of timesteps taken by the schedule taking the greatest number of timesteps within the set of schedules.

We show that the problem of finding a task-fulfilling schedule in at most L timesteps is NP-complete for many simple classes of graphs. Explicitly, we provide this result for complete graphs, bipartite graphs, star graphs, and planar graphs. Finally, we provide positive results for line graphs, showing that we can find an optimal set of schedules for k robots completing m tasks of equal length of a path of length n in $O(kmn)$ time, and a k-approximation when the length of the tasks is unbounded.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Graph Exploration, Scheduling, NP-Completeness, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.22

Related Version *Full Version*: https://arxiv.org/abs/2402.12019 [1]

1 Introduction

In this paper, we are interested in the scheduling of robots within chemistry labs, motivated by a significant and expanding body of work concerning robotic chemists. Initial work on these systems focused on building robots performing reactions within fixed environments [4, 3], however recently Burger et al. [2] have presented a robot capable of moving within a laboratory and completing tasks throughout the space. The works of Burger et al. [2] and Liu et al. [5] provide the main motivation for this work, namely the problem of moving robots within a laboratory environment (as presented by Burger et al. [2]) while avoiding collisions (as investigated in the manufacturing context by Liu et al. [5]).

[*] A full version of this paper is available on arXiv [1]

2 Preliminaries

In this problem, we consider a set of agents, which we call *robots*, moving on a given graph $G = (V, E)$ and completing a set of tasks $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$. As mentioned in our introduction, this problem originates in the setting of lab spaces, particularly in the chemistry setting. As such, our definitions of robots and tasks are designed to mimic those found in real-world problems. We associate each task with a vertex on which it is located and the duration required to complete the task. We do not allow tasks to be moved by a robot, a task can only be completed by a single robot remaining at the station for the entire task duration, and any robot may complete any number of tasks, with no restrictions on which task a robot can complete. This requirement reflects the motivation from chemistry, where tasks reflect reactions that must be done within an exact time frame and at a fixed workstation.

Formally, we define a task t_i as a tuple (d_i, v_i) where d_i is the *duration* of the task, and v_i is the vertex at which the task is located. We use $|t_i|$ to denote the duration of the task t_i. In general, the reader may assume that for a graph $G = (V, E)$ containing the vertex set $V = \{v_1, v_2, \dots, v_n\}$, the notation i_t is used to denote the index of the vertex at which task $t = (d, v_{i_t})$ is located. This will be specified throughout the paper where relevant.

To complete tasks, we assign each robot a *schedule*, composed of an alternating sequence of walks and tasks. We note that each schedule can begin and end with either a walk and a walk, a walk and a task, a task and a walk, or a task and a task. We treat each schedule as a set of commands to the robot, directing it within a given time frame. In this way, we partition the schedule into a set of time steps, with each time step allowing a robot to move along one edge or complete some fraction of a task, with a task t requiring exactly $|t|$ time steps to complete. We call the time span of a schedule the total number of timesteps required to complete it. We denote the time span of the schedule C containing the walks $w_1, w_2, \dots, w_\ell$ and tasks $t_1, t_2, \dots, t_m$ by $|C| = \left(\sum_{i\in[1,\ell]} |w_i|\right) + \left(\sum_{j\in[1,m]} |t_j|\right)$. Given a walk w directly following the task t in the schedule C, we require that the first edge traversed in w begins at the vertex v_{i_t} on which t is located. Similarly, we require that the task t' following the walk w' in the schedule C is located on the last vertex in the last edge in w'.

The *walk representation* $\mathcal{W}(C)$ of a schedule C is an ordered sequence of edges formed by replacing the task $t_i = (d, v_i)$ in C with a walk of length $|t_i| = d$ consisting only of the edge (v_i, v_i), then concatenate the walks together in order. Note that $|\mathcal{W}(C)| = |C|$. For a given robot R assigned schedule C, in timestep j R is located on the vertex $v \in V$ that is the end vertex of the i^{th} edge in $\mathcal{W}(C)$, i.e. the vertex v such that $\mathcal{W}(C)[i] = (u, v)$. We require the first vertex in the walk representation of any schedule C assigned to robot R to be the *starting vertex* of R, i.e. some predetermined vertex representing where R starts on the graph. If the schedule C containing the task t is assigned to robot R, we say that t is *assigned* to R.

Given a set of schedules $\mathcal{C} = (C_1, C_2, \dots, C_k)$ for a set of k robots $R_1, R_2, \dots, R_k$, and set of tasks $\mathcal{T} = (t_1, t_2, \dots, t_m)$. we say that $\mathcal{C}$ is *task completing* if for every task $t \in \mathcal{T}$ there exists exactly one schedule C_i such that $t \in C_i$. We call $\mathcal{C}$ *collision-free* if there is no timestep where any pair of robots occupy the same vertex or traverse the same edge. Formally, $\mathcal{C}$ is collision-free if, for every C_i, C_j where $i \neq j$ and time-step $s \in [1, |C_i|]$, $\mathcal{W}(C_i)[s] = (v, u)$ and $\mathcal{W}(C_j)[s] = (v', u')$ satisfies $u \neq u'$ and $(v, u) \neq (u', v')$.

For the remainder of this paper, we assume every robot in the graph is assigned exactly 1 schedule. Given 2 sets of schedules $\mathcal{C}$ and $\mathcal{C}'$, we say $\mathcal{C}$ is *faster* than $\mathcal{C}'$ if $\max_{C_i\in\mathcal{C}} |C_i| < \max_{C'_j\in\mathcal{C}'} |C'_j|$.Given a graph $G = (V, E)$, a set of k robots $R_1, R_2, \dots, R_k$ starting on vertices $sv_1, sv_2, \dots, sv_k$, and a set of tasks $\mathcal{T}$, a *fastest* task-completing, collision-free set of k schedules is the set of schedules $\mathcal{C}$ such that any other set of task-completing, collision-free schedules is no faster than $\mathcal{C}$. Note that there may be multiple such sets of schedules.

▶ **Problem 1** (k-ROBOT SCHEDULING). *Given a graph $G = (V, E)$, set of k robots $R_1, R_2, \dots, R_k$ starting on vertices $sv_1, sv_2, \dots, sv_k$, and set of tasks $\mathcal{T}$, what is the fastest task-completing, collision-free set of k-schedules $\mathcal{C} = (C_1, C_2, \dots, C_k)$ such that C_i can be assigned to R_i, $\forall i \in [1, k]$?*

3 Results

Hardness Results

We have found that the ROBOT SCHEDULING problem is NP-Hard, and that hardness remains even when we restrict the class of graphs we consider, the known results are shown in Table 1.

Table 1 Our results for different graph classes and numbers, k, of robots.

Setting	Result
General graphs, $k \in \mathbb{N}$	NP-complete
Complete graphs, $k \geq 2$	NP-complete
Bipartite graphs, $k \geq 2$	NP-complete
Star graphs (and trees), $k \geq 2$	NP-complete
Planar graphs, $k \in \mathbb{N}$	NP-complete
Path graphs, with m tasks of equal duration, $k \in \mathbb{N}$	Optimal $O(kmn)$ time Algorithm (Theorem 7)
Path graphs, $k \in \mathbb{N}$	k-approximation Algorithm (Theorem 8)

Algorithmic Results for Path Graphs

1-Robot Scheduling on Path Graphs. In this section, we provide an algorithm for finding the optimal schedule for a single robot on a path. Corollary 3 shows that the time needed to complete the fastest schedule can be computed via a closed-form expression.

1-Robot Scheduling Algorithm. Let P be a path graph of length n, let $T = (t_1, t_2, \dots, t_m)$ be a set of tasks, and let R be the single robot starting on vertex $sv = v_{i_s}$. We assume, without loss of generality, that t_j is located on v_{i_j} such that v_{i_j} is left of $v_{i_{j+1}}$, i.e. $\forall j \in [1, m-1], i_j < i_{j+1}$. Note that there may exist some task t_i located on sv without contradiction. Using this notation, the optimal schedule $\mathcal{C} = \{C\}$ is:

- $C = \{\ (v_s, v_{s+1}),\ (v_{s+1},\ v_{s+2}),\ \dots,\ (v_{i_m-1},\ v_{i_m}),\ t_m,\ (v_{i_m}, v_{i_m-1}),\ (v_{i_m-1}, v_{i_m-2}),\ \dots,\ (v_{i_{m+1}+1}, v + i_{m+1}),\ t_{m-1},\ \dots,\ (v_{i_1+1}, v_{i_1}),\ t_1\ \}$ if $|i_s - i_m| \leq |i_s - i_1|$.
- $C = \{\ (v_s, v_{s-1}),\ (v_{s-1}, v_{s-2}),\ \dots,\ (v_{i_1+1}, v_{i_1}),\ t_1,\ (v_{i_1}, v_{i_1+1}),\ (v_{i_1+1}, v_{i_2+2}),\ \dots,\ (v_{i_2-1}, v_{i_2}),\ t_2,\ \dots,\ (v_{i_m-1}, v_{i_m}),\ t_m\ \}$ if $|i_s - i_m| > |i_s - i_1|$.

▶ **Lemma 2.** *The fastest task-completing schedule for 1-ROBOT SCHEDULING on a path graph P of length n with m tasks $T = (t_1, t_2, \dots, t_m)$ located on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$, and a robot R starting on vertex v_s can be constructed in $O(n)$ time.*

▶ **Corollary 3.** *The fastest task-completing schedule for* 1-ROBOT SCHEDULING *on a path graph P of length n with m tasks $T = (t_1, t_2, \dots, t_m)$ located on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and a robot R starting on vertex v_s requires*

$$\min(|s - i_1|, |s - i_m|) + i_m - i_1 + \sum_{t \in T} t$$

timesteps.

2-Robot Scheduling on Path Graphs. We now move to 2-ROBOT SCHEDULING on a path. First, we provide a new algorithm generalising the above algorithm for 1-ROBOT SCHEDULING . Later, we further generalise this to k-ROBOT SCHEDULING on a path; however, it is valuable to consider 2-ROBOT SCHEDULING first, both to illuminate the main algorithmic ideas and to provide a base case for later inductive arguments. We start by providing an overview of our algorithm, which we call the *partition algorithm.*

The 2-Partition Algorithm. Let P be a path graph of length n, let $T = (t_1, t_2, \dots, t_m)$ be the set of tasks, and let R_L and R_R be the pair of robots starting on vertices $sv_L = v_{i_L}$ and $sv_R = v_{i_R}$ respectively. We call R_L the *left robot* and R_R the *right robot*, with the assumption that sv_L is left of sv_R. We denote by i_j the index of the vertex containing the task t_j, and assume that $i_j < i_{j+1}$, for every $j \in [1, n-1]$. For notation, let $C_1(P, T, sv)$ return the optimal schedule for a single robot starting at sv on the path P for completing the task set T.

We construct the schedule by partitioning the tasks into 2 sets, $T_L = (t_1, t_2, \dots, t_q)$ and $T_R = (t_{q+1}, t_{q+2}, \dots, t_m)$. We determine the value of q by finding the value which minimises $\max(|C_1(P_{1,\max(\ell, i_q}, (t_1, t_2, \dots, t_\ell), sv_L)|, |C_1(P_{\min(i_{q+1}, v_r), m}, (t_{q+1}, t_{q+2}, \dots, t_m), sv_R)|$. We will use $C_2(P, T, (sv_L, sv_R))$ to denote the schedule returned by this process.

▶ **Lemma 4.** *Given an instance of* 2-ROBOT SCHEDULING *on an n-length path P with a set of equal-length tasks $T = (t_1, t_2, \dots, t_m)$, and starting vertices $sv_L = v_{i_L}, sv_R = v_{i_R}$, for any schedule $\mathcal{C} = (C_\ell, C_r)$ where the rightmost task t_R assigned to the left robot is right of the leftmost T_L assigned to the right robot, there exists some schedule $\mathcal{C}' = (C'_\ell, C'_r)$ that takes no more time than $\mathcal{C}$ and does not contain any such tasks.*

▶ **Lemma 5.** *Given an instance of* 2-ROBOT SCHEDULING *on an n-length path P with a set of tasks $T = (t_1, t_2, \dots, t_m)$ where the duration of t_i is equal to the duration of t_j for every $i, j \in [1, m]$. Further, let sv_L and sv_R be the starting vertices of the robots. Then $C_2(P, T, (sv_L, sv_R))$ is a fastest set of schedules for this instance and can be found in $O(m)$ time.*

▶ **Theorem 6.** *Given an instance of* 2-ROBOT SCHEDULING *on an n-length path P, with a set of tasks $T = (t_1, t_2, \dots, t_m)$ and starting vertices sv_L and sv_R. Then $C_2(P, T, (sv_L, sv_R))$ is within a factor of 2 of the fastest set of schedules solving this instance.*

***k*-robots on the path.** Now, we generalise the 2 robots on a path instance to an arbitrary number. To do so, we build a dynamic programming algorithm based on the same principles as the previous partition algorithm.

The *k*-Partition Algorithm. Let P be a path of length n, $T = \{t_1, t_2, \dots, t_m\}$ be a set of tasks, and let $sv_1, sv_2, \dots, sv_k$ be the starting vertices of the robots $R_1, R_2, \dots, R_k$ respectively, with the assumption that R_i starts left of R_{i+1}, for every $i \in [1, k-1]$. Further, we denote by i_t the index such that v_{i_t} contains task t, and assume that $i_{t_j} < i_{t_{j+1}}$ (i.e.

task t_j is left of t_{j+1}) for every $j \in [1, m-1]$. We construct a $k \times m$ table S, with $S[c, \ell]$ containing the time required to complete the fastest collision-free schedule completing tasks $t_1, t_2, \dots, t_\ell$ with robots $R_1, R_2, \dots, R_c$.

First, observe that $S[1, \ell]$ can be computed, for every $\ell \in [1, m]$, in $O(m)$ time. Now, assuming the value of $S[c-1, \ell]$ has been computed for every $\ell \in [1, m]$, the value of $S[c, r]$ is computed by finding the value r' such that $\max(|C_1(P, (t_{r'+1}, t_{r'+2}, \dots, t_r))|, S[c-1, r'])$ is minimised, formally $S[c, r] = \min_{r' \in [1,r]} \max(|C_1(P, (t_{r'+1}, t_{r'+2}, \dots, t_r))|, S[c-1, r'])$. Letting $\mathcal{S}$ be an auxiliary table such that $\mathcal{S}[c, \ell]$ contains the schedule corresponding to the time given in $S[c, \ell]$, a task-completing collision-free schedule for the k-Robot Scheduling instance is given in $\mathcal{S}[k, m]$.

Let $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$ return the schedule determined by this table. Note that for $S_2(P, T, (sv_1, sv_2))$, this becomes equivalent to the 2-partition algorithm.

▶ **Theorem 7.** *Given an instance of k-Robot Scheduling on a path $P = (V, E)$ with equal duration tasks $T = (t_1, t_2, \dots, t_m)$ on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and k robots $R_1, R_2, \dots, R_k$ starting at $sv_1, sv_2, \dots, sv_k = v_{j_1}, v_{j_2}, \dots, v_{j_k}$, there are no schedules taking less time than the schedule returned by $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$. Further, this schedule can be found in $O(kmn)$ time.*

▶ **Theorem 8.** *Given an instance of k-Robot Scheduling on a path $P = (V, E)$ with tasks $T = (t_1, t_2, \dots, t_m)$ on vertices $v_{i_1}, v_{i_2}, \dots, v_{i_m}$ and k robots $R_1, R_2, \dots, R_k$ starting at $sv_1, sv_2, \dots, sv_k = v_{j_1}, v_{j_2}, \dots, v_{j_k}$, the schedule returned by $S_k(P, T, (sv_1, sv_2, \dots, sv_k))$ is no more than a factor of k slower than the optimal. Further, this schedule can be found in $O(km^2)$ time.*

4 Conclusion

We have shown that our definition of k-Robot Scheduling is hard, even on highly constrained classes of graphs while being solvable for path graphs with equal-length tasks and approximable for tasks of any length. While these results paint a strong picture of the complexity of this problem, we are left with several open questions. The most direct is as to whether our approximation algorithm for path graphs can be improved or if an optimal algorithm can be found. We conjecture that a polynomial time algorithm exists for this setting; however, at present, no such algorithm has been found. The second natural direction is to look at the remaining classes of graphs that have not been covered by our existing results. The most obvious of these are cycles, which, while closely related to paths, can not be solved by naive application of our current tools. While it seems likely that similar optimality and approximation results can be found, these are currently open problems.

References

1 Duncan Adamson, Nathan Flaherty, Igor Potapov, and Paul Spirakis. Collision-free robot scheduling, 2024. `arXiv:2402.12019`.

2 B. Burger, P. M. Maffettone, V. V. Gusev, C. M. Aitchison, Y. Bai, X. Wang, X. Li, B. M. Alston, B. Li, R. Clowes, et al. A mobile robotic chemist. *Nature*, 583(7815):237–241, 2020.

3 R. D. King. Rise of the robo scientists. *Scientific American*, 304(1):72–77, 2011.

4 J. Li, S. G. Ballmer, E. P. Gllis, S. Fujii, M. J. Schmidt, A. M. E. Palazzolo, J. W. Lehmann, G. F. Morehouse, and M. D. Burke. Synthesis of many different types of organic small molecules using one automated process. *Science*, 347(6227):1221–1226, 2015.

5 S. Liu, J. Shen, W. Tian, J. Lin, P. Li, and B. Li. Balanced task allocation and collision-free scheduling of multi-robot systems in large spacecraft structure manufacturing. *Robotics and Autonomous Systems*, 159:104289, 2023.