Interleaving Scheduling and Motion Planning with Incremental Learning of Symbolic Space-Time Motion Abstractions

Elisa Tosello[1], **Arthur Bit-Monnot**[2], **Davide Lusuardi**[1], **Alessandro Valentini**[1], **Andrea Micheli**[1]

[1]Fondazione Bruno Kessler, Trento, Italy
[2]LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
etosello@fbk.eu, abitmonnot@laas.fr, lusuardi@fbk.eu, alvalentini@fbk.eu, amicheli@fbk.eu

Abstract

Task and Motion Planning combines high-level task sequencing (*what* to do) with low-level motion planning (*how* to do it) to generate feasible, collision-free execution plans. However, in many real-world domains, such as automated warehouses, tasks are predefined, shifting the challenge to *if*, *when*, and *how* to execute them safely and efficiently under resource, time and motion constraints. In this paper, we formalize this as the Scheduling and Motion Planning problem for multi-object navigation in shared workspaces. We propose a novel solution framework that interleaves off-the-shelf schedulers and motion planners in an incremental learning loop. The scheduler generates candidate plans, while the motion planner checks feasibility and returns symbolic feedback, i.e., spatial conflicts and timing adjustments, to guide the scheduler towards motion-feasible solutions. We validate our proposal on logistics and job-shop scheduling benchmarks augmented with motion tasks, using state-of-the-art schedulers and sampling-based motion planners. Our results show the effectiveness of our framework in generating valid plans under complex temporal and spatial constraints, where synchronized motion is critical.

Code — https://github.com/fbk-pso/tampest.git
Extended version — http://arxiv.org/abs/2603.10651

Introduction

Task and Motion Planning (TAMP) is the problem of combining high-level decision-making, i.e., deciding which tasks to perform, with low-level motion planning, i.e., ensuring that these tasks are carried out via physically feasible, collision-free, trajectories (Garrett et al. 2021; Dantam 2020). This integration is critical in domains where symbolic actions must be grounded in real-world geometry and dynamics, including robotics and automated manufacturing. While traditional TAMP focuses on *what* to do and *how* to execute it, many real-world scenarios assume a predetermined set of tasks, shifting the challenge to *if* and *when* to perform them. This reframes the problem as scheduling under resource, precedence, and timing constraints. For example, in an automated warehouse, mobile robots must transport goods from storage to delivery stations. With tasks such as move, pick, and drop predetermined, the problem becomes (i) deciding the order and timing for each robot (*scheduling*) and (ii) computing dynamically and kinematically feasible, collision-free trajectories to execute these tasks in the continuous physical environment (*motion planning*) (see Figure 1). The interplay between space and time is crucial: motion planning must ensure not only spatial feasibility but also precise temporal coordination among agents, which may need to wait, sequence, or synchronize their movements to safely share constrained regions (e.g., narrow passages) and to prevent conflicts or deadlocks. Unlike discrete path-finding abstractions, this requires reasoning directly in continuous configuration spaces with explicit kinodynamic constraints. We refer to this integrated challenge as the Scheduling and Motion Planning (SAMP) problem.

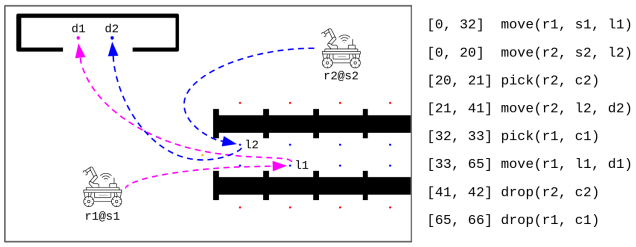

Figure 1: SAMP schedule of robots (r_1, r_2) executing move–pick–drop tasks. Intervals $[t_i, t_j]$ denote start/end times. Robots move from (s_1, s_2) to pick (c_1, c_2) at (l_1, l_2) and deliver them to (d_1, d_2), parallelizing when possible.

In this paper, we formally define the SAMP problem for multiple objects navigating in a shared workspace and propose a framework that addresses SAMP by interleaving off-the-shelf schedulers and motion planners in an incremental learning loop of symbolic motion abstractions. The scheduler generates candidate schedules without considering the underlying motion. The motion planner, treated as a black-box, evaluates them accounting for the kinematics and dynamics of the objects involved, and returns either feasible trajectories or symbolic refinements to help the Scheduler finding a valid solution. Feedbacks include geometric refinements, highlighting spatial conflicts (i.e., unreachable goals and blocking obstacles), and temporal refinements, adjusting activity durations or requesting delays to enable feasi-

ble motion synchronization. By incrementally learning such symbolic motion abstractions, our framework does not need to fully ground all constraints in advance, enabling better scalability in complex and dynamic domains.

We provide constraint formulations either via fluents (conditions and effects) or through precedence and resource constraints, enabling different schedulers (e.g., Aries (Bit-Monnot 2023) and OR-Tools (Perron and Didier 2025)) to be combined with (instrumented) motion planners (ST-RRT* (Grothe et al. 2022)) under optimal/non-optimal, with/without fluents settings. We evaluate these combinations on classical logistics and job-shop benchmarks (Taillard 1993) augmented with navigation tasks. Results show that our framework produces valid, eventually optimal, synchronized plans under temporal and spatial constraints.

Related Work

Several studies address TAMP: from domain-specific solutions (Garrett, Lozano-Pérez, and Kaelbling 2018; Toussaint 2015) to general frameworks (Dantam et al. 2016; Garrett, Lozano-Pérez, and Kaelbling 2020; Cashmore et al. 2015; Tosello, Valentini, and Micheli 2024). While effective in combining symbolic and geometric planning, they neglect the temporal dimension, crucial when dealing with multi-agent scenarios. This gap led to works on temporal coordination, from motion-control strategies (Pecora et al. 2018) to optimal multi-agent planning (Faroni et al. 2024) and Temporal Task and Motion Planning (Tosello, Valentini, and Micheli 2025). However, they overlook cases where activities are predefined and the problem shifts toward scheduling, focusing on the temporal allocation and synchronization of known tasks rather than their dynamic generation.

This motivates Simultaneous Task and Motion Scheduling (STAAMS), which assigns and orders high-level actions while accounting for motion-level constraints. Although STAAMS combines Constraint Programming and Motion Planning, most existing approaches are tailored to specific domains, e.g., dual-arm manipulation (Zanlongo et al. 2021), traffic coordination (Leet et al. 2023), and assembly lines (Neville, Chernova, and Ravichandar 2023). Some methods either rely on precomputed trajectories or perform online motion planning but use the scheduler primarily to enforce static constraints (e.g., collision avoidance or kinematic limits), often without temporal feedback (Behrens, Stepanova, and Babuska 2020) or tight scheduling-motion integration. Such approaches could serve as a reasonable baseline for our method, but high customization and limited reproducibility hinder direct comparison. We therefore start from benchmarks that may seem simpler (e.g., 2D navigation) but are generic and, importantly, require tightly interleaving scheduling with motion-level feasibility over time.

To overcome the high computational cost of reasoning over continuous state spaces with explicit modeling of agent kinematics and dynamics, one may shift to Multi-Agent Path Finding (MAPF) (Stern et al. 2021). However, classical MAPF assumes point-like agents on discrete spaces, neglecting geometry, kinematics, and dynamics. Extensions for large agents (Li et al. 2019; Li 2024), kinematic constraints (Hönig et al. 2017; Ma et al. 2019) or temporal dependencies (Jiang, Lin, and Li 2025) partially address these limitations, but rely on discretization, while continuous-time MAPF (Andreychuk et al. 2019) still omits full kinodynamic modeling. In this paper, we remain focused on the continuous state space with kinodynamic constraints, leaving the combination with MAPF in discrete space as a new problem to be explored in the future.

Problem Statement

Consider a fleet of mobile robots moving products from shelves to a delivery station (see Figure 1). Robots are movable objects whose configurations change during tasks, and the schedule defines when they move, pick items, or return, along with the trajectories and control laws enabling these actions. Activities can be optional, allowing the scheduler to skip them if they are desirable but non-essential (their inclusion improves plan quality) or to select among mutually exclusive alternatives (e.g., a delivery may be made to one of several possible locations). Motions must be geometrically feasible, avoiding static (walls, shelves) and dynamic (other robots) obstacles, and temporally feasible, satisfying the timing scheduling constraints. We call this problem SAMP and formalize it here, starting from the concept of Optional Scheduling (OS).

Definition 1. *An **Optional Scheduling** (OS) problem (with fluents and effects) is a tuple $\phi = \langle \mathcal{V}, \mathcal{A}, \mathcal{R}, \mathcal{C}, eff, init \rangle$:*

- *$\mathcal{V} = \{f_1, .., f_k\}$ is a finite set of fluents $f \in \mathcal{V}$, each with a finite domain Dom(f).*
- *$\mathcal{A}$ is the set of mandatory and optional activities, where optional ones may be excluded. Each $a \in \mathcal{A}$ has a duration bound $[lb_a, ub_a]$, with $lb_a, ub_a \in \mathbb{N}^+$ being the lower and upper bounds, respectively. We denote a.present, a.start, and a.end the variables for presence, start, and end times of a.*
- *$\mathcal{R}$ is the set of available resources, with each resource $r \in \mathcal{R}$ having availability $\lambda_r \in \mathbb{N}^+$.*
- *$\mathcal{C} = \mathcal{C}_f \sqcup \mathcal{C}_t \sqcup \mathcal{C}_r$ is the set of constraints, divided into:*

– *$\mathcal{C}_f$ the set of fluent constraints of the form $([\kappa_1, \kappa_2], a, f = v)$, where κ_i is an expression $a.start + k$ or $a.end - k$, $k \in \mathbb{N}$, with $a \in \mathcal{A}$, $f \in \mathcal{V}$, $v \in$ Dom(f).*

– *$\mathcal{C}_t$ the set of constraints enforcing precedence relations and temporal ordering between activities. They are arbitrary Boolean combination of atoms of the form:*

∗ *a.present for some $a \in \mathcal{A}$, where a.present is True iff the activity is scheduled (i.e. appears in the solution);*

∗ *$\kappa_1 - \kappa_2 \leq \Delta t$, with $\kappa_i \in \{a_j.start, a_j.end\}$ for $a_j \in \mathcal{A}$, and $\Delta t \in \mathbb{Z}$ being the maximum delay between them.*

– *$\mathcal{C}_r$ the set of constraints on resource usage, where activity $a \in \mathcal{A}$ uses γ_r^a units of resource r over $[a.start, a.end]$.*

- *$eff : \mathcal{A} \to \mathcal{E}$ maps an activity to its timed effects on fluents. Each element of $eff(a)$ is of the form $(\kappa, f := v)$ with κ being either $a.start + k$ or $a.end - k$ with $k \in \mathbb{N}$, $f \in \mathcal{V}$ and $v \in$ Dom(f); it indicates that at timing κ (relative to a), fluent f is assigned to value v due to activity a.*
- *init is the initial fluent state, which assigns a value $init(f) \in Dom(f)$ to each $f \in V$ at time 0.*

The schedule solving an OS problem is defined as follows.

Definition 2. *A **schedule** ρ solving ϕ is a tuple $\langle p, s, e\rangle$:*
- $p : \mathcal{A} \to \{\top, \bot\}$ *indicates if an activity is present,*
- $s : \mathcal{A} \to \mathbb{N}$ *indicates the starting time of an activity, and*
- $e : \mathcal{A} \to \mathbb{N}$ *indicates the ending time.*

We now define the semantics. In particular, for an expression κ of the form $a_i.start+k$ (resp. $a_i.end-k$), its evaluation in ρ is $\rho(\kappa) = s(a_i)+k$ (resp. $\rho(\kappa) = e(a_i)-k$). A schedule ρ is **non-conflicting** if there exists no time $t \in \mathbb{N}$ and activities $a_1, a_2 \in \mathcal{A}$ with effects $(\kappa_1, f_1 := v_1) \in \mathit{eff}(a_1), (\kappa_2, f_2 := v_2) \in \mathit{eff}(a_2)$ such that $\rho(\kappa_1) = \rho(\kappa_2) \wedge f_1 = f_2$, i.e., two effects on the same fluent never overlap, as required by the PDDL semantics (Fox and Long 2003). To define the validity of a non-conflicting schedule, we first introduce a function tracking fluent changes over time.

Definition 3. *For a non-conflicting schedule $\rho = \langle p, s, e\rangle$, the **evaluation function** $\xi_\rho : \mathcal{V} \times \mathbb{N} \to \bigcup_{f\in\mathcal{V}} Dom(f)$ maps a fluent $f \in \mathcal{V}$ and a time point $t \in \mathbb{N}$ to the value of f at time t under ρ. It is defined as:*

$$\xi_\rho(f,t) = \begin{cases} \mathit{init}(f) & \text{if } t = 0 \\ v & \text{if } \exists a \in \mathcal{A} \text{ s.t. } p(a) \wedge \\ & (\kappa, f := v) \in \mathit{eff}(a) \wedge \rho(\kappa) = t \\ \xi_\rho(f, t-1) & \text{otherwise} \end{cases}$$

Intuitively, $\xi_\rho(f,t)$ gives the value of fluent *f* at time *t*, set by the most recent activity *a* ending by *t* that updates *f*. If none exists, it returns the initial value of *f*.
Validity and optimality of ρ can then be defined as follows.

Definition 4. *Let the set of activities active at time t under schedule ρ be $\mathcal{A}^t_\rho = \{a \in \mathcal{A} \mid p(a) \wedge s(a) \le t \le e(a)\}$. A schedule ρ is **valid** for an OS ϕ if it is non-conflicting and the following conditions hold.*

1. *$\forall a \in \mathcal{A}$, $\neg p(a) \vee e(a) - s(a) \in [lb_a, ub_a]$, i.e., if the activity is present, its duration satisfies the duration bounds.*
2. *For each $r \in \mathcal{R}$, $\forall t \in \mathbb{N}. \sum_{a\in\mathcal{A}^t_\rho} \gamma^a_r \le \lambda_r$, i.e., total resource demand at any time does not exceed availability.*
3. *Constraints in $\mathcal{C}_t$ are satisfied using standard Boolean logic, with the value of atoms defined as follows:*
   - *$a.present$ is true iff $p(a)$ (presence);*
   - *$\kappa_1 - \kappa_2 \le \Delta t$ iff $\rho(\kappa_1) - \rho(\kappa_2) \le \Delta t$ (precedence).*
4. *Constraints in $\mathcal{C}_f$ are satisfied: given $([\kappa_1, \kappa_2], a, f := v) \in \mathcal{C}_f$, either $\neg p(a)$ or $\forall t \in [\rho(\kappa_1), \rho(\kappa_2)]$, $\xi_\rho(f,t) = v$.*

Definition 5. *Given an OS ϕ, a set of schedules $\mathcal{S}$, and a function $opt : \mathcal{S} \to \mathbb{R}$ to be minimized, $\rho \in \mathcal{S}$ is **optimal** for ϕ if it is valid for ϕ and for every other valid schedule $\rho' \neq \rho \in \mathcal{S}$, $opt(\rho) \le opt(\rho')$.*

We now incorporate motion activities, i.e., tasks involving object movement subject to motion constraints.

Definition 6. *A **Scheduling and Motion Planning** (SAMP) problem is a tuple $\psi = \langle \phi, \mathcal{O}, \mathcal{W}, \mathcal{Q}, u, i, mc\rangle$, where:*
- *$\phi = \langle \mathcal{V}, \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathit{eff}, \mathit{init}\rangle$ is an OS as per Definition 1.*
- *$\mathcal{O} \subseteq \mathcal{R}$ is a set of movable objects, where each object $o \in \mathcal{O}$ is characterized by a geometric model g_o and a control model u_o, with $\lambda_o = 1$ (only one is available).*
- *$\mathcal{W} \subseteq \mathbb{R}^N$ (N = 2 or N = 3) is the workspace, i.e., the volume of reachable end-points for objects in $\mathcal{O}$. $\mathcal{W}_{free}$ is the portion of $\mathcal{W}$ that is free from fixed obstacles.*
- *$\mathcal{Q}$ is the configuration space, with $\mathcal{Q}_o \subseteq \mathcal{Q}$ the subset of $\mathcal{Q}$ representing the configurations that $o \in \mathcal{O}$ may assume given its motion model. $occ(o,q) \subseteq \mathcal{W}_{free}$ is the set of points in $\mathcal{W}_{free}$ occupied by o when in $q \in \mathcal{Q}_o$.*
- *$i : \mathcal{O} \to \mathcal{Q}_o$ is a function that assigns to a movable object $o \in \mathcal{O}$ an initial configuration $q \in \mathcal{Q}_o$.*
- *$mc : \mathcal{A} \to$ mot associates an activity a to its motion constraint, where a motion constraint can be $\bot$ (indicating no motion constraint) or a tuple $\langle o_a, q^S_a, q^G_a\rangle$, where:*
  - *$o_a \in \mathcal{O}$ is the movable object involved in the activity;*
  - *$q^S_a, q^G_a \in \mathcal{Q}_{o_a}$ are the configurations it must assume at the start and the end of the activity, respectively.*

For each activity a where $mc(a) \neq \bot$, we set $\gamma^a_{o_a} = 1$, i.e., each activity moving object o uses the resource o.

This definition adds motion constraints to an OS problem. Since a trajectory $\tau(a) : \mathbb{R}_{\ge 0} \to \mathcal{Q}$ specifies the configuration of object o_a at each time $t \in [s(a), e(a)]$, describing its continuous motion from q^S_a to q^G_a, we now extend solution schedules to handle SAMP problems.

Definition 7. *A **SAMP schedule** for $\psi = \langle \phi, \mathcal{O}, \mathcal{W}, \mathcal{Q}, u, i, mc\rangle$ is a tuple $\pi = \langle p, s, e, \tau\rangle$, with $\langle p, s, e\rangle = \rho$ a schedule for the OS problem ϕ, and $\tau : \mathcal{A} \to \mathbb{R}_{\ge 0} \to \mathcal{Q} \cup \{\bot\}$ a function that assigns to each $a \in \mathcal{A}$ a trajectory for the movable object o_a, if $mc(a) \neq \bot$.*

Note that we use (as customary) integer time for scheduling and real time for motion trajectories, following common practice in each domain; uniforming the time domain would require only minor adjustments in the formalization.

Let o_a be the object moved by activity $a \in \mathcal{A}$, i.e., whose motion constraint is $mc(a) = \langle o_a, q^S_a, q^G_a\rangle$. A SAMP schedule is **non-conflicting** if ρ is non-conflicting and there exists no time $t \in \mathbb{R}$ and activities $a_1 \neq a_2 \in \pi$ such that $o_{a_1} = o_{a_2}$ and $s(a_1) \le t \le e(a_1) \wedge s(a_2) \le t \le e(a_2)$, i.e., two activities moving the same object do not overlap in time. Note that any valid OS schedule satisfies this condition, as movable objects are modeled as unary resources.

We now define a function that maps object configurations over time under a non-conflicting SAMP schedule.

Definition 8. *Let π be a non-conflicting SAMP schedule, and let the sequence of motions moving o be $\mathcal{A}^o_\pi = \langle a \in \mathcal{A} \mid p(a) \wedge o = o_a\rangle = \langle a_0, a_1, \ldots, a_n\rangle$, ordered so that for any $a_i, a_j \in \mathcal{A}^o_\pi$, if a_i precedes a_j in π, then $s(a_i) < s(a_j)$. The **evaluation function** $\zeta_\pi : \mathcal{O} \times \mathbb{R}_{\ge 0} \to \mathcal{Q}$, which returns the configuration of $o \in \mathcal{O}$ at time $t \in \mathbb{R}_{\ge 0}$, is defined as:*

$$\zeta_\pi(o,t) = \begin{cases} i(o) & \text{if } t < s(a_0), \\ \tau(a_i)(t) & \text{if } s(a_i) \le t \le e(a_i), i \in \{0, \ldots, n\}, \\ \tau(a_i)(e(a_i)) & \text{if } e(a_i) < t < s(a_{i+1}), i \in \{0, \ldots, n\text{-}1\}, \\ \tau(a_n)(e(a_n)) & \text{if } t > e(a_n). \end{cases}$$

The validity of π is then defined as follows.

Definition 9. *A non-conflicting SAMP schedule π is **valid** for the SAMP problem ψ if ρ is valid for the OS problem ϕ as per Definition 4 and the following constraints hold:*

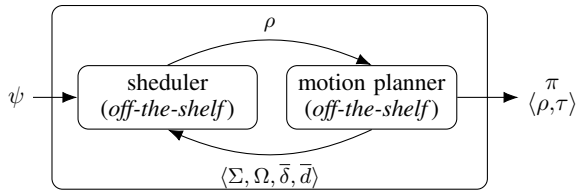

Figure 2: Our framework. Given a SAMP problem ψ, the scheduler sends a candidate schedule ρ to the motion planner. If invalid, the planner returns *geometric* (unreachable configurations Σ and obstacles Ω) and *temporal* (new delays $\overline{d}$ and durations $\overline{\delta}$) refinements until a valid SAMP schedule π is found (with trajectories τ), if one exists.

1. $\forall t \in \mathbb{R}_{\geq 0}$, $\forall o_i, o_j \in \mathcal{O}$ *such that* $o_i \neq o_j$, $occ(o_i, \zeta_\pi(o_i, t)) \cap occ(o_j, \zeta_\pi(o_j, t)) = \emptyset$, *i.e., object motions are collision-free.*
2. $\forall o \in \mathcal{O}$, $\forall t \in \mathbb{R}_{\geq 0}$, *the configuration* $\zeta_\pi(o, t)$ *lies on a trajectory that is dynamically feasible under the control model* u_o *of* o*; i.e., that can be executed by its controller.*
3. $\forall o \in \mathcal{O}$, *let* $\mathcal{A}^o_\pi = \langle a_0, \ldots, a_n \rangle$ *be the sequence of activities in* π *moving* o*. Then,* $\tau(a_i)(e(a_i)) = \tau(a_{i+1})(s(a_{i+1}))$ *for all* $i \in 0, \ldots, n-1$*, ensuring that the trajectory of* o *is continuous in space-time among all activities moving it.*

One interesting case is the one with no fluents nor effects. This is practically relevant because not all schedulers support fluents (e.g., OR-Tools (Perron and Didier 2025)).

Definition 10. *An **OS problem without fluents** is defined as OS* ϕ *with* $\mathcal{V} = \emptyset$*. Accordingly, a **SAMP problem without fluents** is defined as SAMP* ψ *with* $\mathcal{V} = \emptyset$*.*

In this paper, we propose a framework for SAMP that supports different schedulers by expressing constraints either using fluent conditions and effects or just using precedence constraints. The framework is detailed in the next section.

The Core Framework

Our SAMP framework incrementally learns symbolic abstractions of motion tasks (Algorithm 1). It interleaves an off-the-shelf scheduler, which proposes a motion-agnostic candidate schedule ρ (line 4), with an (instrumented) off-the-shelf motion planner that checks the feasibility of ρ via GETMOTIONORREFINE (line 13). The motion planner returns valid trajectories, used to decorate the motion activities in ρ, if they exist; otherwise, it provides spatio-temporal refinements for the next scheduling iteration (Figure 2).

The initial problem submitted to the scheduler is the OS ϕ from the SAMP problem enriched with simple constraints to ensure object trajectories are continuous (third condition of Definition 9). This can be achieved either by a fluent tracking each object's configuration and imposing a condition for each motion activity, or via precedence constraints restricting the admissible ordering of motion activities. For the motion planning problem, solving it monolithically would be computationally infeasible; thus, we divide the schedule into

Algorithm 1: The Core Framework

```
1  begin SOLVE(ψ, opt, t_p, timeout)
2    ψ' ← ψ    it ← 0
3    while Now() < timeout do
4      ρ, status ← get-schedule(ψ', opt)        ▷ Invoke the scheduler
5      if status ∈ [VALID, OPTIMAL] then
6        τ(ρ) ← ∅
7        conf(o) ← i(o)   ∀o ∈ O
8        foreach G ∈ P(ρ) do ▷ Check each parallel motion group
9          foreach a ∈ G do      ▷ Geometric check of each activity
10           if ¬ GETMOTIONORREFINE({a}, ψ', conf, GEOM, t_p)
               then goto 20
11         foreach a ∈ G do      ▷ Temporal check of each activity
12           if ¬ GETMOTIONORREFINE({a}, ψ', conf, TIME, t_p)
               then goto 20
13         τ(G) ← GETMOTIONORREFINE(G, ψ', conf, ALL, t_p)
14         if τ(G) ≠ ∅ then τ(ρ) ← τ(ρ) ∪ τ(G) else goto 20
15         update(conf, G)
16       if τ(ρ) ≠ ∅ then return ⟨ρ, τ⟩ ▷ Return SAMP schedule
17     else
18       if it == 0 then return UNSOLVABLE
19       t_p ← 2 · t_p    ψ' ← ψ        ▷ Double timeout and reset ψ
20     it ← it + 1
21   return INCOMPLETE
22 begin GETMOTIONORREFINE(G, ψ, conf, refs, t_p)
23   τ(G) ← ∅    path-found ← True
24   s_min ← min({s(a)|a ∈ G})
25   C_G ← {(mc(a), δ_a = s(a) − s_min)|a ∈ G}
26   if refs = GEOM ∧ G = {a} then
27     path-found, Σ, Ω ← get-path(C_G, conf, t_p)
28   else
29     τ(Ḡ), d̄, δ̄, Σ, Ω ← get-motion(C_G, conf, t_p)        ▷ Ḡ ⊆ G
30   if τ(Ḡ) = ∅ ∨ ¬path-found then
31     ψ.add-geometric-refinements(Ḡ, Σ, Ω, conf)
32   else if refs = TIME or refs = ALL then
33     if ¬ ⋀_{a∈Ḡ} d̄_a + δ̄_a ≤ d_a + δ_a then
34       C̄_Ḡ ← {(mc(a), δ̄_a)|a ∈ Ḡ}
35       ψ.add-temporal-refinements(Ḡ, C̄_G, d̄, conf)
36       return ∅
37   return τ(G)
```

parallel motion groups: subsets of activities that can interfere with each other but are independent from other groups.

Definition 11. *Two activities* $a, b \in \mathcal{A}$ *are **parallel** in* ρ *if* $p(a)$ *and* $p(b)$ *hold, and* $\exists t \in \mathbb{R}$ *such that* $s(a) \leq t \leq e(a) \wedge s(b) \leq t \leq e(b)$*. They are further defined as **motion-parallel** if they are parallel,* $mc(a) \neq \perp$*, and* $mc(b) \neq \perp$*.*

A ***parallel motion group*** $\mathcal{G}(\rho)$ ($\mathcal{G}$ for the rest of paper) is a maximal set of motion-parallel activities in ρ, with $\mathcal{P}(\rho)$ the set of all such groups (see Figure 3). Given $\mathcal{G} \in \mathcal{P}(\rho)$, and the configuration of each object at the start time of $\mathcal{G}$ (*conf*, line 7), GETMOTIONORREFINE either:

- **Gets** $\tau(\mathcal{G})$: finds collision-free, temporally consistent trajectories for all the activities in $\mathcal{G}$.
- **Adds geometric refinements**: exploits the motion planner's exploration to identify unreachable locations and blocking obstacles for those activities that are spatially infeasible and adds them as new constraints of ψ.

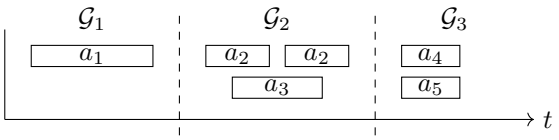

Figure 3: Parallel motion groups $\mathcal{P}(\pi) = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$.

• **Adds temporal refinements**: adds constraints on execution durations and inter-activity delays for motions that are geometrically feasible but violate the temporal constraints imposed by the scheduler, ensuring safe synchronization.

If $\tau(\mathcal{G})$ exists, the planner updates each object's configuration to the goal of the last activity moving it (line 15) and goes to the next group. If all groups are valid, a SAMP plan $\pi = \langle \rho, \tau \rangle$ is returned (line 16), optimal under Definition 5 (assuming optimality depends solely on ρ, not on trajectory optimization); otherwise, new constraints are generated.

Before describing how constraints are computed, we give a few final details of the core framework. Since many motion planners are sample-based, they may fail to terminate if no path exists or even time out despite a solution existing (*false-negative* sensitivity). This means a schedule marked as unsolvable may be truly infeasible (line 18) or *falsely* unsolvable due to insufficient sampling. To address this, we impose a timeout t_p per planner call and, when a solution is initially unsolvable, double t_p up to an upper limit, reset the refinements (line 19), and restart, mitigating false negatives with minimal added complexity. To further improve efficiency, we cache the trajectories of activities or groups whose motion constraints have already been validated. Before re-evaluating any constraints, the cache is checked to avoid redundant computations. This cache persists across restarts, enabling the reuse of previously validated trajectories and reducing computational overhead. The final optimization (gray box of Algorithm 1) reduces the cost of evaluating entire groups via a layering architecture that, before checking entire groups (Layer 2), validates their single activities (Layer 1). Each activity undergoes a geometric feasibility check (line 9), followed by a temporal feasibility check (line 11). Both operate as specialized instances of GETMOTIONORREFINE (see Algorithm 1), but applied to single activities and their respective refinements. In this context, the geometric check effectively verifies the existence of a path, simplifying the motion planner's role to that of a path finder (line 27). This pre-check boosts performance by avoiding unnecessary and expensive group-level synchronization checks, as shown in our experimental evaluation.

Geometric and Temporal Refinements

In this section, we detail how GETMOTIONORREFINE computes and formulates the spatio-temporal refinements.

Geometric Refinements. Let $\mathcal{Q}_\psi \subseteq \mathcal{Q}$ be the finite subset of configurations relevant to the problem, i.e., those actually involved in the motion activities of the problem, defined as:

$$\mathcal{Q}_\psi = \{q_a^S, q_a^G | a \in \mathcal{A}, mc(a) = \langle o_a, q_a^S, q_a^G \rangle\},$$

If the motion planner fails to find a trajectory for the group $\mathcal{G}$ (line 29), or a path in the case of a singleton group (line 27), it reports the spatial conflicts encountered, being:

• $\Sigma = \{\sigma_a \subseteq \mathcal{Q}_\psi \mid a \in \mathcal{G}\}$: the set of **unreachable configurations** for each activity $a \in \mathcal{G}$. For an activity a moving o_a from q_a^S to q_a^G, the reachable set is:

$$\widetilde{\sigma}_a = \left\{ q \in \mathcal{Q}_\psi \mid occ(o_a, q) \subseteq \mathit{reach}(q_a^S) \right\},$$

where $\mathit{reach}(q_a^S)$ is the region reachable from q_a^S, computed from the area explored by the motion planner. $\sigma_a = \mathcal{Q}_\psi \setminus \widetilde{\sigma}_a$ includes all unreachable configurations, i.e., all configurations outside $\mathit{reach}(q_a^S)$, including q_a^G if not reachable.

• $\Omega = \{\omega_a \subset \mathcal{O} \mid a \in \mathcal{G}\}$: the set of **blocking obstacles** identified by collision checking for each $a \in \mathcal{G}$. Given a, when using a sampling-based motion planner, an object $o \in \mathcal{O}$ is added in ω_a if $o \neq o_a$ and a collision was detected between o and o_a when extending the motion-tree of o_a, i.e., o blocks the expansion of possible motions of o_a.

Such spatial conflicts are used to inform the scheduler that the configuration of at least one blocking obstacle must be modified to make an otherwise unreachable configuration reachable. This refinement is performed by ψ.*add-geometric-refinements*$(\mathcal{G}, \Sigma, \Omega,$ *conf*$)$ (line 31). To formalize this constraint, we first define the set of **rivals** of a parallel motion group $\mathcal{G}$ as the motion activities not in $\mathcal{G}$:

$$\widetilde{\mathcal{G}} = \{r \in \mathcal{A} \mid r \notin \mathcal{G}, mc(r) \neq \bot\}.$$

An activity $r \in \widetilde{\mathcal{G}}$ does not overlap with $\mathcal{G}$ ($\neg$*overlaps*$(r, \mathcal{G})$) if it does not exist or is entirely scheduled before or after $\mathcal{G}$:

$$r.\text{present} \rightarrow [\bigwedge_{a \in \mathcal{G}} (r.\text{end} < a.\text{start}) \vee \bigwedge_{a \in \mathcal{G}} (r.\text{start} > a.\text{end})]$$

Given $\mathcal{G}$, let $\mathcal{G}_o = \{a \in \mathcal{G} \mid o_a = o\}$ be the activities in $\mathcal{G}$ moving o, and let a^o_{min} be the first activity moving o (i.e., $s(a^o_{min}) \leq s(a)\ \forall a \in \mathcal{G}_o$). We define the refinement condition formula $\text{RCOND}(\mathcal{G})$ as:

$$\bigwedge_{a \in \mathcal{G}} a.\text{present} \wedge \bigwedge_{\substack{o \in \mathcal{O} \\ a \in \mathcal{G}_o \setminus \{a^o_{min}\}}} a^o_{min}.\text{end} \leq a.\text{start} \wedge \bigwedge_{r \in \widetilde{\mathcal{G}}} \neg \mathit{overlaps}(r, \mathcal{G})$$

It specifies the condition under which all activities in $\mathcal{G}$ are scheduled, with each activity moving an object, and all rival activities not overlapping $\mathcal{G}$ (temporal and geometric refinements thus depend only on the activities of the group and the initial configuration of each movable object). The new constraint sent to the scheduler is then defined as follows:

$$\text{RCOND}(\mathcal{G}) \rightarrow \bigvee_{b \in \mathcal{G}, o \in \omega_b} \text{CHCONF}(b, \mathit{conf}(o))$$

with $o \in \omega_b$ being a blocking obstacle for $b \in \mathcal{G}$ and *config(o)* its current configuration. That is, if $\mathcal{G}$ is scheduled and none of its rival activities overlap with it, then the configuration of at least one blocking obstacle must change. This constraint can be encoded either using fluents or without them.

In the case of fluents, for each activity $b \in \mathcal{G}$ we introduce into ψ a corresponding auxiliary activity b'. The activity b' has the same start and end times as b, and it includes

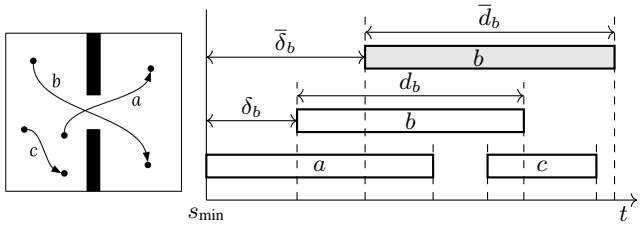

Figure 4: Temporal refinement for $\mathcal{G} = \{a, b, c\}$, starting at $s_{\min}$ (start of a). The motion planner delays the start of b (from δ_b to $\overline{\delta}_b$) and increases its duration (from d_b to $\overline{d}_b$).

a precondition on the fluent f_o, which represents the configuration of the object $o \in \omega_b$. This precondition requires that the value of f_o be *different* from the blocking configuration $conf(o)$. Intuitively, the role of b' is to ensure that, whenever b is relevant for execution, the object o is not in the configuration that would block or invalidate the execution of b. Then, our refinement requires b' to be present:

$$\textsc{ChConf}(b, \mathit{conf}(o)) = b'.\text{present} \wedge (b'.\text{start} = b.\text{start}) \wedge (b'.\text{end} = b.\text{end})$$

Without fluents, *helper activities* $\mathcal{H} = \{h \in \mathcal{A} \mid mc(h) = \langle o_h, q_h^S, q_h^G \rangle \wedge q_h^G \neq \mathit{conf}(o)\}$ move o to any state different from *conf(o)* and *deleter activities* $\mathcal{X} = \{x \in \mathcal{A} \mid mc(x) = \langle o, q_x^S, q_x^G \rangle \wedge q_x^G = \mathit{conf}(o)\}$ place o in the blocking configuration *conf(o)*. We define $\textsc{ChConf}(a, \mathit{conf}(o))$ as:

$$\textsc{Del}(b, \mathit{conf}(o)) \vee \bigvee_{h \in \mathcal{H}} \left((h.\text{present} \wedge (h.\text{end} < b.\text{start})) \wedge \bigwedge_{x \in \mathcal{X}} (x.\text{present} \rightarrow (x.\text{end} < h.\text{start}) \vee (x.\text{start} > b.\text{end})) \right)$$

with $\textsc{Del}(b, \mathit{conf}(o))$ being

$$\begin{cases} \bigwedge_{x \in \mathcal{X}} x.\text{present} \rightarrow (x.\text{start} > b.\text{end}) & \text{if } \mathit{conf}(o) \neq i(o) \\ False & \text{otherwise} \end{cases}$$

Intuitively, this constraint requires that either there is no deleter activity before b and the initial configuration of o is different from *conf(o)*, or that there exists an helper activity occurring before b and any deleter activities happen before the helper or after b. In essence, obstacles must be removed before executing a motion they would otherwise block.

To improve performance and avoid repeated computation, we propagate and cache reachability information for all equivalent objects, i.e., sharing the same geometry and control, located within the same reachability area. For singleton groups $\mathcal{G} = \{a\}$, we generalize geometric constraints to all activities moving equivalent objects from the same reachability area $\widetilde{\sigma}_a$ toward a target configuration within the set of configurations σ_a deemed unreachable by a.

Temporal Refinements. Geometric feasibility does not guarantee temporal feasibility: scheduled times may differ from those needed for execution, and parallel motions may require delay adjustments for collision-free synchronization.

GetMotionOrRefine performs this check. It computes the earliest start time $s_{min} = \min\{s(a) \mid a \in \mathcal{G}\}$ among the activities of the group (line 24), and it collects $\mathcal{C}_\mathcal{G}$ (line 25), i.e., the set of motion constraints with scheduled start times $\delta_a = s(a) - s_{min}$. If for at least one subset $\overline{\mathcal{G}} \subseteq \mathcal{G}$ (possibly $\mathcal{G}$ itself) the motion planner identifies trajectories $\tau(\overline{\mathcal{G}})$ that are geometrically feasible but fail to satisfy the timing constraints imposed by the scheduler (line 29), then *get-motion* immediately returns (as this is sufficient to prove the whole candidate schedule is unfeasible) and outputs:

- $\overline{d} = \{\overline{d}_a \in \mathbb{R}_{\geq 0} \mid a \in \overline{\mathcal{G}}\}$: new estimated durations.
- $\overline{\delta} = \{\overline{\delta}_a \in \mathbb{R}_{\geq 0} \mid a \in \overline{\mathcal{G}}\}$: new estimated delays from s_{min}

These values are computed from the space-time trajectories generated by the motion planner. Specifically, given an activity a moving o_a, and given the space-time sequence of states along its planned trajectory, we compute the actual motion start time $\overline{\delta}_a$ as the earliest timestamp at which o_a exhibits non-negligible translational or angular displacement with respect to s_{min}. We then determine the motion duration $\overline{d}_a$ by summing the time intervals Δt from the first movement of o_a until it comes to rest (see Figure 4). In this case, GetMotionOrRefine verifies whether the needed timing $\overline{d}_a + \overline{\delta}_a$ does not exceed the scheduled $d_a + \delta_a$ for $a \in \overline{\mathcal{G}}$ (line 33). If this condition holds, the space-time trajectory is deemed valid (cached) and returned: we assume it is always possible for movable objects to pause. If the computed timings exceed the scheduled ones, ψ.*add-temporal-refinments*($\overline{\mathcal{G}}$, $\overline{\mathcal{C}}_g$, $\overline{d}$, *conf*) uses needed durations ($\overline{d}$), delays ($\overline{\delta}$, included in $\overline{\mathcal{C}}_g$), and current configurations (*conf*) to add to the problem a new temporal refinement indicating that $\overline{\mathcal{G}}$ cannot be executed as scheduled unless at least one duration or delay is adjusted (line 35). Formally:

$$\textsc{RCond}(\mathcal{G}) \rightarrow \textsc{ChTime}(\overline{\mathcal{G}})$$

This means that if the parallel group $\mathcal{G}$ is scheduled and no rival $r \in \widetilde{\mathcal{G}}$ overlaps with any $a \in \mathcal{G}$, the timing of activities in $\overline{\mathcal{G}}$ must be adjusted. Given $\overline{\delta}_a$, $\overline{d}_a$, and ω_a, $\textsc{ChTime}(\overline{\mathcal{G}})$ is

$$\bigvee_{\substack{a \in \mathcal{G} \\ o \in \mathcal{O}}} \textsc{ChConf}(a, \mathit{conf}(o)) \vee \bigvee_{a \in \overline{\mathcal{G}}} a.\text{start} - \min_{b \in \overline{\mathcal{G}}}(b.\text{start}) < \delta_a \vee \bigvee_{a \in \overline{\mathcal{G}} \mid (d_a + \delta_a) < (\overline{\delta}_a + \overline{d}_a)} (a.\text{end} - \min_{b \in \overline{\mathcal{G}}}(b.\text{start})) \geq \overline{\delta}_a + \overline{d}_a$$

Thus, the scheduler must either require an object's configuration to change before at least one group activity starts, advance the start of at least one activity in the subgroup, or extend the duration of some activity a in the subgroup to at least the value $\overline{\delta}_a + \overline{d}_a$ estimated by the motion planner.

As an example, consider the parallel motion group $\mathcal{G} = \{a, b, c\}$ of Figure 4, which starts at $s_{\min} = s(a)$ (the start time of a). To ensure b is feasible when executed in parallel with a ($\overline{\mathcal{G}} = \{a, b\}$), the motion planner schedules b to start at $\overline{\delta}_b$ with an updated duration $\overline{d}_b$ (no obstacle obstructs o_b, the object moved by b). In this case, the motion planner must inform the scheduler that either (i) b must be anticipated with respect to the current schedule (an option the scheduler has

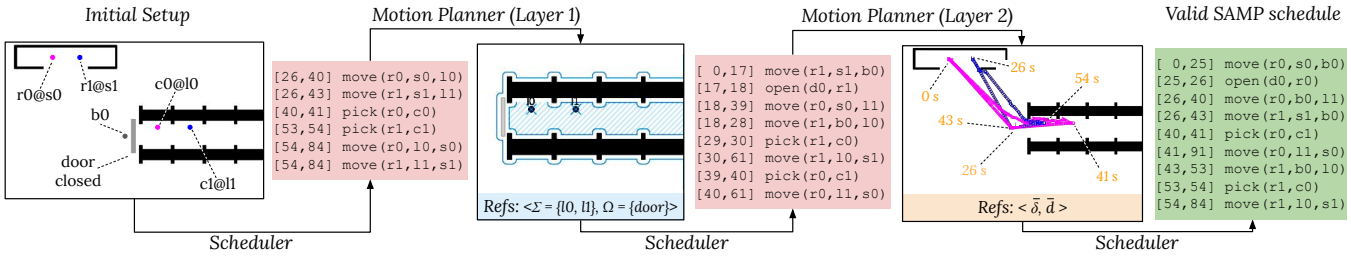

Figure 5: A logistics scenario with two robots (r_0, r_1) delivering two items (c_0, c_1) from l_0 and l_1. The first schedule is infeasible as $\Sigma = \{l_0, l_1\}$ is blocked by $\Omega = \{\text{door}\}$ (Layer 1, RRT). The second schedule is geometrically feasible but trajectories need updated delays $\overline{\delta}$ and durations $\overline{d}$ (Layer 2, ST-RRT*). Such motion planning's feedback leads to a final valid SAMP schedule.

not yet requested the motion planner to evaluate), or (ii) b must be assigned a new end time equal to $\overline{\delta}_b + \overline{d}_b$. Formally:

$$\textsc{ChTime}(\overline{\mathcal{G}}) = b.\text{start} - s(a) < \delta_b \vee b.\text{end} - s(a) \geq \overline{\delta}_b + \overline{d}_b.$$

For singleton groups, temporal refinements apply to all equivalent activities, as with geometric refinements.

As a result, our framework synchronizes parallel motion activities by postponing starts, adjusting trajectories and durations, or executing stop-and-go maneuvers on the objects.

Formal guarantees. Our framework is *sound*: it returns a solution only if the schedule satisfies all constraints and the motion planner finds valid trajectories for all its motion activities. It is *relatively optimal* for makespan optimization: if the motion planner computes duration-minimal solutions and the scheduler generates makespan-minimal plans, the resulting SAMP solution is makespan optimal. Assuming the motion planner is complete, *relative completeness* (returning a solution if one exists) relies on showing that the learned constraints always prune the candidate schedule from the solution space of the scheduler and do not cut any valid solution. The first property follows from the presence of spatio-temporal refinements, while the second relies on the refinements being triggered by RCOND .

Experimental Evaluation

We now evaluate our framework's ability to generate valid plans in complex multi-object scenarios using state-of-the-art solvers. We extend the logistics benchmark and the Job Shop Problem with transportation (JSP) (Nouri, Driss, and Ghédira 2016) to include navigation tasks, stressing both the scheduler and motion planner with space-time refinements:

- **Logistics**: n_r robots, starting at a depot, must transport items from n_s shelves back to the depot (as in Figure 1). Shelves are arranged into narrow corridors, sometimes blocked by obstacles (closed doors) that must be moved to allow access. Each shelf contains n_i items, accessible from either the corridor (inner) side or the outer side.
- **JSP**: n_r robots must move n_i items between n_m machines for treatment. Then, each item is placed on a pallet for collection. The machines are initially blocked by closed doors.

Available activities include robot *navigation*, door *opening/closing*, and item *loading/unloading*, all with certain durations. Only navigation requires motion planning with obstacle avoidance, door activities are instantaneous changes of door configurations, and others are symbolic. The optimization metric aims to minimize the makespan.

Tests. As the first SAMP study, we adopt a 2D setup to establish the foundations. Despite its apparent simplicity, the problem remain challenging: multi-robot coordination requires time-parametrized, dynamically feasible trajectories in continuous space (with car-like dynamics), and the motion planner's search space grows exponentially with the number of agents. We consider the following test cases:

- **Logistics**. We analyze $n_r \in \{1, 2, 3\}$ robots and $n_s = 2$ shelves forming a narrow corridor with an entrance door, each shelf holds $n_i = 4$ items [tot. instances: 24]. The door is open (DO in Table 1) or closed (DC), and items are picked only from the corridor (OC) or from both sides (ALL).
- **JSP**. We consider $n_r \in \{1, 2, 3\}$ robots, $n_i \in \{1, 2, 3\}$ items to be treated, and $n_m \in \{1, 2, 4, 6\}$ machines for treatment (i.e., n_m doors to open) [tot. instances: 36].

Our framework is domain-independent and built upon the Unified Planning library (Micheli et al. 2025), enabling seamless substitution or extension of the scheduling methods. On the motion planning side, it integrates the Open Motion Planning Library (OMPL) (Şucan, Moll, and Kavraki 2012), supporting all its planners. In our experiments, we use Aries (Bit-Monnot 2023) and its optimal variant Aries-opt (both with and without fluents), and our OR-Tools-based Constraint Programming Scheduling Engine (CPSE, without fluents). They are combined with RRT (LaValle 1998) for path planning (Layer 1) and ST-RRT* (Grothe et al. 2022) for space-time multi-robot motion planning ($t_p = 10$ s, Layer 2), following the layering approach of the gray box of Algorithm 1. We also instrument the collision checker to record obstacles encountered during the search.

Layer 2 checks motion feasibility sequentially: for parallel robots, spatio-temporal trajectories are planned one at a time, each respecting previously planned trajectories to avoid collisions. This instantiation trades completeness for scalability, and remains aligned with the ST-RRT* setup (Grothe et al. 2022; Kerimov, Onegin, and Yakovlev 2025), where scalability has been proved for up to 11 concurrent robots. This highlights the difficulty of our setting, which combines an already challenging multi-robot motion-planning problem with a scheduling problem that must sequence many (potentially optional) activities.

Tests were run on an AMD EPYC 7413 with a 1800 s

Benchmark	**CPSE** (no fluents)			**Aries** (no fluents)			**Aries** (with fluents)			**CPSE-opt** (no fluents)			**Aries-opt** (no fluents)			**Aries-opt** (with fluents)		
	#*sol*	t [s] (% t_p)	*refs*	#*sol*	t [s] (% t_p)	*refs*	#*sol*	t [s] (% t_p)	*refs*	#*sol*	t [s] (% t_p)	*refs*	#*sol*	t [s] (% t_p)	*refs*	#*sol*	t [s] (% t_p)	*refs*
LOG. OC-DO	16.7	323 (84%)	0.0, 6.4, 1.6	9.0	60 (81%)	0.0, 4.2, 0.0	18.3	286 (88%)	0.0, 6.1, 2.5	13.3	227 (77%)	0.0, 5.0, 0.9	10.3	126 (72%)	0.0, 4.2, 0.2	12.7	166 (78%)	0.0, 4.9, 0.8
LOG. OC-DC	14.7	359 (80%)	1.0, 9.1, 1.5	11.0	161 (85%)	1.0, 7.0, 0.2	19.3	377 (87%)	1.0, 10.9, 2.1	12.3	416 (75%)	1.0, 8.6, 1.9	8.0	251 (75%)	1.0, 6.5, 0.0	10.0	203 (78%)	1.0, 6.9, 0.3
LOG. ALL-DO	14.0	447 (72%)	0.0, 10.5, 2.1	9.3	96 (79%)	0.0, 6.4, 0.1	17.0	270 (87%)	0.0, 10.5, 2.6	9.7	349 (76%)	0.0, 8.7, 3.7	7.3	198 (81%)	0.0, 7.3, 3.1	8.3	195 (79%)	0.0, 6.8, 1.8
LOG. ALL-DC	11.7	401 (68%)	1.0, 12.3, 1.1	10.3	166 (81%)	0.9, 7.2, 0.4	16.0	392 (81%)	1.0, 11.9, 2.5	7.3	336 (76%)	1.0, 12.4, 3.1	2.3	165 (**92%**)	1.0, 3.6, 2.7	6.0	224 (76%)	1.0, 10.3, 0.6
JSP	13.0	355 (76%)	1.5, 6.7, 0.3	12.0	152 (89%)	1.5, 3.6, 0.1	17.0	389 (91%)	1.9, 9.5, 0.5	11.7	342 (65%)	1.4, 3.9, 0.1	12.3	210 (77%)	1.5, 3.7, 0.0	17.7	291 (79%)	2.0, 6.5, 0.1
TOTAL	70.0	378 (76%)	0.7, 8.8, 1.4	51.7	132 (83%)	0.8, 5.7, 0.2	**87.7**	343 (87%)	0.8, 9.7, 2.1	54.3	332 (74%)	0.7, 7.2, 1.7	40.3	194 (77%)	0.7, 5.1, 0.8	54.7	227 (78%)	0.9, 6.7, 0.6

Table 1: Overall performance: each cell shows the number of problems solved (# *sol*), average planning time in seconds (t), percentage of time spent in motion planning ($\%t_p$), and average refinement counts (*refs*) by type and layer: geometric (single activity), temporal (single activity), and group (combined geometric and temporal). All averaged over three runs per instance.

timeout and a 20 GB memory limit.

Results. Table 1 shows the performance averaged over three runs per instance, accounting for variability of sampling-based motion planners. In both domains, all solvers solve at least one instance with 3 robots, confirming correct handling of temporal constraints and inter-object synchronization. Scheduling requires many refinement loops (avg. 9.1) due to optional navigation activities, while motion planning remains costly, taking up to 92% of total planning time.

The framework effectively handles geometric complexity: in logistics scenarios, performance is similar whether doors are open or closed, showing robustness to spatial constraints. On the temporal side, solving instances with up to 3 robots indicates that the framework can manage synchronization. To further evaluate this ability and the benefits of parallelizing multi-robot activities, we compared the makespan of solutions produced by our approach with fully sequential schedules (i.e., no parallelization). For the instances considered, the average theoretical maximum improvement in makespan due to parallelization is 50% (e.g., 2 robots picking 2 items from shelves in parallel versus sequentially). In all cases where parallelization can improve the makespan, our approach achieves an average reduction of 41%.

Handling synchronization justifies the use of ST-RRT*, a typically expensive motion planner that, however, accounts for kinodynamic constraints and produces time-optimal trajectories. Planning times remain relatively low due to the layered architecture, which absorbs most geometric and temporal refinements at the single-action level, reducing multi-robot ST-RRT* calls (*refs* column: single-action geometric refinements, single-action temporal refinements, joint geometric–temporal refinements at the motion-parallel-group level). Layering also improves coverage: with both layers, 359 instances are solved on average; disabling Layer 1 reduces coverage to 140, and disabling only its single-robot temporal check (keeping the geometric one) yields 182. To further assess the value of refinements, we evaluated a sequential pipeline: first solve scheduling in a motion-agnostic way, then invoke motion planning once, without refinement if it fails. In our setup, such sequential pipeline cannot solve any problem. Although some instances require no geometric refinements at the single-action level, there is always at least one temporal refinement. This is not due to unrealistic duration estimates: we use a standard symmetric trapezoidal velocity profile, but it ignores multi-agent interactions, necessitating additional temporal refinements.

Focusing on performance, Aries with fluents performs best, solving 87.7 instances. In logistics (see Figure 5), at least one instance is solved with 1 robot and 8 items, 2 robots and 8 items, and 3 robots and 7 items; in JSP, instances are solved with 1–3 robots, up to 2 machines, and 3 pallets. Using fluents consistently improves performance, showing that richer state representations better guide refinements. Solvers generally handle more instances without makespan minimization, reflecting the added complexity of optimization. In Aries-Opt with fluents, the planning time spent on motion planning drops to 78%, compared to 87% in the non-optimal version, indicating more effort devoted to optimization. Without fluents, CPSE outperforms Aries (124.3 vs. 92.0 total instances solved, opt and non-opt), suggesting CPSE is more effective in this configuration. A final observation concerns refinements under different door settings. In the logistics domain, Aries-Opt without fluents generates more refinements when doors are open (ALL-DO) than when closed (ALL-DC). With open doors, fewer geometric bottlenecks and ordering constraints exist: the planner generates highly parallel schedules early in the search. Although symbolically consistent, these schedules can cause spatio-temporal conflicts at the motion-planning level (e.g., corridor congestion), requiring additional temporal refinements. When doors are closed, door-opening activities introduce explicit ordering and synchronization constraints that restrict parallelism and reduce invalid combinations.

Conclusion and Future Work

This paper defines the SAMP problem and presents a framework that solves it by interleaving off-the-shelf schedulers with (instrumented) motion planners, guided by incremental learning-based motion abstractions. The scheduler proposes candidate plans, and the motion planner checks feasibility, returning symbolic constraints to refine spatial and temporal decisions when needed. Experiments on scheduling benchmarks with navigation tasks, testing various scheduling strategies (optimal, non-optimal, with/without fluents) and planners, show the framework's effectiveness in handling multiple synchronized agents, coordinated stop-and-go behaviors, and complex spatio-temporal constraints.

In future work, we plan to extend our framework to support MAPF in addition to motion planning. Once we understand how to formulate this new problem and generate refinements, we will layer scheduling on top of MAPF to obtain a MAPF-aware scheduler, enabling the framework to tackle this problem as well and bridging continuous and discrete reasoning.

Acknowledgments

This work has been partially supported by the AI4Work project funded by the EU Horizon 2020 research and innovation program under GA n. 101135990, the STEP-RL project funded by the European Research Council under GA n. 101115870, and by the Interconnected Nord-Est Innovation Ecosystem (iNEST) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – mission 4 component 2, investment 1.5 – D.D. 1058 23/06/2022, ECS00000043).

The work of Arthur Bit-Monnot has been supported by the HumFleet project ANR-23-CE33-0003 and benefited from the AI Interdisciplinary Institute ANITI. ANITI is funded by the France 2030 program under the Grant agreement n°ANR-23-IACL-0002.

References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-Agent Pathfinding with Continuous Time. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 39–45.

Behrens, J. K.; Stepanova, K.; and Babuska, R. 2020. Simultaneous task allocation and motion scheduling for complex tasks executed by multiple robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 11443–11449.

Bit-Monnot, A. 2023. Enhancing Hybrid CP-SAT Search for Disjunctive Scheduling. In *26th European Conference on Artificial Intelligence (ECAI)*, volume 372, 255–262.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. *25th International Conference on Automated Planning and Scheduling (ICAPS)*, 25(1): 333–341.

Dantam, N. T. 2020. *Task and Motion Planning*, 1–9.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2016. Incremental Task and Motion Planning: A Constraint-Based Approach. In *Robotics: Science and Systems XII (RSS)*.

Faroni, M.; Umbrico, A.; Beschi, M.; Orlandini, A.; Cesta, A.; and Pedrocchi, N. 2024. Optimal Task and Motion Planning and Execution for Multiagent Systems in Dynamic Environments. *IEEE Transactions on Cybernetics*, 54(6): 3366–3377.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1): 265–293.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2018. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1): 104–136.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning. *30th International Conference on Automated Planning and Scheduling (ICAPS)*, 30(1): 440–448.

Grothe, F.; Hartmann, V. N.; Orthey, A.; and Toussaint, M. 2022. ST-RRT*: Asymptotically-Optimal Bidirectional Motion Planning through Space-Time. In *2022 International Conference on Robotics and Automation (ICRA)*, 3314–3320.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2017. Summary: Multi-Agent Path Finding with Kinematic Constraints. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 4869–4873.

Jiang, H.; Lin, M.; and Li, J. 2025. Speedup Techniques for Switchable Temporal Plan Graph Optimization. *39th AAAI Conference on Artificial Intelligence (AAAI)*, 39(22): 23212–23221.

Kerimov, N.; Onegin, A.; and Yakovlev, K. 2025. Safe Interval Randomized Path Planning For Manipulators. *35th International Conference on Automated Planning and Scheduling (ICAPS)*, 35(1): 213–217.

LaValle, S. M. 1998. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*.

Leet, C.; Oh, C.; Lora, M.; Koenig, S.; and Nuzzo, P. 2023. Task Assignment, Scheduling, and Motion Planning for Automated Warehouses for Million Product Workloads. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 7362–7369.

Li, J. 2024. Intelligent Planning for Large-Scale Multi-Robot Coordination. *37th AAAI Conference on Artificial Intelligence (AAAI)*, 37(13): 15445–15445.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2019. Multi-Agent Path Finding for Large Agents. *33th AAAI Conference on Artificial Intelligence (AAAI)*, 33(01): 7627–7634.

Ma, H.; Hönig, W.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2019. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. *33th AAAI Conference on Artificial Intelligence (AAAI)*, 33(01): 7651–7658.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Neville, G.; Chernova, S.; and Ravichandar, H. 2023. D-ITAGS: A Dynamic Interleaved Approach to Resilient Task Allocation, Scheduling, and Motion Planning. *IEEE Robotics and Automation Letters*, 8(2): 1037–1044.

Nouri, H. E.; Driss, O. B.; and Ghédira, K. 2016. A Classification Schema for the Job Shop Scheduling Problem with Transportation Resources: State-of-the-Art Review. In *Artificial Intelligence Perspectives in Intelligent Systems*, 1–11. Cham.

Pecora, F.; Andreasson, H.; Mansouri, M.; and Petkov, V. 2018. A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control. *28th International Conference on Automated Planning and Scheduling (ICAPS)*, 28(1): 485–493.

Perron, L.; and Didier, F. 2025. CP-SAT. https://developers.google.com/optimization/cp/cp_solver/. Version v9.12.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K.; Barták, R.; and Boyarski, E. 2021. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *International Symposium on Combinatorial Search*, 10(1): 151–158.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4): 72–82.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2): 278–285.

Tosello, E.; Valentini, A.; and Micheli, A. 2024. A meta-engine framework for interleaved task and motion planning using topological refinements. In *27th European Conference on Artificial Intelligence (ECAI)*, volume 392, 4377–4384.

Tosello, E.; Valentini, A.; and Micheli, A. 2025. Temporal Task and Motion Planning with Metric Time for Multiple Object Navigation. *39th AAAI Conference on Artificial Intelligence (AAAI)*, 39(25): 26716–26724.

Toussaint, M. 2015. Logic-geometric programming: an optimization-based approach to combined task and motion planning. In *24th International Conference on Artificial Intelligence (IJCAI)*, 1930–1936.

Zanlongo, S. A.; Dirksmeier, P.; Long, P.; Padir, T.; and Bobadilla, L. 2021. Scheduling and Path-Planning for Operator Oversight of Multiple Robots. *Robotics*, 10(2).