A Petri net based approach for multi-robot path planning
-draft-

Marius Kloetzer and Cristian Mahulea*

December 23, 2014

Abstract

This paper presents a procedure for creating a probabilistic finite-state model for mobile robots and for finding a sequence of controllers ensuring the highest probability for reaching some desired regions. The approach starts by using results for controlling affine systems in simpliceal partitions, and then it creates a finite-state representation with history-based probabilities on transitions. This representation is embedded into a Petri Net model with probabilistic costs on transitions, and a highest probability path to reach a set of target regions is found. An online supervising procedure updates the paths whenever a robot deviates from the intended trajectory. The proposed probabilistic framework may prove suitable for controlling mobile robots based on more complex specifications.

Published as:
M. Kloetzer and C. Mahulea, "A Petri net based approach for multi-robot path planning," Discrete Event Dynamic Systems: Theory and Applications, vol. 24, no 4, pp. 417-445, December 2014. DOI: 10.1007/s10626-013-0162-6

*M. Kloetzer is with Department of Automatic Control and Applied Informatics, Technical University "Gheorghe Asachi" of Iasi, Romania, *kmarius@ac.tuiasi.ro*.
C. Mahulea is with Aragón Institute of Engineering Research (I3A), University of Zaragoza, Spain, *cmahulea@unizar.es*.

1 INTRODUCTION

Planning and controlling mobile robots is a research area that continues to receive a lot of attention [2, 26]. An intensively studied problem in this area is the navigation of a mobile robot, where a control strategy should be automatically generated such that the robot reaches a target position without colliding with obstacles. Some solutions proposed for this problem directly work in a continuous state space (*e.g.* navigation and potential functions), while others abstract the initial problem to a finite-state one and find a solution by a search on a graph (*e.g.* cell decomposition methods) [2].

The latter methods were successfully tailored such that more complicated problems can be solved, where the specification is close to natural language and it is given as a formula combining temporal and logic operators [32, 9, 20]. However, most of these approaches are conservative mainly because of the abstraction to a finite-state (or discrete) problem, as described below. The main idea is to partition the environment into a set of cells (regions) having the same geometrical shape, and then to construct and to reason on a finite-state representation in form of a graph or of a transition system. In this finite representation a node or state corresponds to a region from the partition, and the edges between nodes are added based on adjacency and on the existence of continuous controllers steering the robot from one region to another, independent of the initial position of the robot in the first region. Most of the approaches based on this abstraction process are applicable to mobile robots with specific dynamics, for which such continuous controllers can be easily designed, *e.g.* affine dynamics in simpliceal or polytopal partitions, and multiaffine dynamics in rectangular partitions [13, 1]. The approaches become conservative because the controllers have to correctly steer any initiating trajectory from a region, and this steering may not be possible for the whole region, but only for a subset. Therefore, less conservative approaches can be designed for reachability analysis or for control with the goal of avoiding certain sets, rather than visiting them. Thus, some works design refining procedures for finding such subsets [21], but in such cases uncontrolled dynamics are assumed and the computational complexity increases because of the great number of obtained regions. Other works design more relaxed controllers that can steer trajectories from a region to a set of neighboring regions [12], and then use a nondeterministic abstraction (with worst case scenario on possible transitions) when searching for a solution on the finite-state abstraction.

This paper addresses the navigation problem of a team of mobile robots in a partitioned environment, with the simple tasks of reaching a set of target regions by starting from a set of given initial regions. The robots are assumed to have a negligible size, and they have affine dynamics with bounded control inputs. Obstacles can be easily avoided using the proposed approach. The problem we solve belongs to a class of multi-robot task allocation problems [11], and its usefulness follows from its possible applicability in rescue scenarios, resource gathering or resource allocation tasks.

The first main result is obtaining a probabilistic finite-state representation that allows a computationally attractive solution which is less conservative than using simpler controllers from [13] or controllers from [12] and a nondeterministic abstraction. Thus, when constructing the finite-state representation, controllers steering trajectories between adjacent regions are designed based on the results in [13, 12] and probabilities are assigned to every edge of the abstraction. Whenever there exists a control law as in [13] driving all trajectories from one region to an adjacent one, probability 1 is added to the corresponding transition. When such a controller does not exist, less restrictive controllers to set of facets are employed [12], and a history-based probability of reaching the next region is computed, based on the entering facet for the current region.

The second main result is a procedure to obtain a Petri net model of the environment that includes the history. By using the methodology proposed, the Petri net model belongs to a particular class of Petri nets called *state machines* having some good properties to overcome the computational complexity of obtaining optimal paths. The history-based probabilities computed in the first step will be used to define the objective function of the optimization programming problem. Moreover, the use of Petri nets is twofolded: (i) if more robots are considered, the number of states (places) of the model remains the same; (ii) the model can be used for task planning, plan execution and plan analysis. In particular, for plan analysis, properties such as *boundedness* and *liveness* of Petri nets correspond to checking if resources usage is stable and plans have no deadlocks, as well as on stochastic performance, concerning the plan success probability [4].

Related works and contributions of the paper:
Modeling and control of robot tasks using a Petri net approach has been considered in literature in the last years [19, 18, 4]. In [19], a Petri net model of a robot task ensuring some qualitative properties is proposed and the quantitative properties are studied through simulations. In [18, 4], Generalized Stochastic Petri net models are considered for modeling and analyzing robot tasks. A modular strategy is proposed, where different Petri net models are given for environment layer and action executor layer. However, the model of the environment considers different modes of the dynamics, and in general it is obtained by identification/simulation. In our case, we split the environment in regions and then we show how to automatically compute control laws to move from one region to another and how to compute probabilities to characterize these movements.

Thus, this work creates a probabilistic framework for abstracting the control capabilities of mobile robots into finite-state representations. Such representations can be further used for solving more complex robotic problems. For example, in [25] a probabilistic planning based on linear temporal logic specifications is envisioned, but the probabilities are fixed and they come from measurement uncertainty, rather than from controller design. In [8] the authors assume controls yielding to probabilistic movements, but the actual probabilities are priori given and not computed from specific control laws. A probabilistic model of foraging robots composing a swarm system [28] is the starting point of [24], and the goal is to analyze possible behaviors by employing model checking algorithms. Two global models for the whole team are constructed in [24], either by a product approach that quickly yields a state-space explosion problem, or by a model that counts the number of robots in each behavioral state. The tokens from the Petri net model we use are related with the counters from [24], but our abstraction models robot locations and it is aimed towards solving a control problem, while the one from [24] models robot behaviors and analyzes swarm properties. Probabilistic model checkers are used in [16] for quantifying the influence of probabilistic measures on the satisfaction of a temporal logic specification for a mobile robot. However, the probabilities from [16] come from sensor errors and the robot actuation is assumed to be perfectly deterministic, while in our scenario the probabilities come from uncertainties induced by the abstraction of movement capabilities. Feedback policies for a mobile robot are designed in [7] by considering a continuous model affected by bounded disturbances. The focus of [7] is on directly working in the infinite state space of a discrete time model and on providing formal guarantees under given disturbance bounds, rather than abstracting to finite state representations and computing disturbances (probabilities) resulting from this process. The idea of having history-based transition costs is used in works as [33, 5, 6]. In [5, 6] the goal

is to find a feasible trajectory for a car-like robot with a kinematic model. By assuming a partition in rectangular and equal tiles, history-dependent costs are computed for transitions between adjacent cells, based on the capability of the vehicle of following a sequence (history) of tiles. The framework from our method is different since we use a simpliceal partition and we compute history-based costs from control laws for affine systems. Moreover, different than most of the above mentioned related works, we focus on proposing a computationally feasible algorithm for a team of robots rather than for a single agent. Also, we outline both an explicit method for computing probabilities based on movement control laws, and a modeling methodology for the whole team of robots that avoids an explosion in the number of states.

The results presented in this paper extend the ones from [22, 29] in several ways. Unlike [22], the abstraction performed takes into account obstacles from the environment, the Petri net model is extended by allowing multiple identical agents, and an online solution for updating the paths is integrated in the method. Different than [29], we assume static obstacles, fact which yields a reduced computational complexity for obtaining the optimal solution and a fully automated framework that does not require heuristic adjustments of cost parameters. Moreover, some possible extensions for improving the probabilistic abstraction are discussed. Although the algorithms presented in this work are used for planning mobile robots, they can be also applied to different research scenarios where similar finite state abstractions are of interest.

All algorithms presented in this paper have been implemented in MATLAB and are freely downloadable from [23]. The implementation also includes the polyhedral operation package from [10], and the linear programming problem (LPP) solver from [30].

The paper is structured as follows. Sec. 2 presents some preliminaries necessary throughout the paper and formulates the problem to solve. The algorithms for computing history-based probabilities are included in Sec. 3, and the Petri Net based solution is given in Sec. 4. Sec. 5 discusses the complexity and the conservativeness of our method. An example is presented in Sec. 6, and Sec. 7 formulates some concluding remarks.

2 PRELIMINARIES

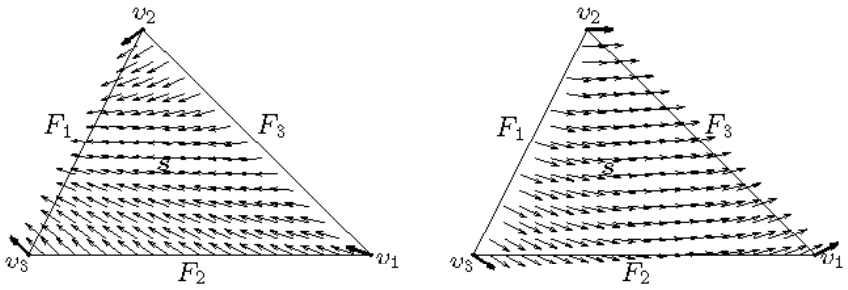
First, some results concerning control of affine systems on simplices are presented, and then some preliminary notions on discrete Petri Nets (PN) are given. The problem to solve is formulated towards the end of this section.

2.1 Affine System Control

This subsection briefly presents results from [12, 13] concerning feedback control design for an affine system evolving in a simplex. Although these results are applicable to arbitrary state-space dimensions, this paper assumes only two-dimensional systems. This restriction is motivated by the intention of providing a framework suitable to mobile robots evolving in planar environments.

Consider a two-dimensional simplex (triangle) $s \in \mathbb{R}^2$, and the following affine control system evolving in s:

$$\dot{x} = Ax + b + Bu,\ x \in s,\ u \in U \subset \mathbb{R}^m, \tag{1}$$



(a) (b)

Figure 1: Vector fields ensuring that the triangle s is left in finite time. At any point inside triangle, the vector field is a convex combination of its values at vertices (represented with thick arrows). a) Any trajectory initiating in triangle s leaves it through facet F_1. b) Triangle is left in finite time through either F_2 or F_3, and there is no additional information available on corresponding probability.

where $A \in \mathbb{R}^{2\times 2}$, $B \in \mathbb{R}^{2\times m}$, $b \in \mathbb{R}^2$, and U is a given polyhedral subset of $\mathbb{R}^m$ ($m \leq 2$) capturing control constraints.

In [13] the authors formulate necessary and sufficient conditions for constructing a feedback control law $u(x)$, $x \in s$ such that any trajectory initiating in s leaves the simplex through a desired facet in finite time. However, due to dynamics (1) and control bounds U, in some cases such control laws cannot be created. Therefore, a less restrictive solution was developed in [12], where control laws driving any trajectory initiating in s through *a set* of desired facets were designed.

The problem of designing such control laws is computationally attractive, since it reduces to solving a set of three LPPs, one for each vertex of s. Controllers are constructed only at vertices, and then the feedback control anywhere inside s is a convex combination of these values, given by (2), where $v_1, v_2, v_3 \in \mathbb{R}^2$ denote the vertices of s, and $x \in s$.

$$u(x) = \begin{bmatrix} u(v_1) & u(v_2) & u(v_3) \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2}$$

As a consequence of (2), the dynamics anywhere in s are a convex combination of the controlled dynamics at vertices. Furthermore, on a facet of s, the control is uniquely defined as a convex combination of control values from the vertices of that facet, and the same applies to the controlled dynamics. The interested reader is referred to [12, 13] for technical details on testing the existence and on constructing such controllers. For an easier understanding of the outcome of such control problems, Fig. 1 illustrates some vector fields driving all initial states from a simplex to a single exit facet or to a set of exit facets. Thus, the vector field in Fig. 1(a) guarantees that the simplex is left in finite time through facet F_1, while in Fig. 1(b) the simplex is left through either F_2 or F_3.

2.2 Petri Nets

This subsection introduces the basic notions of PN (see [31, 35] for a gentle introduction).

Definition 1 *A Petri net (PN) is a tuple* $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$ *with* P *and* T *two disjoint sets of places and transitions;* $\boldsymbol{Pre}, \boldsymbol{Post} \in \mathbb{N}^{|P| \times |T|}$ *the pre and post matrices defining the weights of the arcs from places to transitions and from transitions to places, respectively.*

For $h \in P \cup T$, the sets of its input and output nodes are denoted as ${}^\bullet h$ and $h^\bullet$, respectively. Let p_i, $i = 1, \ldots, |P|$ and $t_j, j = 1, \ldots, |T|$ denote the places and transitions. Each place can contain a non-negative integer number of tokens, and this number represents the marking of the place. The distribution of tokens in places is denoted by $\boldsymbol{m}$, while $\boldsymbol{m}(p_i)$ or simply m_i is the marking of place p_i. The initial token distribution, denoted by $\boldsymbol{m}_0 \in \mathbb{N}^{|P|}$, is called the initial marking of the net system. A PN with an initial marking is a PN system $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$. The enabling degree of a transition t_j at a marking $\boldsymbol{m}$ is given by

$$enab(t_j, \boldsymbol{m}) = \min_{p_i \in {}^\bullet t_j} \left\{ \left\lfloor \frac{\boldsymbol{m}(p_i)}{\boldsymbol{Pre}(p_i, t_j)} \right\rfloor \right\},$$

which represents the maximum amount in which t_j can fire.

A transition $t_j \in T$ is enabled at $\boldsymbol{m}$ if and only if $enab(t_j, \boldsymbol{m}) > 0$. An enabled transition t_j can fire in any integer amount α, with $0 < \alpha \leq enab(t_j, \boldsymbol{m})$, leading to a new state $\boldsymbol{m'} = \boldsymbol{m} + \alpha \cdot \boldsymbol{C}(\cdot, t_j)$, where $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$ is the *token flow matrix* and $\boldsymbol{C}(\cdot, j)$ is its j^{th} column. It will be said that $\boldsymbol{m'}$ is a *reachable marking* that has been reached from $\boldsymbol{m}$ by firing t_j.

If $\boldsymbol{m}$ is reachable from $\boldsymbol{m}_0$ through a finite sequence of transitions $\sigma = t_{i1} t_{i2} \ldots t_{ik}$, the following state (or fundamental) equation is satisfied:

$$\boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}, \tag{3}$$

where $\boldsymbol{\sigma} \in \mathbb{N}^{|T|}$ is the *firing count vector*, *i.e.*, its j^{th} element is the cumulative amount of firings of t_j in the sequence σ. Notice that (3) is only a necessary condition for the reachability of a marking. The markings solutions of (3) that are not reachable are called *spurious markings*. Checking if a marking $\boldsymbol{m}$ is reachable or not is not an easy problem due to these spurious markings.

A PN where each transition has at most one input and at most one output place is called *state machine*. Formally, a PN is state machine if $|{}^\bullet t| \leq 1$ and $|t^\bullet| \leq 1, \forall t \in T$. A PN is called *live* if from any reachable marking any transition can eventually fire (possibly after first firing other transitions). It is well known that for state machine PNs, liveness is equivalent to strongly connectedness and non emptiness of (initial) marking. Moreover, in a live state machine, there exist *no spurious* markings [36], *i.e.*, the solutions of the fundamental equation (3) give the set of reachable markings.

2.3 Objective and Approach

This paper consideres a simpliceal partition of the free space of a planar environment cluttered with obstacles, where multiple identical mobile robots with affine dynamics evolve. Each robot has a negligible size and it starts from a given simplex, and the objective is to reach a set of simplices, without a priori assigning a target simplex to each robot. The main goal is to design a control strategy for each robot such that the overall probability of reaching targeted simplices is maximized.

The remainder of this section outlines the main steps of the proposed solution, and motivates several aspects of the used strategy. Instead of trying to solve the problem by

directly working with the continuous dynamics, the control capabilities of each robot are abstracted into a finite state representation in form of a graph. Each node in this graph corresponds to a simplex from the partition, and the outgoing edges correspond to the capabilities of constructing continuous controllers driving any state from the current simplex to adjacent triangles. These controllers are designed as in Sec. 2.1. Whenever a control ensuring that a triangle is left through a single exit facet cannot be constructed, controllers driving the continuous system through a set of facets are searched. In such situations, the next reached triangle is not exactly known (*e.g.*, see Fig. 1.b) and, from the point of view of the constructed abstraction, a probability of reaching a certain triangle from the possible set of next triangles is needed. For reducing the conservativeness of the abstract model, history-based probabilities for each edge are computed, where the history is represented by the triangle from which the current triangle was entered.

The choice of using finite-state models is motivated by results where such models allowed extensions to human-like specifications for mobile robots, as temporal and logic tasks [32, 9, 20]. However, in such results only deterministic models were used (corresponding to controllers enforcing that a simplex is left through a single facet), or probabilistic models were constructed based on measurements uncertainty [25, 16], or worst-case scenarios on possible transitions were used [21]. Therefore, this part of the proposed solution can be a valuable framework for explicitly constructing probabilistic models suitable for planning mobile robots.

The probabilistic abstraction of each robot will be further embedded into a Petri Net model, where the history-based probabilities will become static probabilities associated with transitions. This model allows an easy and computationally feasible extension to multiple identical robots, in contrast with existing complex approaches relying on product of transition systems or graphs [20]. After the PN model is constructed, sequences of regions (transitions in PN) solving the problem are obtained. The overall probability of following these sequences is optimized by solving an LPP, and the feasibility of the obtained paths is guaranteed by sstructural properties of the constructed PN model. Whenever the sequences are left (due to less than one probabilities), an online algorithm updates the paths.

3 HISTORY-BASED PROBABILITIES

This section details the algorithms that allow the abstraction of the behavior of a single agent in the given environment to a finite state probabilistic representation. Subsection 3.1 introduces some necessary notations and presents a general algorithm for finding the probabilities. Then, specific cases encountered during these computations are detailed in subsections 3.2 and 3.3. Subsection 3.4 formulates some additional aspects that can be investigated in such problems.

3.1 Control laws and corresponding probabilities

The first step is to partition in triangles the free space of the given environment (the space not covered by obstacles). The details on constructing such a partition are not in the scope of this paper, and here it is just mentioned that this construction can be performed by using an available software tool for constrained triangulations, as [37, 3, 34]. Assume that $|S|$ triangles are obtained, and they are labeled with symbols from set $S = \{s_1, s_2, \ldots, s_{|S|}\}$. Two triangles are called adjacent if they share a facet, and

the adjacency relation $adj \subseteq S \times S$ can be constructed such that $(s_i, s_j) \in adj$ if and only if $s_i \neq s_j$ and triangles s_i and s_j are adjacent.

The obtained finite state representation has the form of a special graph $G = (S, adj, p, ctrl)$, where:

- S is the set of nodes;
- adj is the adjacency relation;
- $p : S \times S \times S \to [0, 1]$ is a history-based probability map;
- $ctrl : S \times S \times S \to \mathbb{R}^{m\times 3}$ is a control assignment at vertices of a simplex, as necessary in (2).

This section is mainly concerned with creating maps p and $ctrl$. The intuition behind these maps is as follows: $p(s_i, s_j, s_k)$ is the probability of steering trajectories of system (1) from triangle s_i to s_j, given that s_i was entered from simplex s_k. For a more intuitive notation, from now on notation $p_{s_i \to s_j | s_k}$ will be used instead of $p(s_i, s_j, s_k)$. One can observe that this probability does not come from a nondeterminism in system dynamics or in system evolution, but it comes from the performed abstraction, by reducing each simplex to a single state in the finite state representation G. Also, for obtaining a more accurate abstraction G, in the following, such history-based probabilities are considered that depend not only on the current and next triangle, but also on the triangle from where the current one was entered.

For any tuple (s_i, s_j, s_k) for which $p_{s_i \to s_j | s_k} > 0$, control values are required ($ctrl(s_i, s_j, s_k) = [u(v_1)\, u(v_2)\, u(v_3)]$) at vertices of s_i for constructing a feedback control law everywhere inside s_i, as in (2). This control law implies that s_i is exited in finite time through the facet shared with s_j (if $p_{s_i \to s_j | s_k} = 1$), or through one of a set of multiple facets (if $p_{s_i \to s_j | s_k} < 1$).

For avoiding supplementary indexing or notations, in the remainder of this section, a single tuple of triangles (s_i, s_j, s_k) is considered, for which the proposed method of computing $p_{s_i \to s_j | s_k}$ is described. Of course, the map p can be easily obtained by iterating the presented procedures for all tuples $(s_i, s_j, s_k) \in S \times S \times S$ for which $(s_i, s_j) \in adj$ and $(s_k, s_i) \in adj$ (otherwise, obviously $p_{s_i \to s_j | s_k} = 0$). Fig. 2 illustrates a generic triangle s_i that was entered from state s_k through facet F_1, and the goal is to leave this triangle through facet F_3 (for reaching simplex s_j). We denote by v_i the opposite vertex of facet F_i, and n_i denotes the outer normal of facet F_i, $i = 1, 2, 3$. In the following it is assumed that $s_i \notin \{s_j, s_k\}$, and the case $s_j = s_k$ is allowed.

Alg. 1 presents the generic procedure for finding an affine feedback control law and the corresponding probability for exiting s_i to s_j, given that s_i was entered from s_k. The control law is found by using the results briefly mentioned in Sec. 2.1, and when solving the involved LPPs, an optimality criterion is imposed for maximizing the resulted speed projection on n_3. Alg. 1 starts by finding the set F_{exit} containing combinations of facets through which s_i can be left without hitting any obstacle or without leaving the environment bounds (lines 1-4). F_1 and F_3 are included in this set because there exist neighbors to s_i that share one of these facets (s_k and s_j respectively). If there is no simplex sharing F_2 with s_i, then s_i should never be left through F_2, because this would lead to either hitting an obstacle, or leaving the defined environment. It is assumed that F_{exit} is scanned by loop on line 5 in the same order enumerated on lines 2 or 4. Thus, only facet F_3 is first considered as exit facet. If a control law is found, then s_i is left for sure in finite time by hitting s_j, so the algorithm is stopped by returning exit probability 1 and the computed control law. Otherwise, the set of feasible

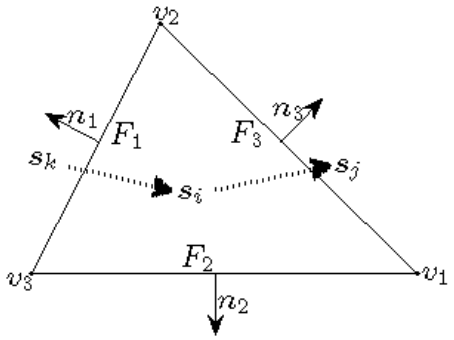


Figure 2: The desired sequence of triangles (suggested by dotted arrows) and several notations concerning s_i.

controls is relaxed by allowing a whole set of exit facets including F_3. Controllers are searched iteratively for each such set of exit facets, and whenever a controller is found, the probability of exiting s_i through F_3 is computed with procedure *find_probability*. If more probabilities were computed (line 12 reached), then the maximum of these probabilities and the corresponding control law are returned.

Algorithm 1: Control law and corresponding probability

Input: S, adj ; `/* the set of triangles and the adjacency relation */`
Output: p, $ctrl$; `/* history-based probabilities and control laws */`

1 **if** $\exists s_l \in S \setminus \{s_i, s_j, s_k\}$ *s.t.* $(s_i, s_l) \in adj$ **then**
2 $\quad F_{exit} \in \{F_3, \{F_2, F_3\}, \{F_1, F_3\}, \{F_1, F_2, F_3\}\}$
3 **else**
4 $\quad F_{exit} \in \{F_3, \{F_1, F_3\}\}$
5 **for** $F \in F_{exit}$ **do**
6 $\quad$ find control u_F for exiting simplex s_i through facet(s) from F; (by solving LPP in theorem 4.17 from [12])
7 $\quad$ **if** u_F *was found* **then**
8 $\quad\quad$ **if** $F = F_3$ **then**
9 $\quad\quad\quad$ **return** $p_{s_i \to s_j | s_k} = 1$ and u_F
10 $\quad\quad$ **else**
11 $\quad\quad\quad$ $p_{s_i \to s_j | s_k}(F) = find\_probability(u_F, s_i, s_j, s_k)$
12 $F' = \arg\max_F p_{s_i \to s_j | s_k}(F)$
13 **return** $p_{s_i \to s_j | s_k} = p_{s_i \to s_j | s_k}(F')$ and $ctrl(s_i, s_j, s_k) = u_{F'}$

3.2 Procedure *find_probability* when $s_j \neq s_k$

The procedure *find_probability* finds the probability of leaving s_i through F_3 under a feedback control law which is generically denoted in the remainder of this section by u. This probability is computed as the ratio between the length of a subset of F_1

that guarantees exiting through F_3 and the whole length of F_1. The steps followed by the *find_probability* procedure differ depending on the relation between s_j and s_k. This subsection details these steps for the case $s_j \neq s_k$, and subsection 3.3 focuses on the case $s_j = s_k$. The resulting velocity at a current state x inside s_i is denoted by $f(x) = \dot{x} = Ax + Bu(x) + b$, where the control law u found as in Alg. 1 yields $u(x)$ as in (2).

Alg. 2 starts by restricting F_1 to a subset from where the continuous trajectories do not immediately leave s_i through the entering facet F_1 (lines 1-10). The fulfillment of condition from line 1 implies zero probability for reaching s_j, because the continuous trajectories would immediately leave s_i by bouncing back from F_1 (see Fig. 3.a). If only one vertex of F_1 has a controller yielding a dynamics that prevents entering s_i through that vertex, the vertex is moved on F_1 to a point where the vector field has zero projection on n_1. Such a point is uniquely determined by solving a linear system of two equations, since the vector field on F_1 is a convex combination of its values from v_2 and v_3 (Sec. 2.1). Fig. 3.b illustrates a situation when condition from line 3 is fulfilled, and v_3 is moved along F_1. For avoiding sliding along F_1 from a moved vertex, a neighboring position where the projection on n_1 is negative is picked, such that s_i is entered.

Next, numerical integration is applied for deciding what subset of F_1 contains trajectories that leave s_i through F_3. For this, a procedure denoted $integrate_{s_i,f}$(direction,x_0) is used, procedure which integrates forward or backward in time the system with dynamics given by f, starting from initial condition x_0 in simplex s_i. The integration stops when s_i is left, and the procedure returns the exiting facet and the point on the exiting facet. This procedure correctly handles situations when the trajectory from a vertex immediately leaves s_i (as in Fig. 3.c).

First a forward integration from v_2 is employed (line 11), guided by the fact that the initial position of v_2 is the common vertex of s_i, s_j and s_k. If $F_{v_2\_exit}$ is different than the exit facet F_3, probability 0 is assigned and the *find_probability* procedure is stopped. For example, if $F_{v_2\_exit}$ is F_2, then no trajectory initiating on F_1 could reach F_3, because the trajectories of affine systems cannot intersect. Fig. 3.c and Fig. 3.d present two cases when the algorithm continues after integrating from v_2.

If $F_{v_2\_exit}$ is equal to F_3, then the integration from v_3 is performed. Fig. 3.e presents a situation when $F_{v_3\_exit} = F_3$, and the probability is computed as in line 16 (ratio between the green segment from Fig. 3.e and the length of F_1).

In case $F_{v_3\_exit} = F_2$ (from line 19 on), the backward integration from v_1 is prepared. If necessary, v_1 is moved on F_3 (similar to moving v_2 or v_3 on F_1). Note that, if condition on line 20 is true, then $f(v_{2\_init})$ has positive projection on n_3 because F_3 was included among exit facets when designing controllers. Therefore, v_{1_new} can be determined when needed, similar to finding v_{2_new} or v_{3_new}.

From v_1, backwards integration is performed with the hope of reaching F_1, case when the probability is computed as on line 26 (see Fig. 3.f and Fig. 3.g for such situations). If backward integration from v_1 yields facet $F_{v_1\_back} = F_2$, a point P_{back} on F_1 has to be found, such that the trajectory starting from P_{back} is tangent to F_2, and after that it leaves s_1 through F_3 (lines 27-32), as in Fig. 3.h. P_{back} is found (if possible) by first choosing the point $P \in F_2$ where vector field is tangent to F_2. P can be uniquely determined, because all situations handled until here guarantee that $v_1 = v_{1\_init} = v_{1\_back}$, $n_2^T f(v_{1\_init}) < 0$, and $n_2^T f(v_{3\_init}) \geq 0$ (otherwise, it would be impossible to have $F_{v_3\_exit} = F_2$ and $F_{v_1\_back} = F_2$).

Line 33 collects all unsuccessful situations when algorithm did not returned. The‘proposed algorithm can be conservative in cases when $F_{v_2\_exit} = F_1$, $F_{v_3\_exit} = F1$, or when

Algorithm 2: Procedure *find_probability* when $s_j \neq s_k$

Input: S, adj, u_F $(= ctrl(s_i, s_j, s_k))$, s_i, s_j, s_k
Output: $p_{s_i \to s_j | s_k}$

if $n_1^T f(v_2) \geq 0 \ \wedge \ n_1^T f(v_3) \geq 0$ **then**
  return $p_{s_i \to s_j | s_k} = 0$
else if $n_1^T f(v_2) < 0 \ \wedge \ n_1^T f(v_3) \geq 0$ **then**
  $v_{3\_init} = v_3$
  find $v_{3\_new} \in F_1$ s.t. $n_1^T f(v_{3\_new}) = 0$
  $v_3 = v_{3\_new}$
else if $n_1^T f(v_2) \geq 0 \ \wedge \ n_1^T f(v_3) < 0$ **then**
  $v_{2\_init} = v_2$
  find $v_{2\_new} \in F_1$ s.t. $n_1^T f(v_{2\_new}) = 0$
  $v_2 = v_{2\_new}$
$[F_{v_2\_exit}, P_{v_2\_exit}]$ = $integrate_{s_i,f}$(forward,v_2)
if $F_{v_2\_exit} \neq F_3$ **then**
  return $p_{s_i \to s_j | s_k} = 0$
$[F_{v_3\_exit}, P_{v_3\_exit}]$ = $integrate_{s_i,f}$(forward,v_3)
if $F_{v_3\_exit} = F_3$ **then**
  return $p_{s_i \to s_j | s_k} = \frac{\|v_2 - v_3\|}{\|v_{2\_init} - v_{3\_init}\|}$
if $F_{v_3\_exit} = F_1$ **then**
  return $p_{s_i \to s_j | s_k} = 0$
/* $F_{v_3\_exit} = F_2$ */
if $n_3^T f(v_1) < 0$ **then**
  $v_{1\_init} = v_1$
  find $v_{1\_new} \in F_3$ s.t. $n_3^T f(v_{1\_new}) = 0$
  $v_1 = v_{1\_new}$
$[F_{v_1\_back}, P_{v_1\_back}]$ = $integrate_{s_i,f}$(backward,v_1)
if $F_{v_1\_back} = F_1$ **then**
  return $p_{s_i \to s_j | s_k} = \frac{\|v_2 - P_{v_1\_back}\|}{\|v_{2\_init} - v_{3\_init}\|}$
if $F_{v_1\_back} = F_2$ **then**
  find $P \in F_2$ s.t. $n_2^T f(P) = 0$
  $[F_{P\_back}, P_{back}]$ = $integrate_{s_i,f}$(backward,P)
  $[F_{P\_exit}, P_{exit}]$ = $integrate_{s_i,f}$(forward,P)
  if $F_{P\_back} = F_1 \ \wedge \ F_{P\_exit} = F_3$ **then**
    return $p_{s_i \to s_j | s_k} = \frac{\|v_2 - P_{back}\|}{\|v_{2\_init} - v_{3\_init}\|}$
return $p_{s_i \to s_j | s_k} = 0$

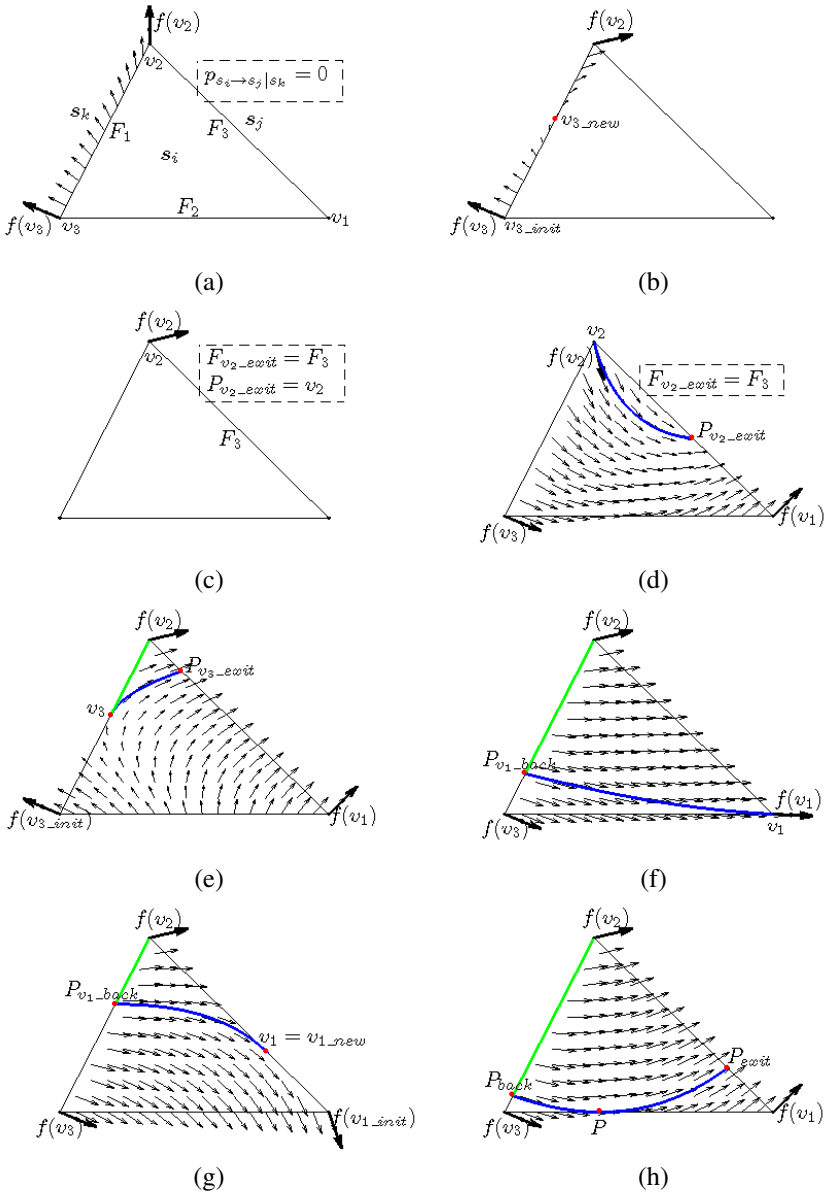


Figure 3: Possible situations encountered by Alg. 2. Vector field values at vertices of s_i are represented by thick arrows, and they define the vector field in the whole simplex s_i. The dashed boxes contain some decisions taken in Alg. 2. The trajectories obtained by numerical integration are represented with blue, while the subset of F_1 from where all originating trajectories leave s_i through F_3 are represented with green.

tests involving $P \in F_2$ did not yield a feasible solution (*e.g.* due to computation errors induced by numerical integration). In all these situations, one could try to reduce the conservativeness by choosing a random set of points on F_1 and integrating forward with the hope that some of these points will exit s_i through F_3. Such an approach is not used here for two reasons: (i) the construction from Alg. 2 guarantees that whenever a positive probability is returned, this is the exact (up to integration and representation errors) exit probability, and (ii) for avoiding too many numerical integrations.

3.3 Procedure *find_probability* when $s_j = s_k$

When $s_j = s_k$, the exit probability is computed by Alg. 3. In this situation, the entry facet is the same as the desired exit one, and it is denoted by F_1 in this subsection. Basically, the probability of hitting s_j from s_i is equal to the ratio of subsegment of F_1 where continuous trajectories bounce back to s_j. Since the vector field on F_1 is a convex combination of its values at v_2 and v_3, in Alg. 3 ideas from lines 1-10 of Alg. 2 can be adapted.

Algorithm 3: Procedure *find_probability* when $s_j = s_k$

Input: S, adj, u_F $(= ctrl(s_i, s_j, s_k))$, s_i, s_j, s_k
Output: $p_{s_i \to s_j | s_k}$

1 **if** $n_1^T f(v_2) \geq 0 \ \wedge \ n_1^T f(v_3) \geq 0$ **then**
2 **return** $p_{s_i \to s_j | s_k} = 1$
3 **else if** $n_1^T f(v_2) < 0 \ \wedge \ n_1^T f(v_3) \geq 0$ **then**
4 find $P \in F_1$ s.t. $n_1^T f(P) = 0$
5 **return** $p_{s_i \to s_j | s_k} = \frac{\|P - v_3\|}{\|v_2 - v_3\|}$
6 **else if** $n_1^T f(v_2) \geq 0 \ \wedge \ n_1^T f(v_3) < 0$ **then**
7 find $P \in F_1$ s.t. $n_1^T f(P) = 0$
8 **return** $p_{s_i \to s_j | s_k} = \frac{\|P - v_2\|}{\|v_2 - v_3\|}$
9 **else**
10 **return** $p_{s_i \to s_j | s_k} = 0$

Thus, whenever the vector field value at v_2 and v_3 has projections with different signs on n_1, one can uniquely identify the point $P \in F_1$ from Alg. 3 where the vector field projection on n_1 is zero. If the vector field projections at v_2 and v_3 on n_1 have the same sign, the probability is 1 (when projections are positive) and it is 0 (when projections are negative). Fig. 4 illustrates the situations encountered by Alg. 3.

3.4 Possible extensions for computing history-based probabilities

This subsection indicates some additional ideas that have been investigated when computing abstractions G with history-based probabilities.

First, a refinement procedure that might increase the value of $p_{s_i \to s_j | s_k}$ computed by Alg. 2 from subsection 3.2 has been considered. The idea is to partition s_i in two subtriangles, based on the the point found on the entry facet F_1 which yields $p_{s_i \to s_j | s_k}$ in some cases encountered by Alg. 2. Depending on specific situations, this point was denoted by v_3, $P_{v_1\_back}$ or P_{back} in Alg. 2 and in Fig. 3, and it is generically denoted here by P, as represented in Fig. 5. The triangles partitioning s_i are determined by v_1,

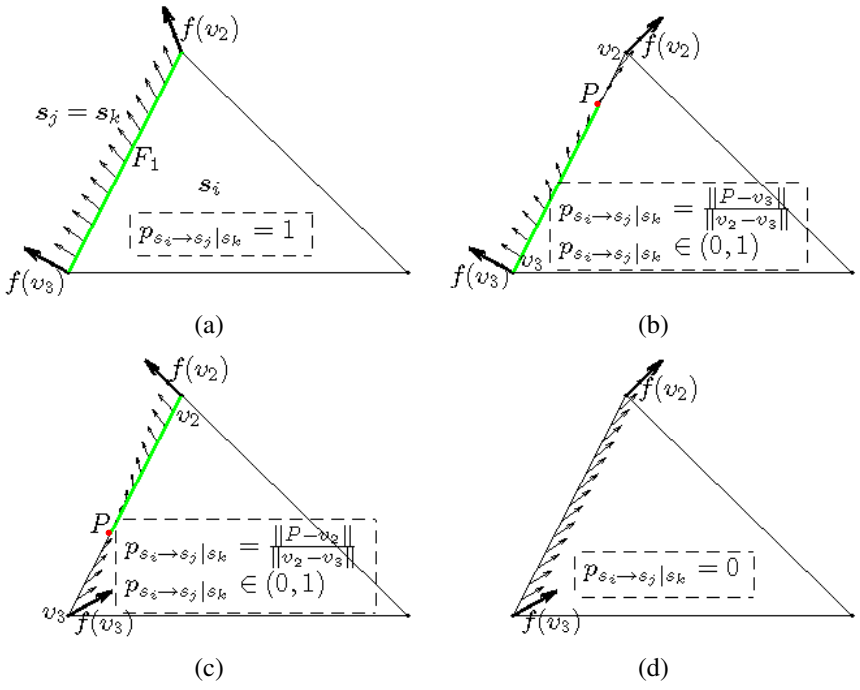


Figure 4: The four possible situations encountered by Alg. 3. The subset of entry facet from where all trajectories immediately bounce back to the previous triangle is represented with green. For each case, the output of Alg. 3 is given in the dashed box.

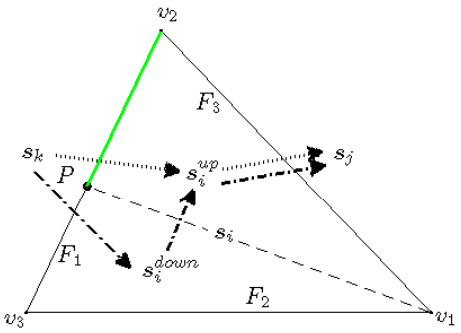


Figure 5: Refinement of triangle s_i. The dashed line linking P with v_3 splits s_i in two subtriangles. The dotted lines and the dash-dot lines suggest the possible sequences of triangles for reaching s_j from s_k, by crossing through s_i.

v_2, P (triangle s_i^{up}), and by v_1, P, v_3 (triangle s_i^{down}). Now, it is searched for a control law and a corresponding probability for exiting s_i^{down} to s_i^{up}, given that s_i^{down} was entered through the facet determined by P and v_3 (denote this probability by $p_{down \to up}$). Then, we search for a control law and the corresponding probability for exiting s_i^{up} to s_j, given that the entry facet to s_i^{up} is defined by v_1 and P (denote this probability by $p_{up \to s_j}$). The above control law and probabilities can be easily found by using ideas from Alg. 1 and Alg. 2. If $p_{s_i \to s_j | s_k}$ was returned by Alg. 2 for the original (not partitioned) s_i, based on the above refinement of s_i, $p_{s_i \to s_j | s_k}$ can be increased by adding to it the value $p_{down \to up} \cdot p_{up \to s_j}$.

If s_i is not partitioned and (after projecting a solution found on G to a sequence of continuous controllers) a continuous trajectory entering from s_k to s_i hits F_1 between P and v_3, then the control law assumed in Alg. 2 cannot steer to s_j. In such a case, by assuming the partition of s_i in s_i^{up} and s_i^{down}, the sequence of control laws determined when computing $p_{down \to up}$ and $p_{up \to s_j}$ may lead to hitting s_j. However, in the performed tests, such a refinement procedure did not yield a meritorious increase in history-based probabilities, when balanced with the increase in the necessary computation for creating G.

Secondly, some probabilities from G may have larger values if F_2 is not excluded from F_{exit} in Alg. 1, *i.e.* one allowed for possible obstacle intersections or for leaving the environment bounds. In such a case, the multiple choices from F_{exit} could lead to an increased value for $p_{s_i \to s_j | s_k}$. However, if the control law corresponding to this increased value includes possible collisions with obstacles that neighbor s_i, a corresponding "error" state and a transition to it should be included in G. This transition would be activated with the probability of hitting an obstacle or the environment bounds from s_i. Such an approach is not included in the proposed framework, because in the targeted problem the possibility of hitting any obstacle should have probability zero.

Finally, in some cases the probabilities could be increased by heuristically adjusting restrictions from [12, 13] for finding control laws. However, such control problems are not within the main scope of the proposed problem, because the current interest is in employing largely used methods, as are the ones suggested by [12], as premises for creating probabilistic abstractions on which one can solve various problems.

4 PETRI NET MODEL AND SOLUTION

Subsection 4.1 presents an algorithm to obtain a PN model for a team of agents evolving in a planar environment. In the PN model, each transition is modeling a cross by a robot of an edge between two adjacent regions, and a weight is associated to each transition based on the history-based probabilities computed in Sec. 3. Based on this PN model, an LPP is developed in subsection 4.2 to compute the optimal paths that may bring the team of robots from some given initial states to some desired final states. Optimality is understood here as a set of paths having the highest overall (product) probability. Since the paths may not be always followed because of the involved probabilities, subsection 4.3 provides an online algorithm that updates the solution based on the current execution.

4.1 Petri net model

For a simplex $s_i \in S$, denote by $adj(s_i)$ (with a slight abuse of notation) the set of all neighbor regions, *i.e.*, $adj(s_i) = \{s \in S | adj(s_i, s) \in adj\}$. Since in this paper triangular partitions are considered, $|adj(s_i)| \leq 3$. The PN model is composed by $\sum_{s_i \in S} |adj(s_i)|$ places. For each simplex s_i and for each $s_k \in adj(s_i)$ a new place p_i^k is defined. The number of tokens in this place, denoted by $m(p_i^k)$, indicates that $m(p_i^k)$ robots exist in region s_i, all of them entering in s_i from s_k. According to the previous observation, the number of places is upper bounded by $3 \cdot |S|$ (S is the set of regions of the environment). Therefore,

$$|P| = \sum_{s_i \in S} |adj(s_i)| \leq 3 \cdot |S| \tag{4}$$

For each place p_i^k, a number of $|adj(s_i)|$ transitions will be added. Each transition models the move from region s_i (that has been entered from s_k) to a neighbor region belonging to $adj(s_i)$ (including s_k). Thus, transition ${}_k^i t^j$ has only one input place (p_i^k) and only one output place (p_j^i) and its firing removes one token from p_i^k (since robot will leave s_i that has been entered from s_k) and puts one token in p_j^i (the robot moved to s_j and the previous region becomes s_i). Since each place p_i^k (with $s_k \in adj(s_i)$) has $|adj(s_i)|$ output transitions, there are $|adj(s_i)|^2$ transitions added for modeling the possibilities of leaving simplex s_i. Therefore, the total number of transitions is:

$$|T| = \sum_{s_i \in S} |adj(s_i)|^2 \leq 3^2 \cdot |S| \tag{5}$$

To each transition ${}_k^i t^j$ the probability $p_{s_i \to s_j | s_k}$ computed in Sec. 3 is associated.

Example 1 *Assume an environment composed by three simplices, as depicted in Fig. 6(a), and assume that the only history-based probabilities different than zero are:* $p_{1\to 2|3} = 0.4$, $p_{1\to 2|2} = 0.7$ *and* $p_{3\to 1|1} = 0.9$.

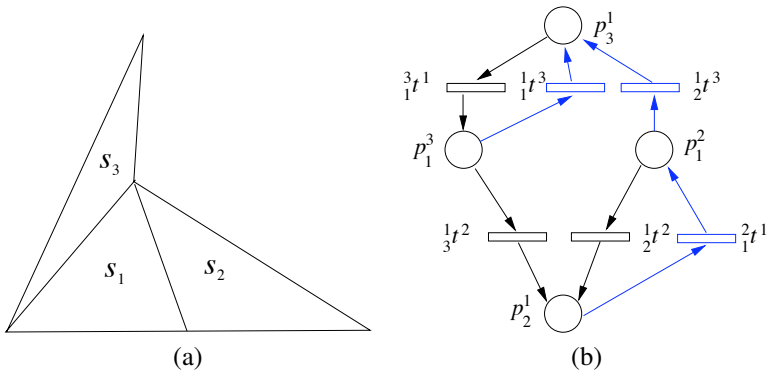
The PN model is composed by $|adj(s_1)| + |adj(s_2)| + |adj(s_3)| = 2 + 1 + 1 = 4$ *places and* $|adj(s_1)|^2 + |adj(s_2)|^2 + |adj(s_3)|^2 = 4 + 1 + 1 = 6$ *transitions. In particular,* $P = \{p_1^2, p_1^3, p_2^1, p_3^1\}$ *is the set of places and* $T = \{{}_1^3 t^1, {}_1^1 t^3, {}_2^1 t^3, {}_3^1 t^2, {}_2^2 t^2, {}_1^1 t^1\}$ *is the set of transitions. For example,* ${}_3^1 t^2$ *corresponds to the move from* s_1 *to* s_2*, given that* s_1 *has been entered from* s_3*. Therefore, the input place of* ${}_3^1 t^2$ *is* p_1^3 *(*s_1 *is the actual simplex entered from* s_3*) while the output place is* p_2^1 *(the new simplex is* s_2 *entered from*

Algorithm 4: Procedure *construct_PN*

Input: S, adj, p ; `/* the set of triangles, the adjacency relation and the history-based probabilities */`

Output: $\boldsymbol{Pre}, \boldsymbol{Post}, \boldsymbol{prob\_vect}$; `/* the Petri net model and the probability vector */`

Construct P by adding a place p_i^k for each $s_i \in S$ and $s_k \in adj(s_i)$;
Let $T = \emptyset$;
for each p_i^k **do**
    for each $s_j \in adj(s_i)$ **do**
        Add a transition ${}_k^i t^j$: $T = T \cup {}_k^i t^j$;
        $\boldsymbol{Pre}(p_i^k, {}_k^i t^j) = 1$; ($p_i^k$ is the input place of ${}_k^i t^j$);
        $\boldsymbol{Post}(p_j^i, {}_k^i t^j) = 1$; ($p_j^i$ is the output place of ${}_k^i t^j$);
        $\boldsymbol{prob\_vect}({}_k^i t^j) = p_{s_i \to s_j | s_k}$;
return $\boldsymbol{Pre}, \boldsymbol{Post}, \boldsymbol{prob\_vect}$



(a) (b)

Figure 6: An environment (a) and the corresponding Petri net model (b) considered in Ex. 1

s_1). The complete PN is given in Fig. 6(b). Notice that transitions ${}_1^1t^3$, ${}_2^1t^3$ and ${}_1^2t^1$ correspond to impossible movements (the corresponding history-based probabilities are zero). ■

Proposition 1 *The PN of a given environment obtained by applying Alg. 4 is a strongly connected* state machine.

Proof: *The result trivially holds based on the procedure of obtaining the PN model.* □

One immediate consequence of the previous result is that the PN system of an environment obtained by applying Alg. 4 is *bounded*, *i.e.*, the number of tokens in each place is upper-bounded by the number of robots. Therefore, its reachability graph is equivalent to a finite-state automaton. However, the number of states of this automaton could be every large and in the following we use the PN structure to compute the optimal plan to overcome the enumeration of the reachability states.

4.2 Optimal plan path

Let $R = \{r_1, r_2, \ldots, r_{|R|}\}$ be the set of robots and let $\mathcal{F} : S \rightarrow \mathbb{N}$ be a function that associates to each region (simplex) the number of robots that should be sent there, *i.e.*, if $\mathcal{F}(s_i) = n_i$ then n_i robots should eventually reach simplex s_i. We assume that $\mathcal{F}$ is consistent with the number of robots: $\sum_{\forall s_i \in S} \mathcal{F}(s_i) = |R|$. Using the obtained PN a path for each robot is computed such that the product of the individual path probabilities is as high as possible. First, the initial and final markings of the PN system are constructed. The initial marking, $\boldsymbol{m}_0$, is computed as follows: $\boldsymbol{m}_0$ is initialized with the null vector and then, for each robot, one token is added to the place corresponding to the region where the robot is initially placed. Obviously, it is necessary for each robot to assume a previous region since each place models also the history of one step.

Since for each region s_i, a number of $|adj(s_i)|$ places models that a robot is in region s_i entered from a $|adj(s_i)|$, it is not possible to impose an unique final place (state) for a robot. This requirement imposes restrictions that have to be satisfied by the final marking $\boldsymbol{m}$, which can be expressed by the following constraint belonging to an LPP:

$$\sum_{s_j \in adj(s_i)} \boldsymbol{m}(p_i^j) = \mathcal{F}(s_i), \forall s_i \in S$$

Therefore, trajectories for robots ensuring that all robots start from their initial positions and finish in regions whose collection is $\mathcal{F}$ are solutions of the following system of equalities and inequalities:

$$\begin{cases} \boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}, \\ \sum\limits_{s_j \in adj(s_i)} \boldsymbol{m}(p_i^j) = \mathcal{F}(s_i), \forall s_i \in S \end{cases} \tag{6}$$

where $\boldsymbol{\sigma}$ includes all firings encountered during movement of agents from initial to final states.

An objective function is constructed for being able to find paths having the overall maximum probability. Since the probability of each agent path is the product of the

encountered probabilities, and since $\boldsymbol{\sigma}$ includes movements of all robots, the highest probability path corresponds to $\boldsymbol{\sigma}$ that maximizes:

$$\Phi = \prod_{i=1}^{|T|} prob\_vect_i^{\sigma_i},$$

where $\boldsymbol{\sigma}$ is the firing count vector of (6), and each component σ_i is the number of times that the corresponding transition fires in the optimal path. As noted, firings from $\boldsymbol{\sigma}$ correspond to movements of all agents, and for finding individual paths for robots, just fire all enabled transitions until final states are reached, as explained towards the end of this section.

Obviously, this is not a linear objective function but, the maximization of

$$ln(\Phi) = \sum_{i=1}^{|T|} \sigma_i \cdot prob\_log_i = \boldsymbol{\sigma} \cdot \boldsymbol{prob\_log},$$

can be considered instead, where $\boldsymbol{prob\_log}$ is a vector containing the natural logarithm of the probability vector element by element. Since $\boldsymbol{prob\_vect}$ is a probability vector, $\mathbf{0} < \boldsymbol{prob\_vect} \leq \mathbf{1}$, the following is true: $-\boldsymbol{\infty} < \boldsymbol{prob\_log} \leq \mathbf{0}$. Observe that if a probability is zero, its logarithm is $-\infty$. Therefore, if the cost of the solution is finite, all transitions belonging to the path have positive probability and the path can be followed by the robots. On the contrary, a path containing a transition with probability zero means that the final state cannot be reached with the computed control laws.

Observe that the optimal solution of maximizing $ln(\Phi)$ is not unique, in general. In particular, if there exist cycles of transitions with associated probability equal to one, they can be added to the solution without changing the resulted path probability. In order to limit the number of solutions, the objective function is changed for minimizing the number of transitions belonging to the paths. Obviously, maximizing $ln(\Phi)$ is equivalent to minimizing

$$-ln(\Phi) = \boldsymbol{\sigma} \cdot (-\boldsymbol{prob\_log}),$$

where $(-\boldsymbol{prob\_log}) \geq \mathbf{0}$.

Proposition 2 *The shortest highest probability path for reaching the final regions cannot contain the firing of a transition more than r times (where r is the number of robots).*

Proof: *Consider first the path of one robot. If a transition appears twice in the firing sequence, then there exists an evolution of the form $s_k \rightarrow s_i \rightarrow s_j$ appearing twice. Since only the history of the previous state is considered, the probability of the evolution $s_k \rightarrow s_i \rightarrow s_j$ is the same both times, and the path probability cannot be improved by firing the corresponding transition two times. Having r robots, the maximum number of times of firing a transition in the shortest optimal path is r.* □ □

Define $\boldsymbol{\Delta} = M \cdot (-\boldsymbol{prob\_log}) + \mathbf{1} \cdot \epsilon$, where $\mathbf{1}$ is a vector having all elements equal to 1, ϵ is a small positive constant, and $M \in \mathbb{R}$ such that $M \geq r \cdot |T|$. Notice that $r \cdot |T|$ is an upper bound on the number of transitions that have to be fired in the optimal path according to Prop. 2. It can be shown that the minimization of $\boldsymbol{\sigma} \cdot \boldsymbol{\Delta}$ yields the highest probability paths ensuring also the minimization of the number of transitions fired, *i.e.*, it is not considering unnecessary firings of transitions with probability one.

Putting together, the following LPP can be used to compute the optimal paths of reaching the desired states:

$$\begin{aligned} \Gamma = \min \quad & \boldsymbol{\sigma} \cdot \boldsymbol{\Delta} \\ \text{s.t.} \quad & \boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}, \\ & \sum_{s_j \in adj(s_i)} \boldsymbol{m}(p_i^j) = \mathcal{F}(s_i), \forall s_i \in S \\ & \boldsymbol{m}, \boldsymbol{\sigma} \geq \mathbf{0} \end{aligned} \tag{7}$$

Recall that a square integer matrix is called *unimodular* if its determinant is equal to ± 1. A *totally unimodular matrix* is a matrix for which every square non-singular sub-matrix is unimodular.

Proposition 3 *If optimal solution of LPP* (7) *has a finite cost, then the obtained firing count vector* $\boldsymbol{\sigma}$ *corresponds to the maximum probability paths of the robots.*

Proof: *According to Prop. 1, the PN model is a state machine. The incidence matrix* $\boldsymbol{C}$ *of a state machine PN is an unimodular matrix having in each column maximum one element equal to* 1 *and one equal to* -1 *[14]. Therefore, the reachability set is an integer polytope [14]. If the simplex method is used to solve the LPP* (7)*, the optimal solution is an integer vertex. As mentioned before, a finite cost corresponds to a path with probability greater than zero.* □

Let Γ^* be the optimal solution of (7) and $N = \sum_{i=1}^{|T|} \sigma_i$, *i.e.*, the number of transitions that have to be fired in the optimal path. The probability of the optimal path is $e^{-(\Gamma^* - N)/M}$. Once having an optimal firing count vector $\boldsymbol{\sigma}$, the robot sequences of regions are obtained by enumerating regions corresponding to places that receive tokens while firing transitions in $\boldsymbol{\sigma}$. Notice that while finding individual paths there is no need to keep track of each individual token (as in a colored PN), and when there are multiple tokens in a place, random assignments of tokens to corresponding robots are used. The existence of such random assignments means is just a consequence of the fact that the individual paths may not be unique for a given optimum $\boldsymbol{\sigma}$. Also, observe that another possible source for non-uniqueness of individual paths is the fact that LPP (7) may have multiple optimal solutions. Once the individual paths are found, the control strategies for each robot have a hybrid nature, because each path consists of a finite number of regions, and in each region a different continuous control law has to be applied until the current simplex is left.

4.3 Online algorithm

Because of the probabilities on transitions, the desired simplices may not be always reached when the control laws corresponding to the solution of LPP (7) are applied. This subsection proposes an online algorithm that dynamically adjusts the trajectories based on the current position of the robots, such that if a desired probabilistic transition of PN is not actually followed, then the robot planning automatically adapts its trajectory.

The approach is centralized, meaning that there exists a central computing unit that can always communicate with the robots and that coordinates their movement. In the following, we first introduce notations that characterize the robots and the signals they

send to the central unit. Then we present the online algorithm that runs on the central unit.

During robotic movement, we assume that the position of each robot at any time is known through a function $f : R \times \mathbb{R}_{\geq 0} \rightarrow S$, *i.e.*, $f(r_i, \tau) = s_j$ means that the robot r_i is inside region s_j at time τ. The robots move and change their simplices due to the applied affine control laws. Let $move_i$ be the event generated by the robot r_i when it moves to a new region, formally defined by:

$$move_i(\tau) = \begin{cases} 0, & \text{if } f(r_i, \tau) = f(r_i, \tau^-) \\ 1, & \text{otherwise} \end{cases}$$

where $f(r_i, \tau^-)$ is the value of f at the previous time instant τ^-. If a robot moves from region s_j to s_k we assume that it can return to s_j after some time.

Algorithm 5 presents the main steps of the online approach that runs on the central unit and updates when necessary the robot paths (sequence of simplices to be followed). The new paths for robots (solution $\boldsymbol{\sigma}$) are computed only when a robot leaves its desired path (iteration conditions on line 12). The algorithm reiterates until each final region is reached by the intended number of robots. If the LPP (7) is unbounded then the final regions cannot be reached and the algorithm stops.

Algorithm 5: Online implementation

Input: $\boldsymbol{Pre}$, $\boldsymbol{Post}$, $\boldsymbol{prob\_log}$, $\mathcal{F}$; `/* the Petri net model, the logarithm of history-based probabilities and the desired simplices */`

Output: - ; `/* τ′ the robot movements */`

1 ; `// Let τ be the current time, updated by the central unit`

2 $\tau' := \tau$; `// τ′ is the previous time when an obstacle moved`

3 Compute control laws and $\boldsymbol{prob\_vect}$ by applying alg. in section 3;

4 **if** *eq.* (7) *is unbounded* **then**

5 ⌊ **return**; ; `// The final regions cannot be reached.`

6 **while** *final regions are not reached* **do**

7 Obtain new robot paths by solving (7);

8 **if** *eq.* (7) *is unbounded* **then**

9 ⌊ **return**; ; `// The final regions cannot be reached.`

10 **repeat**

11 Apply the control laws to the robots;

12 **until** *(*$\exists i \in \{1, \ldots, |R|\}$ *s.t.* $f(r_i, \tau)$ *does not belong to the trajectory)*;

5 CONSERVATISM AND COMPLEXITY

Our approach for solving the focused problem is conservative due to several reasons, which are discussed below in the order of their occurrence in our framework.

(i) Our approach assumes multiple identical robots, rather than accommodating agents with different dynamics. This leads to lower computation complexity, because the

probabilistic behavior is computed only for one robot, and the model for the team represents identical agents as tokens in a standard PN. An extension for considering different dynamics for the robots would imply a more complex and challenging problem. This is because standard Petri nets could not be used anymore, and colored Petri nets [15] may be necessary in order to distinguish between the different tokens of the model.

(ii) We assume a simpliceal partition of the continuous state space and we abstract each triangle to a single node in a graph. While this allows the construction of a decidable finite-state representation, it also leads to the following two aspects that limit the method.

(iii) When searching for continuous control laws for steering between adjacent simplices, we employ the well-established results from [12] rather than investigating other control possibilities or partition refining techniques.

(iv) The one step history-based exit probabilities are computed by assuming a uniform distribution along the entry facet of each simplex. Although this assumption is in general not true, it is a price paid for obtaining the finite abstraction of the environment. For longer sequences of simplices, the product of successive transition probabilities tend to become different than the real probability of following the sequence. In principle, this modeling limitation can be relaxed by assuming longer histories, *i.e.* by investigating (for each step in the history) the control law in the previous simplex and by finding the possible subset of the common facet where the continuous trajectories can cross from one simplex to another. However, we do not implement such longer histories because they would affect both the computational complexity for computing history-based probabilities and the size of the PN model (for a history length of k simplices, there would be up to 3^k places in PN for each simplex from partition).

(v) The optimality of the solution found in (7) only refers to maximizing the overall probability of following some robotic paths, and it does not include any penalty on the trajectory lengths. This is due to the difficulty of appropriately combining in a cost function the product of probabilities and the sum of expected traveled distances along firing sequences of PN. However, the optimization of path following probability reduces the number of iterations of the online Algorithm 5.

For analyzing the complexity of our approach, let $|S|$ denote the number of simplices in the partition. As stated in equation (4), the number of places in the PN model is upper bounded by $3 \cdot |S|$. Each simplex has at most 3 neighbors, and by accounting (5), the number of transitions in PN is upper bounded by $|T| \leq 9 \cdot |S|$.

As explained in Section 4, each transition has associated a history-based probability computed by Algorithm 1. A single run of this algorithm includes at most 4 iterations (maximum cardinality of F_{exit} from Algorithm 1, line 2). Each iteration embeds 3 LPPs for solving the control-to-facet problem, and the computation of exit probabilities (Algorithms 2 and 3), with some numerical integration routines called in Algorithm 2. Thus, for computing the history-based probabilities one has to solve at most $3 \cdot 4 \cdot 9 \cdot |S|$ = $108 \cdot |S|$ low-complexity LPPs (each having a small number of restrictions, given by the complexity of set U bounding the control inputs). We mention that for solving an LPP there exist polynomial time algorithms. For example, the Karmarkar algorithm [17] has a time complexity given by $O(n^{3.5} \cdot L)$, where n is the number of variables and L depends on the input length.

Therefore, the PN model construction is computationally tractable, since its complexity linearly increases with the number of partition simplices. An optimal movement plan is found by solving LPP (7) (which can be done in polynomial time), and if the paths are left the solutions are updated by iterating optimization (7), as in Algorithm 5. It is worth noticing that the number of robots r influences only the number of LPP constraints - in the worst case, LPP (7) will have $3 \cdot r + 1$ equality constraints (when $|\mathcal{F}| = r$, *i.e.* the destination simplices are different).

In the remainder of this section we briefly mention other techniques one can envision for solving such problems. One idea would be to model the environment using a finite state transition system and then applying a shortest path algorithm on the graph, *e.g.*, Dijkstra's algorithm. Notice that the finite state transition system that is obtained in the case of one robot is very similar with the PN model: just remove PN transitions, merge the corresponding input/output arcs, and weight each resulted arc with the corresponding element from $\boldsymbol{prob\_log}$ (probability logarithms ensuring optimization based on product rather than sum of individual probabilities). Another idea is to model the behavior of a single robot by a Markov Decision Process (MDP), *e.g.*, as used in [8]. This formalism can be viewed as a discrete event system where transitions are probabilistically executed (rather than having a probability-based cost). Since MDP requires static (history-free) probabilities, the structure of the MDP model mimics the one of the above-mentioned transition system. The advantage is that a control policy for the MDP can be found such that the system is steered to a desired state without the need of having an online updating algorithm. While the choice between an offline generated policy and an online replanning procedure is generally debatable [27], we prefer for our method an online planning algorithm, due to the following reasons.

Both methods mentioned above work with similar complexity for the case of a single robot. However, the PN approach can be easily extended to the case in which more than one robot is deployed since the structure is the same and only the initial and final markings are changed. Thus, the behavior of all agents is modeled by a simple increase in the number of tokens. As explained above, the change in complexity of solving (7) with respect to the number of robots r is negligible.

However, in the case of transition system- or MDP- based approaches the problem is not so simple, mainly because the optimum allocation of robots to required destinations has to be found. A first solution is to successively consider all possible robot-destination pairs and find the optimum one, but this would require a combinatorial number of solutions to individual (single agent) problems. An alternate solution is to to model the behavior of all robots by a product of individual models, but this yields to a possible explosion in the number of states [20]. Hence, the PN approach provides a considerable computational complexity advantage.

The number of iterations of the online replanning algorithm is not priori known. An upper bound can be given but, in general, this upper bound would be very far from the real value. Nevertheless, in many cases it is preferable to solve many low-complexity problem instead of solving a single but very complex problem.

6 EXAMPLE

Consider the planar environment from Fig. 7, cluttered with 8 obstacles and whose free space is composed by 50 simplices $s_1, \ldots, s_{50}$. The following dynamics are considered in all regions:

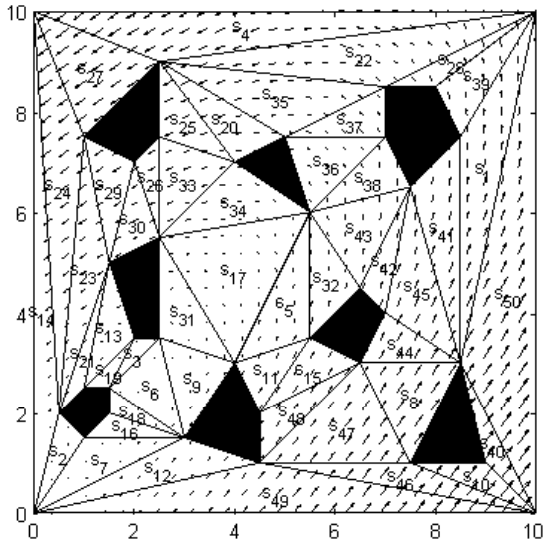


Figure 7: A planar environment with the free space composed by 50 simplices. The vector field of uncontrolled system in represented with black arrows.

$$\dot{x} = \begin{bmatrix} 0.5 & -0.6 \\ 0.7 & -0.4 \end{bmatrix} \cdot x + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot u + \begin{bmatrix} 0.3 \\ -0.3 \end{bmatrix} \tag{8}$$

The control u is bounded by a disc in $\mathbb{R}^2$, centered at origin and with radius 3, which is approximated by a 12-sided regular polygon. The vector field corresponding to the uncontrolled system is represented in Fig. 7.

Controllers and transitions probabilities are computed as described in Sec. 3, and the corresponding PN is obtained as in Sec. 4. The PN has 114 places and 270 transitions. From these transitions, the following have probabilities between 0 and 1: ${}^{14}_{24}t^{24} = 0.17$; ${}^{20}_{35}t^{35} = 0.74$; ${}^{22}_{4}t^{28} = 0.01$; ${}^{23}_{24}t^{21} = 0.75$; ${}^{23}_{29}t^{21} = 0.89$; ${}^{23}_{24}t^{29} = 0.25$; ${}^{23}_{29}t^{29} = 0.23$; ${}^{24}_{14}t^{23} = 0.44$; ${}^{24}_{23}t^{23} = 0.84$; ${}^{24}_{27}t^{23} = 0.22$; ${}^{25}_{20}t^{20} = 0.48$; ${}^{26}_{30}t^{33} = 0.68$; ${}^{29}_{23}t^{30} = 0.4$; ${}^{30}_{26}t^{26} = 0.73$; ${}^{30}_{29}t^{26} = 0.50$; ${}^{33}_{25}t^{25} = 0.61$; ${}^{33}_{34}t^{25} = 0.44$; ${}^{41}_{1}t^{45} = 0.5$; ${}^{41}_{45}t^{45} = 0.38$; ${}^{45}_{41}t^{42} = 0.68$; ${}^{45}_{44}t^{42} = 0.91$; ${}^{46}_{49}t^{10} = 0.53$; ${}^{46}_{49}t^{47} = 0.5$; ${}^{46}_{47}t^{49} = 0.01$; ${}^{46}_{49}t^{49} = 0.04$; ${}^{47}_{46}t^{46} = 0.26$; ${}^{47}_{48}t^{46} = 0.04$; ${}^{47}_{46}t^{48} = 0.69$; ${}^{49}_{12}t^{12} = 0.95$. There are 42 transitions with null probabilities, namely: ${}^{4}_{27}t^{22}$; ${}^{8}_{44}t^{47}$; ${}^{8}_{47}t^{47}$; ${}^{10}_{40}t^{46}$; ${}^{10}_{46}t^{40}$; ${}^{10}_{46}t^{46}$; ${}^{12}_{7}t^{49}$; ${}^{12}_{49}t^{7}$; ${}^{14}_{2}t^{24}$; ${}^{14}_{24}t^{2}$; ${}^{15}_{11}t^{48}$; ${}^{15}_{48}t^{11}$; ${}^{20}_{25}t^{35}$; ${}^{22}_{35}t^{28}$; ${}^{23}_{21}t^{29}$; ${}^{24}_{14}t^{27}$; ${}^{24}_{23}t^{27}$; ${}^{24}_{27}t^{27}$; ${}^{25}_{20}t^{33}$; ${}^{25}_{33}t^{20}$; ${}^{27}_{4}t^{4}$; ${}^{27}_{24}t^{4}$; ${}^{33}_{26}t^{25}$; ${}^{35}_{20}t^{22}$; ${}^{35}_{20}t^{37}$; ${}^{35}_{22}t^{37}$; ${}^{40}_{10}t^{10}$; ${}^{40}_{50}t^{10}$; ${}^{44}_{8}t^{8}$; ${}^{44}_{45}t^{8}$; ${}^{45}_{41}t^{44}$; ${}^{45}_{42}t^{44}$; ${}^{45}_{44}t^{44}$; ${}^{46}_{10}t^{47}$; ${}^{46}_{10}t^{49}$; ${}^{46}_{47}t^{10}$; ${}^{47}_{8}t^{46}$; ${}^{47}_{8}t^{48}$; ${}^{49}_{46}t^{12}$; ${}^{50}_{1}t^{40}$; ${}^{50}_{40}t^{1}$; ${}^{50}_{40}t^{40}$. The remaining 199 transitions have probability one. As explained in Sec. 4, transitions having associated zero probabilities could have been not introduced in the PN systems. However, their inclusion guarantees that PN is a strongly-connected state machine, and the complexity of finding a transition sequence to reach a final marking is significantly reduced because one has to solve an LPP instead of an integer programming problem. Moreover, spurious markings cannot appear, and one can consider a single vector containing all fired transitions instead of a sequence of firing vectors (each vector corresponding to the firing of only one transition).

For the environment from Fig. 7 five point robots are considered (with dynamics

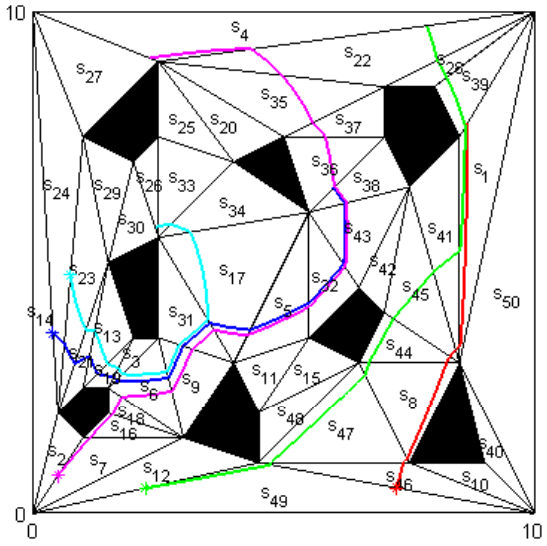


Figure 8: Controlled trajectories that reach the desired simplices. For each trajectory, the initial condition is marked with "*", and the evolution is stopped when the last simplex is entered. Magenta corresponds to the first robot, cyan to the second, blue to the third, red to the fourth and green to the fifth robot.

(8)), initially situated in s_7 (entered from s_2), s_{23} (entered from s_{24}), s_{24} (entered from s_{14}), s_{46} (entered from s_{49}), and s_{49} (entered from s_{12}). The robots have to reach simplices s_4, s_{27}, s_{30}, s_{36} and s_{39}, without having any robot specifically assigned to a desired simplex. By using the LPP (7) and by constructing individual paths as mentioned in Sec. 4, the following sequences of regions to be followed by the robots are obtained:

- **Robot 1**: $(s_2) \rightarrow s_7 \rightarrow s_{16} \rightarrow s_{18} \rightarrow s_6 \rightarrow s_9 \rightarrow s_{31} \rightarrow s_{17} \rightarrow s_5 \rightarrow s_{32} \rightarrow s_{43} \rightarrow s_{38} \rightarrow s_{36} \rightarrow s_{37} \rightarrow s_{35} \rightarrow s_{22} \rightarrow s_4 \rightarrow s_{27}$. This path has a probability to be followed equal to 1.
- **Robot 2**: $(s_{24}) \rightarrow s_{23} \rightarrow s_{21} \rightarrow s_{13} \rightarrow s_{19} \rightarrow s_3 \rightarrow s_6 \rightarrow s_9 \rightarrow s_{31} \rightarrow s_{17} \rightarrow s_{34} \rightarrow s_{33} \rightarrow s_{26} \rightarrow s_{30}$, the probability of the path being 0.75.
- **Robot 3**: $(s_{14}) \rightarrow s_{24} \rightarrow s_{23} \rightarrow s_{21} \rightarrow s_{13} \rightarrow s_{19} \rightarrow s_3 \rightarrow s_6 \rightarrow s_9 \rightarrow s_{31} \rightarrow s_{17} \rightarrow s_5 \rightarrow s_{32} \rightarrow s_{43} \rightarrow s_{38} \rightarrow s_{36}$, with probability 0.33.
- **Robot 4**: $(s_{49}) \rightarrow s_{46} \rightarrow s_{47} \rightarrow s_8 \rightarrow s_{44} \rightarrow s_{45} \rightarrow s_{41} \rightarrow s_1 \rightarrow s_{39}$, with probability 0.5.
- **Robot 5**: $(s_{12}) \rightarrow s_{49} \rightarrow s_{46} \rightarrow s_{47} \rightarrow s_8 \rightarrow s_{44} \rightarrow s_{45} \rightarrow s_{41} \rightarrow s_1 \rightarrow s_{39} \rightarrow s_{28} \rightarrow s_{22} \rightarrow s_4$, having probability 0.5.

The above paths have a combined probability (product of individual probabilities) of 6%, and they include a total number of 65 simplices to be visited. In this case the sequences of regions yielding this global optimum are not unique. This aspect can be quickly understood by observing that some paths include identical sequences of

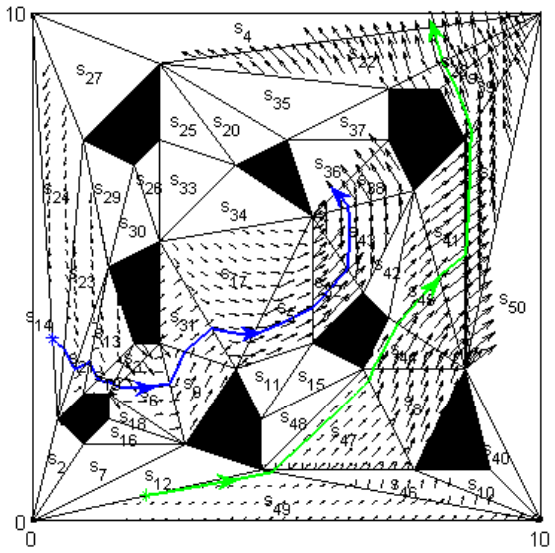


Figure 9: Vector fields corresponding to sequences of simplices to be followed by robots 3 and 5 from Fig. 8, and their continuous trajectories (blue for robot 3, green for robot 5).

simplices, and after these sequences the last parts of the paths can be swapped without affecting the overall optimality. Fig. 8 illustrates some trajectories of robots obtained by applying control laws corresponding to the found sequences of simplices. The vector fields induced by control laws for robots 3 and 5 are given in Fig. 9. It is worth mentioning that the probability returned by the proposed approach is not equal with the real probability of following the corresponding sequence of simplices, because history-based probabilities with history consisting of only one simplex were assumed. However, the obtained probability measures the chance of following the actual paths without having to update them by a reiteration of Algorithm 5. Also, the path probability does not reflect the chance of satisfying the requirement, because whenever the paths are left they are updated by the online algorithm. For example, by simulating trajectories from multiple initial conditions for robot 3 (points on the common facet of s_{24} and s_{14}), the above path (returned by the first iteration of Algorithm 5) was followed in about 44% cases (*e.g.* if the initial condition is chosen in the upper part of line segment between s_{24} and s_{14}, the vector field in simplex s_{24} induces simplex s_{14} being hit instead of s_{23}). On the same lines, for robot 5, if simplex s_{46} would be entered from s_{49} through a point with coordinates close to $[10, 0]^T$, then s_{46} would be left to s_{10} rather than s_{47}, due to vector field shown in Fig. 9. An exact measure of the real probability of following an obtained sequence of simplices could be found by an iterative backwards integration procedure that starts from the last facet of the path (in case of path for robot 3 the common facet of s_{36} and s_{38}). However, such an operation is not necessary, because whenever a robot leaves its current path through simplices, a new optimization (7) is triggered. Also, a more detailed analyzis of the real probability of following a path would go beyond the scope of the current paper, which provides a method for obtaining a finite-state probabilistic model and for finding optimum paths in such an abstraction.

7 CONCLUSIONS

This paper presents a probabilistic abstraction approach for planning and controlling mobile robots with affine dynamics. It is assumed that the environment is partitioned into triangular regions, each region corresponding to a node in a graph. If from a region s_i there exists a control law able to drive the robot to an adjacent region s_j, then there exists an arc between the corresponding nodes of the graph. In many cases it may be impossible to find a control law for crossing only the border (facet) between s_i and s_j, but it may be possible to find a control law for exiting through a set of facets of s_i. In such cases, the probability of crossing the desired facet under the designed control law is computed algorithmically. In order to reduce the conservativeness, history-based probabilities that depend on the previous region are considered. Finally, a PN model is constructed and the task of reaching a set of target regions is achieved by solving an LPP. The optimality of the solution is guaranteed by specific properties of the finite-state model. Therefore, the main benefits of the presented work are (i) a method for constructing probabilistic abstractions for affine systems and (ii) a procedure for constructing PN models for teams of agents and for finding optimal probabilistic paths by using low-complexity algorithms. Enabled by the low-complexity of the PN model, future work can be conducted towards extending the range of handled specifications to high-level formal languages.

Acknowledgements

The authors thank the anonymous reviewers for their useful comments and suggestions. This work has been partially supported at the Technical University of Iasi by the CNCS-UEFISCDI grant PN-II-RU PD 333/2010 and at University of Zaragoza by the CICYT - FEDER grant DPI2010-20413.

References

[1] C. Belta and L.C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *43rd IEEE Conference on Decision and Control*, pages 467–472, Paradise Island, Bahamas, 2004.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

[3] Cgal community. Cgal, Computational Geometry Algorithms Library. `http://www.cgal.org`, 2011.

[4] H. Costelha and P. Lima. Robot task plan representation by Petri nets: modelling, identification, analysis and execution. *Journal of Autonomous Robots*, pages 1–24, 2012.

[5] R.V. Cowlagi and P. Tsiotras. Kinematic feasibility guarantees in geometric path planning using history-based transition costs over cell decompositions. In *American Control Conference (ACC)*, pages 5388–5393, 2010.

[6] R.V. Cowlagi and P. Tsiotras. Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles. *IEEE Transactions on Robotics*, 28(2):379–395, 2012.

[7] J. Ding, E. Li, H. Huang, and C.J. Tomlin. Reachability-based synthesis of feedback policies for motion planning under bounded disturbances. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2160–2165, 2011.

[8] X. C. Ding, S. L. Smith, C. Belta, and D. Rus. LTL control in uncertain environments with probabilistic satisfaction guarantees. In *18th IFAC World Congress*, Milan, Italy, 2011.

[9] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4885 – 4890, 2005.

[10] K. Fukuda. CDD/CDD+ package. *http://www.ifor.math.ethz.ch/∼fukuda/cdd_home/*, 2011.

[11] B. Gerkey and M. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[12] L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transactions on Automatic Control*, 51:938–948, 2006.

[13] L. C. G. J. M. Habets and J. H. van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40:21–35, 2004.

[14] A.J. Hoffman and J.B. Kruskal. Integral boundary points of convex polyhedra. In H.W. Kuhn and A.W. Tucker, editors, *Linear Inequalities and Related Systems. Annals of Mathematics Studies*, volume 38, pages 223–246. Princeton University Press, 1956.

[15] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*. EATCS Monographs on Theoretical Computer Science. Springer, 1994.

[16] B. Johnson and H. Kress-Gazit. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *Robotics: Science and Systems*, Los Angeles, CA, 2011.

[17] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the* 16^{th} *annual ACM symposium on Theory of computing*, STOC '84, pages 302–311, New York, NY, USA, 1984.

[18] G. Kim and W. Chung. Navigation behavior selection using generalized stochastic Petri nets for a service robot. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, 37(4):494–503, 2007.

[19] J. King, R. Pretty, and R. Gosine. Coordinated execution of tasks in a multiagent environment. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 33(5):615–619, 2003.

[20] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Trans. on Robotics*, 26(1):48–61, 2010.

[21] M. Kloetzer, C. Mahulea, C. Belta, and M. Silva. An Automated Framework for Formal Verification of Timed Continuous Petri Nets. *IEEE Transactions on Industrial Informatics*, 6(3):460–471, 2010.

[22] M. Kloetzer, C. Mahulea, and O. Pastravanu. A probabilistic abstraction approach for planning and controlling mobile robots. In *IEEE Conf. on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Toulouse, France, 2011.

[23] M. Kloetzer, C. Mahulea, and O. Pastravanu. Software tool for probabilistic abstraction for planning and controling mobile robots. *http://webdiis.unizar.es/∼cmahulea/research/ prob_abstr.zip*, 2011.

[24] S. Konur, C. Dixon, and M. Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199–213, 2012.

[25] M. Lahijanian, C. Belta, and S. Andersson. A probabilistic approach for control of a stochastic system from LTL specifications. In *IEEE Conf. on Decision and Control*, pages 2236 – 2241, Shanghai, China, 2009.

[26] S. M. LaValle. *Planning Algorithms*. Cambridge, 2006. *http://planning.cs.uiuc.edu*.

[27] I. Little and S. Thiébaux. Probabilistic planning vs replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[28] W. Liu, A.F.T. Winfield, and J. Sa. Modelling swarm robotic systems: A case study in collective foraging. In *Towards Autonomous Robotic Systems (TAROS)*, pages 25–32, 2007.

[29] C. Mahulea and M. Kloetzer. A probabilistic abstraction approach for planning and controlling mobile robots. In *IEEE Conf. on Emerging Technologies and Factory Automation (ETFA)*, Krakow, Poland, 2012.

[30] A. Makhorin. GLPK-GNU linear programming kit. *http://www.gnu.org/software/glpk*, 2007.

[31] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[32] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi. Multi-robot motion planning: A timed automata approach. In *IEEE Conf. on Robotics and Automation*, pages 4417–4422, New Orleans, LA, 2004.

[33] E. Rippel, A. Bar-Gill, and N. Shimkin. Fast graph-search algorithms for general-aviation flight trajectory generation. *Journal of Guidance, Control, and Dynamics*, 28(4):801–811, 2005.

[34] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[35] M. Silva. Introducing Petri nets. In *Practice of Petri Nets in Manufacturing*, pages 1–62. Chapman & Hall, 1993.

[36] M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of net systems. In G. Rozenberg and W. Reisig, editors, *Lectures in Petri Nets. I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 309–373. Springer, 1998.

[37] The MathWorks. MATLAB® 2010b. Natick, MA, 2010.