28th International Conference on Automated Planning and Scheduling

June 24-29, 2018, Delft, the Netherlands

DMAP 2018

Proceedings of the 6th Workshop on

Distributed and Multi-Agent Planning

Edited by:

Michal Stolba and Antonin Komenda

Collaborative Optimization and Planning for Transportation Energy Reduction

Filip Dvorak
Oqton
832 Sansome Street
San Francisco, California 94111
filip@dvorak.fr

Shiwali Mohan, Victoria Bellotti and Matthew Klenk
Palo Alto Research Center
3333 Coyote Hill Rd, Palo Alto, CA 94304
{mohan,bellotti,klenk}@parc.com

Abstract

Transportation accounts for nearly a third of greenhouse gas emissions in the United States. Increasing urbanization and the expansion of transportation services provide fertile ground for developing assistants for personalized transportation decision-making that improves overall efficiency and user satisfaction. In this paper we describe COPTER, a framework that models all transportation options to reduce city-wide energy consumption and congestion by suggesting alternative, energy-saving routes that are accepted by individuals. To accomplish this, our approach begins with human studies to develop a user model and identify requirements for our planning system. Next, drawing on existing research, we created a novel personalized, multi-modal, collaborative planning system to explore the tradeoff between user acceptance of routes and energy savings. Our combination of single user planning and cooperative planning supports running multiple solvers in parallel to support anytime performance. Finally, we present a simulation study in the Los Angeles area demonstrating a 9% reduction in delay and a 4% reduction in fuel consumption with only 4% of travelers changing behavior. We conclude with a discussion of our planned pilot deployment.

Transportation is one of the largest consumers of energy in the world (29% of energy consumption in USA in 2016)[1] and efficiency improvements in transportation are directly measurable in terms of energy savings. Congestion in the United States wastes 6.9 billion hours and 3.1 billion gallons of fuel per year (Schrank et al. 2015). On the other hand, areas of urban transportation networks are underutilized even when other areas are congested. Meanwhile, new transportation services are being created by public and private entities including bike share, car share, ride hailing, and dynamic carpooling to complement the private vehicle and public transit currently used for most trips. To facilitate decision-making, "smart cities" companies are developing mobility marketplaces to aggregate these offerings into single markets similar to what Travelocity does for air travel.

This combination of a large decision space and energy inefficiencies produces a compelling area to apply AI for social good. Our goal is to develop technology that influences travelers to make decisions that reduce the energy consumption of the entire system. COPTER (Collaborative Optimization and Planning for Transportation Energy Reduction) extends AI techniques from route planning to identify multi-modal plans and draws on human-centered AI work to model influence in transportation decision-making. The contributions of this paper include:

- A formulation of transportation influence problem.
- An approach to understanding transportation decision-making.
- Four desiderata for AI planning and optimization to support transportation decision-making: helpful, informative, multi-modal, and personal.
- A time-dependent, collaborative, multi-modal with personalized cost over multiple agents AI planning system that accounts for system-wide energy consumption.

The rest of this paper is organized as follows. First, we discuss how our technology would interact with existing services. Then, we outline our approach for understanding the requirements for influencing human transportation decision-making. Next, we describe how our COPTER combines previous AI planning research and defines new problems in collaborative planning. Finally, we present a simulation study over the Los Angeles region that demonstrates the potential savings in time and energy. We close with a discussion of our pilot deployment.

Influencing Traveling Behavior with COPTER

Transportation services are changing with ride hail companies (e.g., Lyft), commuter shuttles (e.g., Chariot), app-mediated carpooling (e.g., Waze Rider), public/private bike share companies (e.g., GoBike), car sharing (e.g., Zipcar), and personal travel options including electric bikes and skateboards all supplementing the established modes of walking, public transit, and personal vehicles. New applications like SkedGo's TripGo and Daimler's Moovel provide a unified view of these urban transportation offerings along with single point of payment. This emerging commercial landscape provides a rich set of research challenges including questions around efficient allocation of transportation resources and fairness.

Figure 1 outlines the key components of the COPTER system. COPTER's user model predicts future travel needs of individuals as well as their resources and preferences. From the aggregate demand, the planner identifies alternative routes for each user, including carpools and vanpools

[1]U.S. Energy Information Administration, eia.gov

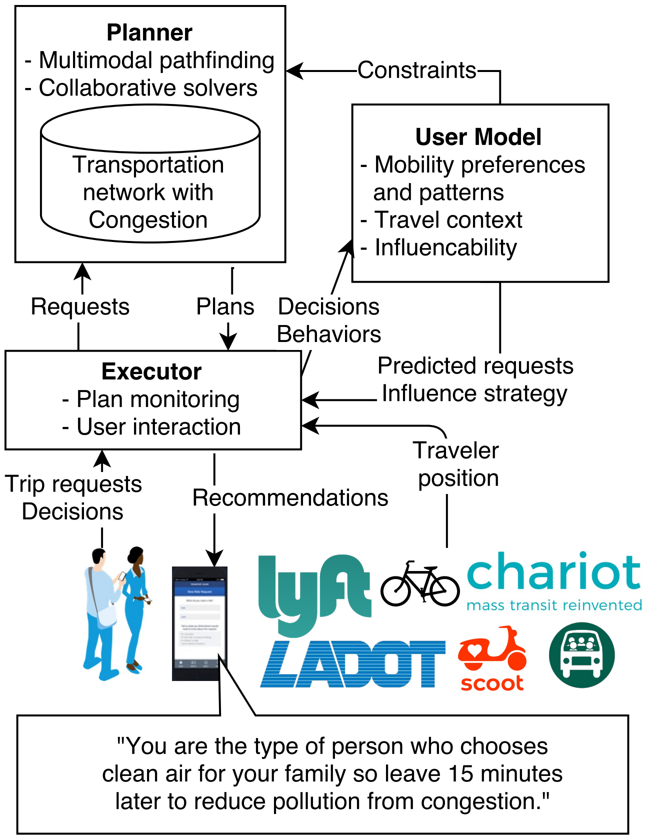


Figure 1: The COPTER architecture delivers personalized route recommendations to users that minimize the expected energy consumption across the system and tracks users to improve user models as well as provide real time updates to changing network conditions.

involving multiple individuals. The route that will lead to the largest expected energy reduction is sent as a recommendation through a phone application prior to the traveler's departure. COPTER will monitor users' travel to collect feedback to improve the user model for future interactions and provide real-time responses to changing transportation conditions. At present, we have identified system desiderata and implemented the AI planning system.

Uncovering AI System Desiderata

The transportation mode and route people pick for their trips determines not only their energy costs, but also impacts energy consumption across the network. This choice is modulated by several factors, including among others the distance they have to travel, accessibility of various transportation modes, income, and the purpose of undertaking a trip (which might necessitate transporting other people or things). To convince an individual to make a more sustainable transportation decision for a trip, it is important to understand how she typically makes decisions and present an alternative that still satisfies her needs. Below, we describe our insights from in-person interviews and an in-depth survey. We, then, derive desiderata for AI solutions to influence people's transportation decisions.

Interviews

We interviewed 20 people (7 women, 13 men) in the age range 21-79 (mean 37.5, standard deviation 16.8) in Los Angeles. Participants varied in terms of their commute flexibility with 9 participants in highly flexible occupations, 4 in medium, and 7 in low flexibility with respect to arrival and departure times. All the participants reported to be living within a mile of a bus stop. 8 people expressed a strict preference for driving regardless of the context they undertook the trip in. 2 people expressed a preference for working from home and driving for non-work related trips. 4 people expressed a clear preference for taking public transit, 3 for walking/biking, 2 for a ride service or carpool, and 1 for driving/biking.

The variety in people's preferences strongly suggests that any good solution to the problem of recommending more sustainable travel behaviors overall would need to consider a variety of alternate modes. Several people talked about the cognitive load of planning a trip which especially affects the decision of choosing a sustainable mode: *"I certainly see congestion over on the Sepulveda parallel to the 405. Very, very, heavy. Puts almost an extra hour on the bus trip, but I don't really know what the solution is. They've already widened the 405"*. Many brought up cost and time trade-off they must make: *"Mainly, I can't think any other way. Well, taxi. That would be prohibitively expensive."*, *"It's really time convenience and cost are the things that we would..."*, *"The first issue I think would be efficiency, the time of route"*. Most importantly, they expressed that they would be open to alternative ways to make their trip if a good alternative was suggested without them having to invest time thinking about it: *"If you were to tell me there was a different way, I would probably take it. If you were to say this is a better way to travel or more efficient way, then yeah I would be open to that."*, *"So, as much information as you can take into account, I would be in favor of using that...I want as much intel as possible."*, *"if there's a new mapping service out that would be more efficient or would expedite the rate at which I got somewhere"*.

Survey

The diversity in preference and context uncovered in our interviews informed our design of a survey to identify what variables and what relationships between them to include in our user model. The survey covered 112 self-reported binary, categorical and scalar variables including public transportation availability and proximity, availability of other transportation modes, typical commute modes, commute constraints, other reasons for travel, reasons for transport mode choices, conditions that might change mode choices and preferred activities engaged in while traveling and more.

We deployed the survey via Qualtrics[2] to 677 participants from urban areas. After eliminating responses that were completed in less than a reasonable threshold time, included straightlining, or contained contradictory responses, we obtained 235 responses (128 women & 106 men; age 18-79, mean 46.51, sd 13.5) for analysis. Participants engaged in

[2]https://www.qualtrics.com/

broad range of transportation behaviors including individuals who used the following modes more than 50% of the previous weeks travel: walking, driving, bus, train, taxi, carpool, bicycle and motorcycle. We performed multi-variate multiple linear regression and chi-square analysis focusing on four categories of factors that impact people's transportation decisions:

- **Personal, static context:** This context includes factors related to a person's job, lifestyle, experience etc. This context does not change from day to day. We found that distance of residence from work as well as the time it takes to travel that distance significantly influences a person's overall mode usage. Additionally, other considerations such as health, ability to work en route, being able to avoid congestion, low risk of being late, reliability of return, and parking are important for mode choice.
- **Situational, static context:** This captures the characteristics of the transportation network a person is embedded in including availability of transportation resources such as a car, bicycle, etc. This does not change from day to day. Not surprisingly, our findings show that distance from transit stops and frequency of transit is correlated positively with choosing public transit for travel. Similarly, having bicycle-friendly routes is postively correlated with choosing sustainable transportation modes.
- **Dynamic context:** This captures aspects of a person's lifestyle, weather, transportation network that vary from day to day. Our analysis suggests that weather, purpose of the trip, and availability of parking at the destination greatly impact mode choice. Particularly, pleasant weather increases the likelihood of walking and biking. Shopping reduces the chances a person would use sustainable modes. An outing where alcohol is expected greatly reduces the likelihood of driving and increases use of ride sharing services. Limited parking greatly reduces the chances that a person will drive.
- **Personality:** Following recent work that suggests that personality may be inferred from online behavior (Kosinski, Stillwell, and Graepel 2013) and may influence transportation choices (Johansson, Heldt, and Johansson 2006), we also included three standard personality scales: TIPI (Gosling, Rentfrow, and Swann 2003), responsibility for events in their lives [3], and susceptibility to persuasion (Cialdini 2001). We did not find any significant impact of aspects of personality on mode choice.

Desiderata

Based on our interview data and survey analysis, we propose that an AI solution for influencing people to take more sustainable modes must:

D1 Be helpful: Our interviews emphasize that COPTER should reduce human cognitive effort by presenting a well-defined, specific plan from source to destination. It should account for departure and arrival times of public transit as well as congestion of roads.

D2 Be informative: COPTER should make explicit the tradeoffs that people are concerned about. Interview data suggests it should indicate the cost and duration of the trip. It may also indicate congestion level, energy expended, etc based on what a person thinks is important for them.

D3 Be multi-modal: Given the diversity of modes indicated in the survey, COPTER should plan over a variety of sustainable modes including ride sharing and carpools instead of limiting choices to a few public transit options.

D4 Be personal: Our survey data suggested it should take into account a person's specific context and preferences in finding a suitable transportation plan. More specifically, it should represent and plan for a person's personal static, situational static, and dynamic context as described above.

In the following sections, we propose how AI planning approaches can be incorporated in a transportation planning system such as they meet the desiderata identified here.

Energy Efficient Collaborative Routing

Efficient routing in transportation networks has been an important domain for algorithm development, starting with Dijkstra's algorithm (Dijkstra 1959) and continuing with methods to exploit additional precomputation and heuristics that now enable today's planners to route journeys for continent-size networks while guaranteeing an optimal solutions in fractions of seconds (Bast et al. 2016).

To meet the above desiderata, our planning system must exhibit the follow properties:

- **Time-dependent multi-modal network [D1, D2, D3].** We consider our network to be composed from multiple transportation networks, where some of them are time-dependent (buses, trains), and some of them time-independent (walking, biking). The time dependency has two dimensions, one is represented by scheduling constraints (there is a set of times when we can take certain mode of transport such as bus or a train). Second dimension of time-dependency is the variability of travel times by time of the day (e.g. congestions related delays). The time-dependency of transportation networks becomes an easy extension when we look only for the earliest arrival times and there are no situations when leaving later would lead to arriving sooner (first in, first out). Extensions relaxing this assumption use waiting constraints (Dean 2004), but are not considered at this time. To provide multi-modal recommendations, we use the standard approach of merging multiple transportation networks (Dibbelt, Pajor, and Wagner 2015).
- **Personalized cost-function [D4].** We consider each traveler to have different preferences for various transportation modes and to have different resources at his or her disposal (e.g., having a car, bike, electric skate-board) (Liu, Fritz, and Klenk 2018).
- **Collaboration [D3].** Typical solutions either consider travelers in isolation or abstract all travelers into traffic flows. Instead, COPTER reasons about journeys where users actively interact with each other and agents outside

[3] `http://ipip.ori.org/`

of the system, by means such as carpooling and vanpooling and choosing to decrease congestion on the road. Significant research effort has been invested in computing efficient collaborative plans in carpooling (Bit-Monnot et al. 2013) or vanpooling (Kaan and Olinick 2013), our framework is designed to integrate these approaches as opportunistic ad hoc solvers.

- **Energy Optimization [D2].** Looking at the overall transportation system of all travelers and assuming we can control some of them, we focus on reducing the energy consumption of that whole system. While the energy consumption of transportation networks has been analyzed in isolation for some of the networks (Muñoz and Laval 2006), to our knowledge, we are the first to collaboratively optimize energy consumption in the context of a large multi-modal network, where the trade-offs in energy consumption between different transportation decisions can be precisely evaluated.

COPTER's planning component first creates individual plans and then incorporates anytime search over possible collaboration options provided by ad hoc solvers. Once the collaboration opportunities are depleted, or we run out of time, we select the combination of options that minimizes energy across the system. The following sections establish the formal description of the problem.

Representation

We use the standard transportation network representation as a directed graph $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of directed edges. We denote the set of all modalities as $\sum$ and we define a labeling function $lbl : E \rightarrow \sum$ that associates a mode with every edge of a graph. We assume a representation of time T and a set of users U.

While some of the edges in the graph are time-dependent (buses, trains), others are not dependent (e.g., walking, biking). We capture the difference in a function $start : U \times E \times T \rightarrow T$, which for a particular time identifies the next time when we can start the traversing of an edge by certain user. For a time-independent edge the function $start$ provides the same time it is given. For a time-dependent edge, it provides the next earliest time in the schedule associated with the edge.

For convenience we define three more functions $dur : U \times E \times T \rightarrow T$, which for a given time identifies the duration of traversing an edge by a user. $ene : U \times E \times T \rightarrow R$, which for a given time identifies the expected energy in kWh for traversing an edge by a user. $cst : U \times E \times T \rightarrow R$, which for a given time gives the user specific cost of traversing an edge. ene, cst and dur functions are similar in capturing the conditions of a particular edge of the network at particular time (e.g., congestion, weather conditions). For convenience, we encapsulate the 5-tuple of functions (lbl, $start$, dur, ene, cst) as Υ.

Our construction of the multi-modal transportation graph is similar to previous work (Bast et al. 2016), we attach the modality labels only to the edges - considering any node we enter to have exactly the modality corresponding to the edge we used to enter the node. We further capture the time dependency within the $start$ function, which uses schedules kept at the time dependent edges.

Multi-Modal Planning Problem

People have various constraints on how, when, and for how much money they would like to accomplish their travels. To achieve the "be personal" requirement, each user has a set of hard constraints that must be satisfied and the soft constraints that impact the cost calculations on each edge. The hard constraints consist of a temporal constraint, defining the acceptable temporal window of the trip, and a path-restriction constraint, where we model the resources available to the user (e.g., has bike, car, electric skate-board, ...), including the capabilities of particular transportation services (e.g., bike fits in the train, but not in the bus). The mode sequence constraints are represented using a language that accept only trips that conform with the resources and capabilities. For a user u we denote the language as $L(u)$, an example language for a traveler who either uses a bike or a car and walks can be $w^*(d^+|b^+)w^*$.

Formally, we have described the multi-modal planning graph and we further call a sequence of connected edges in the graph to be a *path* in the graph. Using the edge labeling function we can project the sequence of edges into a sequence of labels forming a word w from the alphabet $\sum$ and we associate a language L with each user in U. Intuitively, we are only interested in plans represented by word w for a user u if and only if w is recognized by language $L(u)$. For example, if we imagine three modalities $\sum = \{w = walk, d = drive, b = bike\}$, then a language $L(u) = w^*(d^+|b^+)w^*$ would describe the personal transportation constraint when the user u can walk for any amount of time (edges), then either take a bike or take a car and ride or drive for any amount of time, and finally walk for any amount of time to achieve its destination. We are supporting only the regular languages and given Kleen's theorem (Kleene and Beeson 2009) we know that we can construct a finite state machine (FSM) for any regular language.

Having a multi-modal graph $G = (V, E)$, we represent a demand for transporting user $u \in U$ from his original location $v \in V$ starting at a certain time $t_s \in T$ to his destination $w \in V$ before a certain deadline $t_e \in T$. We call the 6-tuple $r = (L(u), u, v, w, t_s, t_e)$ a planning request and we define the Multi-modal Planning Problem (MPP) as (G, Υ, r), where G is a graph, Υ is a set of functions and r is a planning request.

We say that a plan $\pi = ((e_1, t_1), ..., (e_n, t_n))$ is a solution of the problem $(G, \Upsilon, (L(u), u, v, w, t_s, t_e))$ if and only if the following holds:

- All the temporal constraints are satisfied. $t_s \leq t_1 + dur(u, e_1, start(u, e_1, t_s)) \leq t_2 + dur(u, e_2, start(u, e_2, t_2)) \leq ... \leq t_n + dur(u, e_n, start(u, e_n, t_n)) \leq t_e$
- The edges connect the origin with the destination. $e_1 = (v_0, v_1), e_2 = (v_1, v_2), ..., e_n = (v_{n-1}, v_n) \wedge (v_0 = v) \wedge (v_n = w)$
- the word wo created by a concatenation of label of the edges in the plan π is accepted by the language $L(u)$. $wo = lbl(e_1) \cdot lbl(e_2) \cdot ... \cdot lbl(e_n) \wedge wo \in L(u)$

Given a problem (G, Υ, r) and a plan π we define several features we are going to evaluate:

- $ene_m(\pi) = \sum_{lbl(e)=m} ene(u, e, t)$, total energy spent in modality m by user u[4]
- $cst_m(\pi) = \sum_{lbl(e)=m} cst(u, e, t)$, total monetary cost spent in modality m by user u
- $dur_m(\pi) = \sum_{lbl(e)=m} dur(u, e, t)$, total time spent in modality m by user u

We denote a vector of all the trip features as $\vec{f}(\pi) = (ene_{walk}(\pi), .., ene_{drive}(\pi), cst_{walk}(\pi), .., cst_{drive}(\pi), dur_{walk}(\pi), .., dur_{drive}(\pi))$. For a problem (G, Υ, r) and a solution π and a vector of weights $\vec{\theta}$, we define the cost of solution as $cost(\pi) = \vec{\theta} \cdot \vec{f}(\pi)$, in other words, the cost of the plan π is a sum of its weighted features. Further, we denote optimal plan as $\pi^* = \arg\min_{\pi \in \Pi} cost(\pi)$. Finding an optimal plan π^* is a shortest-path problem.

A "shortest-path problem" in a graph is standard problem in computer science, solved by a number of well-known algorithms. We use the A* algorithm upon the transportation graph, where the costs of all edges are computed during search. As search nodes are expanded, COPTER ensures that the partial path meets the mode constraints guaranteeing that the final plan is accepted by the language $L(u)$. Similar approach of encoding the path restrictions is known as User-Constrained Multi-Modal Route Planning (Dibbelt, Pajor, and Wagner 2015). Our implementation is complete, using several standard heuristics for lower/upper bounding of the search space. We can imagine that the vector of weights $\vec{\theta}$ represents a simplified personal preferential model of the user of the transportation system. Using linear weighting of the modal edges in the transportation network is a robust and computationally efficient relaxation, although the personalized decision making in transportation is a multi-criteria optimization problem, which is a subject of study in the future work.

Collaborative Planning Opportunities

While MPP considers the users independently, collaborative planning looks at several users at once and optimizes the cost of achieving all of their goals with a shared pool of resources. COPTER supports combining multiple collaborative planning algorithms. Common collaborative examples are (1) carpooling, where a driver diverges from her path to pick-up and drop-off multiple passengers, (2) vanpooling, where a dedicated driver picks-up larger number of passengers and distributes them to their destinations, and (3) pop-up commuter buses that pickup and drop off employees at central locations following dynamic schedules.

We consider collaborative planning to be an opportunistic extension of MPP. Having a selection of solvers that can generate collaborative plans for subsets of users, we can evaluate those cooperative plans and compare them not only against the MPP solutions but also against those produced by different solvers. The opportunism of our approach controls the unavoidable combinatorial explosion of choosing subsets of collaborating users from a very large set (hundreds of thousands of requests). The typical selection process includes spatiotemporal clustering (e.g., grouping users who live and work near each other with similar schedules).

Let us define a solver $S : R^n \rightarrow \Pi^n$, that takes a set of requests and generate a set of plans that may include some collaborative component (i.e. some part of the trip is shared by multiple users). Given a set of requests R, graph G, a set of functions Υ and a set of solvers $S_1, ..., S_n$ the approach for integration of multiple solvers can be described in following steps:

1. $P \leftarrow \emptyset$, we start from an empty set of plans.
2. $\forall S_i \in S_1, .., S_n : generate(R, G, \Upsilon, S_i, P)$, we generate a set of plans using all available solvers - those are the collaborative solvers such as carpooling and vanpooling collaborative solvers and also the multi-modal shortest paths solver of MPP for individual users.
3. $P^* = \arg\min_{p \subseteq P} cost(\cup_{x \in p} x)$ such that $| \cup_{x \in p} x| = |R|$. We choose the combination of plans with minimal total cost.

We start from an empty set of plans, then we independently run all the solvers in parallel until they finish generating new options or a predefined deadline. Finally, we take all the options and select a combination of them P^* with the minimal cost, such that each user is represented exactly once; this is also known as the minimum exact cover problem (Cover and Thomas 1991). Figure 2 illustrates how the transportation options are evaluated against each other. The combinatorial explosion of choosing subsets of collaborating users does not allow an exhaustive generation for real-world scale. As a result, we cannot ensure the optimality of chosen plans.

Currently, the plans generated by solvers at step 2 are not taking into account congestion impacts of plans from different solvers - i.e. n-th MPP plan is taking into consideretion previous $n-1$ plans, however, carpooling solver is not taking into consideration congestion contribution of MPP plans.

Evaluation

We are evaluating the potential impact of COPTER in simulation and through an upcoming deployment.

Simulation Study

By combining COPTER with a state-of-the-art microsimulation model (Elbery et al. 2017), we are able to measure the potential impact of COPTER technology. Our study area (shown in Figure 3) covers the Los Angeles region from 7am until 12pm on a typical weekday. We expect COPTER will reduce the fuel consumption across the study area. The microsimulation model represents all 2.9 million vehicles that travel across the region's arterials and freeways during the time period with a departure time and origin and destination. This simulation includes the 940,641 passengers traveling on public transit and supports walking and biking.

Recall COPTER's expected use case is in influencing a subset of the population to make transportation decisions

[4]Our train energy model computes energy based on the dynamics of the train along with its total weight. Thus, additional passengers increase the energy consumption (Wang and Rakha 2017).

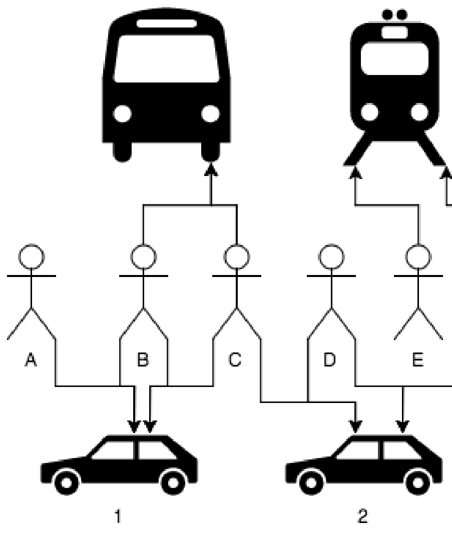


Figure 2: Collaborative planning example combines plans from different solvers. B can either take A or C as a passenger, or use a vanpool together with C. Similarly, D can take C or F as passengers. To transport everyone to their destinations, the only solution is B driving A, D driving C and both E and F taking public transport.

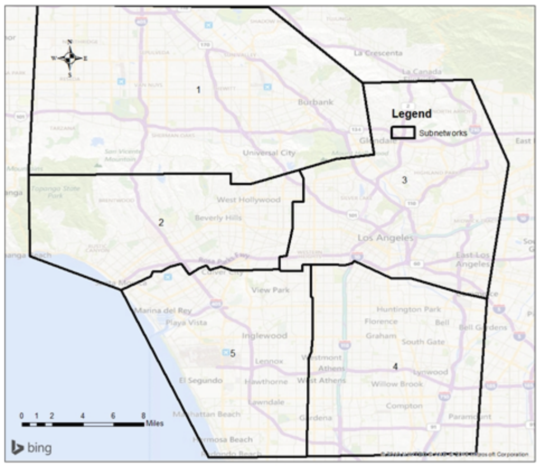


Figure 3: Los Angeles simulation study area

Table 1: Potential improvements in efficiency of the Los Angeles transportation network using COPTER with 10% participation and only 4% accepting alternatives.

	Baseline	Control	Change
Fuel (L)	3,682,998	3,536,777	-4%
Power (kWh)	94,175.7	94,224.9	0.05%
Delay (min/trip)	9.7	8.5	-9%

that improve the efficiency of the overall network. Therefore, we randomly selected 10% of the driving population as our controlled traveler group. For each controlled traveler, we create the temporal constraints for the trip as follows: t_s equal to the trip's departure time and t_e equal t_s plus 1.3 multiplied by the duration of the trip. The simulation runs in close to real time. Controlled traveler demand requests appear in the system 30 minutes before the earliest departure time. COPTER's planner constructs collaborative plans for each controlled traveler. For this study, we assume a cost model that weights the time in each mode equally and incorporates the energy costs associated with travel. At five minutes before the earliest departure, we commit the traveler to the plan. We compare the energy consumption and time lost due to traffic across the entire network against a baseline in which all of the controlled travelers drive alone.

Table 1 summarizes the potential impacts. COPTER reduced fuel consumption by 4% with a minor increase in electrical energy consumption due to extra rail passengers. In addition to the energy impacts, COPTER alleviates congestion with the average delay per auto trip drops from 9.7 minutes to 8.5 minutes in the controlled condition. This translates into saving almost 7 years across all of the travelers. Looking at the differences between the controlled routes and the original routes, only 4% of travelers altered behavior with the overwhelming majority of them (98%) forming into two person carpools. The remainder changed modes to transit, walking, biking, or multi-modal trips.

Future Deployment

While the simulation study illustrates the potential impact of COPTER technology, it is necessary to refine the results by measuring the likelihood of user acceptance of the suggested routes and gather data about the users' cost function. Existing research focuses on either the amount of incentive required to get people to change behavior (Zhao, Xiong, and Zhang 2018) or how to accomplish long-term behavior change (Castellani et al. 2016). To understand if information alone can affect behavior in single interactions, we have a planned pilot deployment of the COPTER technology using the TripGo mobile application. In this pilot, we will explore two hypotheses: (1) personalized cost based recommendations are more likely to be accepted than random alternatives or simple time-based alternatives, and (2) messaging that matches people's susceptibility to persuasion increases the likelihood of message acceptance. The pilot will last two weeks in April 2018 with route recommendations appearing in the TripGo app.

Discussion

COPTER presents a new application of AI for social good focused on reducing congestion and improving the efficiency of transportation networks. We surveyed people's transportation decision-making to identify that users would benefit from personal, helpful, informative, multi-modal recommendations. Therefore, we developed COPTER to provide real-time personal transportation recommendations taking into account unique resources and costs for users. Our simulation study demonstrates the substantial impact this technology could have in improving transportation and reducing emissions at a regional level. Our future work includes a pilot deployment and incorporating learning to improve the user model through user interactions.

Acknowledgements

This work was funded by the Advanced Research Projects Agency-Energy (ARPA-e) under award number DE-AR0000612.

References

Bast, H.; Delling, D.; Goldberg, A.; Müller-Hannemann, M.; Pajor, T.; Sanders, P.; Wagner, D.; and Werneck, R. F. 2016. Route planning in transportation networks. In *Algorithm Engineering*. Springer International Publishing. 19–80.

Bit-Monnot, A.; Artigues, C.; Huguet, M.-J.; and Killijian, M.-O. 2013. Carpooling: the 2 synchronization points shortest paths problem. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, 12–p.

Castellani, S.; Colombino, T.; Grasso, A.; and Mazzega, M. 2016. Understanding commuting to accompany work organisations' and employees' behaviour change. In *Smart Cities Conference (ISC2), 2016 IEEE International*, 1–6. IEEE.

Cialdini, R. B. 2001. Harnessing the science of persuasion. *Harvard Business Review* 79(9):72–81.

Cover, T. M., and Thomas, J. A. 1991. *Elements of Information Theory*. New York, NY, USA: Wiley-Interscience.

Dean, B. C. 2004. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks* 44(1):41–46.

Dibbelt, J.; Pajor, T.; and Wagner, D. 2015. User-constrained multimodal route planning. *J. Exp. Algorithmics* 19:3.2:1–3.2:19.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Elbery, A.; Du, J.; Rakha, H.; Dvorak, F.; and Klenk, M. 2017. Large-scale agent-based multi-modal modeling of transportation networks: System model and preliminary results. In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems*.

Gosling, S. D.; Rentfrow, P. J.; and Swann, W. B. 2003. A very brief measure of the big-five personality domains. *Journal of Research in personality* 37(6):504–528.

Johansson, M. V.; Heldt, T.; and Johansson, P. 2006. The effects of attitudes and personality traits on mode choice. *Transportation Research Part A: Policy and Practice* 40(6):507–525.

Kaan, L., and Olinick, E. V. 2013. The vanpool assignment problem: Optimization models and solution algorithms. *Comput. Ind. Eng.* 66(1):24–40.

Kleene, S., and Beeson, M. 2009. *Introduction to Metamathematics*. Ishi Press International.

Kosinski, M.; Stillwell, D.; and Graepel, T. 2013. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences* 110(15):5802–5805.

Liu, X.; Fritz, C.; and Klenk, M. 2018. On extensibility and personalizability of multi-modal trip planning. In *Proceedings of the 11th Multidisciplinary Workshop on Advances in PReference Handling*.

Muñoz, J. C., and Laval, J. A. 2006. System optimum dynamic traffic assignment graphical solution method for a congested freeway and one destination. *Transportation Research Part B: Methodological* 40(1):1–15.

Schrank, D.; Eisele, B.; Lomax, T.; and Bak, J. 2015. 2015 urban mobility scorecard. *Annual Urban Mobility Scorecard*.

Wang, J., and Rakha, H. A. 2017. Electric train energy consumption modeling. *Applied Energy* 193:346–355.

Zhao, J.; Xiong, C.; and Zhang, L. 2018. A joint-revealed and stated preference analysis of travel behavioral responses to monetary incentives. Technical report.

Solving Concurrent Multiagent Planning using Classical Planning

Daniel Furelos-Blanco and **Anders Jonsson**
Department of Information and Communication Technologies
Universitat Pompeu Fabra
Roc Boronat 138, 08018 Barcelona, Spain
{daniel.furelos, anders.jonsson}@upf.edu

Abstract

In this work we present a novel approach to solving concurrent multiagent planning problems in which several agents act in parallel. Our approach relies on a compilation from concurrent multiagent planning to classical planning, allowing us to use an off-the-shelf classical planner to solve the original multiagent problem. The solution can be directly interpreted as a concurrent plan that satisfies a given set of concurrency constraints, while avoiding the exponential blowup associated with concurrent actions. Theoretically, we show that the compilation is sound and complete. Empirically, we show that our compilation can solve challenging multiagent planning problems that require concurrent actions.

Introduction

Research in multiagent planning has seen a lot of progress in recent years, in part due to the first competition of distributed and multiagent planners, CoDMAP-15 (Komenda, Stolba, and Kovacs 2016). Many recent multiagent planners are based on the MA-STRIPS formalism (Brafman and Domshlak 2008), and can be loosely classified into one of two categories: centralized, in which agents have full information and share the goal, and distributed, in which agents have partial information and individual goals. In CoDMAP-15, the most successful centralized planners were ADP (Crosby, Rovatsos, and Petrick 2013), MAP-LAPKT (Muise, Lipovetzky, and Ramirez 2015) and CMAP (Borrajo 2013), while prominent distributed planners included PSM (Tozicka, Jakubuv, and Komenda 2014), MAPlan (Stolba, Fiser, and Komenda 2016) and MH-FMAP (Torreño, Onaindia, and Sapena 2014).

Although establishing a common set of multiagent benchmark domains was a major step forward, the domains from the centralized track of CoDMAP-15 can all be solved using *sequential* plans in which one agent acts at a time. In contrast, there are many applications that require agents to act in *parallel*; examples include shared journey planning, robot coordination tasks such as RoboCupSoccer (Nardi et al. 2014) and RoboCupRescue (Sheh, Schwertfeger, and Visser 2016), and real-time strategy games, to name a few.

In this paper we consider the problem of *concurrent* centralized multi-agent planning in which agents can act in parallel at each time step. This problem is challenging for different reasons: the number of concurrent actions is worst-case exponential in the number of agents, and restrictions are needed to ensure that concurrent actions are well-formed. Usually, these restrictions take the form of *concurrency constraints* (Boutilier and Brafman 2001; Crosby 2013), which can model both the case when two actions *must* occur in parallel, and the case when two actions *cannot* occur in parallel.

Although some planners from CoDMAP-15 (CMAP, MAPlan and MH-FMAP) can produce concurrent plans, there are few that can reliably handle more complex concurrency constraints. A notable exception is the work of Crosby, Jonsson, and Rovatsos (2014), who associate concurrency constraints with the *objects* of a multiagent planning problem and transform the problem into a sequential, single-agent problem that can be solved using a classical planner.

Brafman and Zoran (2014) extended the distributed forward-search planner MAFS (Nissim and Brafman 2014) to support concurrency constraints while preserving privacy. In MAFS, each agent maintains its own search space, and has a queue for expanded states (closed list) and another for states to be expanded (open list). When a state s is expanded, the agent uses its own operators only; thus, two agents expanding the same state will generate different successors. Messages are exchanged between agents in order to inform each other about the expansion of relevant states. Consequently, agents explore the search space together while preserving privacy. Maliah, Brafman, and Shani (2017) proposed MAFBS, which extended MAFS to use forward and backward messages. This new approach reduced the number of messages required and also resulted in an increase in the privacy of agents.

In this paper we describe a planner that can handle arbitrary concurrency constraints, as originally proposed by Boutilier and Brafman (2001). Our approach is similar to that of Crosby, Jonsson, and Rovatsos (2014) in that we transform a multiagent planning problem into a single-agent problem that is significantly easier to solve, while avoiding the exponential blowup associated with concurrent actions. However, the concurrency constraints of Boutilier and Brafman are significantly more expressive than those of Crosby, Jonsson, and Rovatsos, enabling us to solve multiagent problems with more complex interactions (e.g. effects that depend on the concurrent actions of other agents). We show that our planner is sound and complete, and perform experiments in several concurrent multiagent planning domains to evaluate its performance.

The remainder of this paper is structured as follows. We first introduce the different planning formalisms that are needed to describe our planner. Next, we describe the compilation from multiagent planning to single-agent planning that our planner employs. We then present the results of experiments with our planner in several domains that require agents to act in parallel. Finally, we relate our planner to existing work in the literature, and conclude with a discussion.

Background

In this section we describe the planning formalisms that we use in the paper: classical planning, concurrent planning, and concurrent multiagent planning.

Classical Planning

We consider the fragment of classical planning with conditional effects and negative conditions and goals. Given a set of fluents F, a *literal* l is a valuation of a fluent in F, where $l = f$ denotes that l assigns true to $f \in F$, and $l = \neg f$ that l assigns false to f. A set of literals L is *well-defined* if it does not assign conflicting values to any fluent, i.e. if L does not contain both f and $\neg f$ for some fluent $f \in F$. Let $\mathcal{L}(F)$ be the set of well-defined literal sets on F, i.e. the set of all partial assignments of values to fluents. Given a literal set $L \in \mathcal{L}(F)$, let $\neg L = \{\neg l : l \in L\}$ be the *complement* of L. We also define the *projection* $L_{|X}$ of a literal set L onto a subset of fluents $X \subseteq F$.

A *state* $s \in \mathcal{L}(F)$ is a well-defined literal set such that $|s| = |F|$, i.e. a total assignment of values to fluents. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions, but we often abuse notation by defining a state s only in terms of the fluents that are true in s, as is common in classical planning.

A classical planning problem is a tuple $\Pi = \langle F, A, I, G\rangle$, where F is a set of fluents, A is a set of actions, $I \in \mathcal{L}(F)$ is an initial state, and $G \in \mathcal{L}(F)$ is a goal condition (usually satisfied by multiple states). Each action $a \in A$ has a precondition $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of conditional effects $\mathsf{cond}(a)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a)$ is composed of two literal sets $C \in \mathcal{L}(F)$ (the condition) and $E \in \mathcal{L}(F)$ (the effect).

An action $a \in A$ is applicable in state s if and only if $\mathsf{pre}(a) \subseteq s$, and the resulting (triggered) *effect* is given by

$$\mathsf{eff}(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a), C \subseteq s} E,$$

i.e. effects whose conditions hold in s. We assume that $\mathsf{eff}(s, a)$ is a well-defined literal set in $\mathcal{L}(F)$ for each state-action pair (s, a). The result of applying a in s is a new state $\theta(s, a) = (s \setminus \neg\mathsf{eff}(s, a)) \cup \mathsf{eff}(s, a)$. It is straightforward to show that if s and $\mathsf{eff}(s, a)$ are in $\mathcal{L}(F)$, then so is $\theta(s, a)$.

Given a planning problem Π, a *plan* is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ that induces a state sequence $\langle s_0, s_1, \ldots, s_n\rangle$ such that $s_0 = I$ and, for each i such that $1 \leq i \leq n$, action a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. The plan π *solves* Π if and only if $G \subseteq s_n$, i.e. if the goal condition is satisfied following the application of π in I.

Concurrent Planning

Concurrent planning is the extension of classical planning that allows actions to be applied in parallel, forming *concurrent* or *joint* actions. Given a classical planning problem $\Pi = \langle F, A, I, G\rangle$, an unconstrained concurrent planning problem is given by $\Pi_{conc} = \langle F, 2^A, I, G\rangle$, where the action set 2^A is given by the power set of the original action set A. The aim is to find a *concurrent plan* π_{conc} that solves Π_{conc} by applying a sequence of concurrent actions in 2^A.

To ensure that joint actions have well-defined effects, researchers often impose *concurrency constraints* that model whether two atomic actions *must* or *cannot* be done concurrently. For example, in PDDL 2.1 (Fox and Long 2003), two actions a^1 and a^2 cannot be applied concurrently if a^1 has an effect on a fluent f and a^2 has a precondition or effect on f. Crosby (2013) defines concurrency constraints on actions that have the same *object* in their preconditions or effects.

We adopt a formulation of concurrency constraints due to Boutilier and Brafman (2001), later extended by Kovacs (2012). The idea is to extend preconditions and conditional effects with *actions* in addition to fluents. We overload notation and let each $a \in A$ denote a propositional variable. If a^1 has a precondition a^2, then a^1 is only applicable if a^2 is concurrent with a^1. If a^1 has a precondition $\neg a^2$, then a^1 is only applicable if a^2 is *not* concurrent with a^1.

We extend the notation for classical planning as follows. By viewing A as a set of propositional variables, we can model a subset of atomic actions in 2^A as a well-defined literal set $b \in \mathcal{L}(A)$ such that $|b| = |A|$, analogous to how states are formed from fluents. The subset includes those actions that appear as positive literals in b, while negative literals denote those actions that are *not* part of the subset.

For a given atomic action $a \in A$, the precondition $\mathsf{pre}(a) \in \mathcal{L}(F \cup A)$ and any condition $C \in \mathcal{L}(F \cup A)$ of a conditional effect $C \rhd E \in \mathsf{cond}(a)$ are extended to include actions in addition to fluents. Each effect $E \in \mathcal{L}(F)$ of a conditional effect $C \rhd E \in \mathsf{cond}(a)$ is exclusively on fluents as before. Given a state s and a set of actions that are concurrent with a, represented by a literal set $b \in \mathcal{L}(A)$, a is applicable if and only if $\mathsf{pre}(a) \subseteq s \cup b$, and the resulting effect is given by

$$\mathsf{eff}(s \cup b, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a), C \subseteq s \cup b} E,$$

i.e. effects whose conditions hold in $s \cup b$.

We can now define the semantics of a joint action $b \in 2^A$. Concretely, b satisfies the concurrency constraints if and only if $\mathsf{pre}(a)_{|A} \subseteq b \setminus \{a\}$ for each $a \in b$. If b is applicable, its precondition and effect are defined as the union of the preconditions and effects of the constituent atomic actions:

$$\mathsf{pre}(b) = \bigcup_{a \in b} \mathsf{pre}(a)_{|F},$$

$$\mathsf{eff}(s, b) = \bigcup_{a \in b} \mathsf{eff}(s \cup b \setminus \{a\}, a), \quad \forall s.$$

As before, we assume that $\mathsf{eff}(s, b)$ is a well-defined literal set in $\mathcal{L}(F)$ for each pair of state s and applicable joint action b.

Concurrent Multiagent Planning

In concurrent multiagent planning, each atomic action belongs to an *agent*. We consider the problem of *centralized* multiagent planning in which agents share the goal.

A *concurrent multiagent planning problem* (CMAP) is a tuple $\Pi = \langle N, F, \{A^i\}_{i=1}^n, I, G\rangle$, where $N = \{1, \dots, n\}$ is a set of agents and A^i is the set of atomic actions of agent $i \in N$. This is identical to the standard definition of multiagent planning problems (Brafman and Domshlak 2008), with the only difference being that the actions include concurrency constraints as defined in the previous section. The fluent set F, initial state I and goal condition G are defined as before.

A CMAP implicitly defines a negative concurrency constraint on each pair of atomic actions $(a^1, a^2) \subseteq A^i$ that belong to the same agent i. Consequently, each agent can contribute at most one atomic action to each joint action. These concurrency constraints are not included in action definitions.

To illustrate CMAPs, we use the TABLEMOVER domain (Boutilier and Brafman 2001), in which two agents move blocks between rooms. There are two possible strategies:

1. Pick up blocks and carry them using their arms.
2. Put blocks on a table, carry the table together to another room, and tip the table to make the blocks fall down.

Figure 1 shows the definition of the lift-side action in the notation of Kovacs (2012), which is used by agent ?a to lift side ?s of the table. The precondition is that the side must be down (i.e. on the floor) and the agent cannot be holding anything. Moreover, the precondition also states that no other agent ?a2 can lower side ?s2 at the same time. When the action is applied, ?s is no longer down but up, and ?a is busy lifting ?s. The action also has a conditional effect (represented by the when clause): if some side ?s2 is not lifted by any agent ?a2, then all blocks on the table fall to the floor.

Note that the action lift-side is defined using forall quantifiers. In practice, such quantifiers are compiled away, such that the resulting actions have quantifier-free preconditions and effects, as in our definition of actions.

```
(:action lift-side
 :agent ?a - agent
 :parameters (?s - side)
 :precondition
  (and (at-side ?a ?s)
       (down ?s) (handempty ?a)
       (forall (?a2 - agent ?s2 - side)
        (not (lower-side ?a2 ?s2))))
 :effect
  (and (not (down ?s)) (lifting ?a ?s)
       (up ?s) (not (handempty ?a ?s))
       (forall
        (?b - block ?r - room ?s2 - side)
        (when
          (and (inroom Table ?r)
               (on-table ?b) (down ?s2)
               (forall (?a2 - agent)
                    (not (lift-side ?a2 ?s2))))
          (and (on-floor ?b) (inroom ?b ?r)
               (not (on-table ?b)))))))
```

Figure 1: Definition of the TABLEMOVER action lift-side using the notation of Kovacs (2012) (concurrency constraints in bold).

Compilations for CMAPs

Let $\Pi = \langle N, F, \{A^i\}_{i=1}^n, I, G\rangle$ be a CMAP, and let $A = A^1 \cup \cdots \cup A^n$ be the set of atomic actions of Π. A straightforward approach to solving Π is to define a concurrent planning problem $\Pi_{conc} = \langle F, B, I, G\rangle$ and apply a classical planner to solve Π_{conc}. If $B \subseteq 2^A$ is exactly the subset of joint actions that satisfy the concurrency constraints of the actions in Π, this approach is both sound and complete.

However, even though the joint action set B might be much smaller than 2^A, it is still worst-case exponential in the number of agents. Most classical planners ground the actions, and if B is too large, they often do not make it past preprocessing. Moreover, to generate the set B we would typically have to iterate over *all* joint actions, and test whether each action satisfies the concurrency constraints of atomic actions.

Here we describe an alternative approach to solving a CMAP Π. The idea is to model each joint action $b = \{a^1, \dots, a^k\}$ using multiple atomic actions: one set of actions for *selecting* $a^1, \dots, a^k$, one set of actions for *applying* $a^1, \dots, a^k$, and one set of actions for *resetting* $a^1, \dots, a^k$. The result is a classical planning problem $\Pi' = \langle F', A', I', G'\rangle$ such that the size of the action set A' is *linear* in $|A|$, the number of atomic actions of agents.

Simulating a joint action b using a sequence of atomic actions $\langle a^1, \dots, a^k\rangle$ is problematic for the following reason: when applying an atomic action a^i, we may not yet know which atomic actions will be applied by other agents. Since those other actions may be part of the precondition and conditional effects of a^i, it becomes difficult to ensure that the concurrency constraints of a^i are correctly enforced.

Our approach is to divide the simulation of a joint action b into three phases: selection, application, and reset. In the selection phase, we use an auxiliary fluent active-a^i to model that the atomic action a^i has been selected. In the application phase, since the selection of atomic actions is known, we can substitute each action a^i in preconditions and conditional effects with the auxiliary fluent active-a^i. In the reset phase, various auxiliary fluents are reset.

Note that this compilation takes into account the multiagent nature of the problem. Each agent can apply at most one atomic action per time step, and agents collaborate to form joint actions whose constituent atomic actions are compatible and/or inapplicable on their own.

We proceed to define the components of the compilation.

Fluents

We describe the fluents in PDDL format, i.e. each fluent is instantiated by assigning objects to predicates.

The set of fluents $F' \supseteq F$ includes all original fluents in F, plus the following auxiliary fluents:

- Fluents free, select, apply and reset modeling the phase.
- For each agent i, fluents free-agent(i), busy-agent(i) and done-agent(i) that model the agent state: free to select an action, selected an action, and applied the action.
- For each action $a^i \in A^i$ in the action set of agent i, a fluent active-a^i which models that a^i has been selected. We use F_{act} to denote the subset of fluents of this type.

By simple inspection, the total number of fluents in F' is given by $|F'| = |F| + 4 + 3n + \sum_{i \in N} |A^i| = O(|F| + |A|)$.

The initial state I' of the compilation Π' is given by

$$I' = I \cup \{\text{free}\} \cup \{\text{free-agent}(i) : i \in N\},$$

i.e. the initial state on fluents in F is I, we are not simulating any joint action, and all agents are free to select actions. The goal condition is given by $G' = G \cup \{\text{free}\}$, i.e. the goal condition G has to hold at the end of a joint action simulation.

Actions

For a literal set $L \in \mathcal{L}(F \cup A)$, let $L_{|A}/F_{act}$ denote the projection of L onto A, followed by a substitution of the actions in A with the corresponding fluents in F_{act}. Note that both $L_{|F}$ and $L_{|A}/F_{act}$ are literal sets on fluents in F', i.e. the dependence on actions in A is removed.

The first four actions in the set A' allow us to switch between simulation phases, and are defined as follows:

$$\begin{aligned}
\text{select-phase:} \quad & \text{pre} = \{\text{free}\}, \\
& \text{cond} = \{\emptyset \rhd \{\neg\text{free}, \text{select}\}\}. \\
\text{apply-phase:} \quad & \text{pre} = \{\text{select}\}, \\
& \text{cond} = \{\emptyset \rhd \{\neg\text{select}, \text{apply}\}\}. \\
\text{reset-phase:} \quad & \text{pre} = \{\text{apply}\}, \\
& \text{cond} = \{\emptyset \rhd \{\neg\text{apply}, \text{reset}\}\}. \\
\text{finish:} \quad & \text{pre} = \{\text{reset}, \text{free-agent}(i) : i \in N\}, \\
& \text{cond} = \{\emptyset \rhd \{\neg\text{reset}, \text{free}\}\}.
\end{aligned}$$

For each action $a^i \in A^i$ in the action set of agent i, we define three new actions in A': select-a^i, do-a^i and end-a^i. These actions represent the three steps that an agent must perform during the simulation of a joint action.

The action select-a^i causes i to select action a^i during the selection phase, and is defined as follows:

$$\begin{aligned}
\text{pre} &= \{\text{select}, \text{free-agent}(i)\} \cup \text{pre}(a^i)_{|F}, \\
\text{cond} &= \{\emptyset \rhd \{\text{busy-agent}(i), \neg\text{free-agent}(i), \text{active-}a^i\}\}.
\end{aligned}$$

The precondition ensures that we are in the selection phase, that i is free to select an action, and that the precondition of a^i holds on fluents in F. The effect prevents i from selecting another action, and marks a^i as selected.

The action do-a^i applies the effect of a^i in the application phase, and is defined as follows:

$$\begin{aligned}
\text{pre} &= \{\text{apply}, \text{busy-agent}(i), \text{active-}a^i\} \cup \text{pre}(a^i)_{|A}/F_{act}, \\
\text{cond} &= \{\emptyset \rhd \{\text{done-agent}(i), \neg\text{busy-agent}(i)\}\} \\
&\quad \cup \{C_{|F} \cup C_{|A}/F_{act} \rhd E : C \rhd E \in \text{cond}(a^i)\}.
\end{aligned}$$

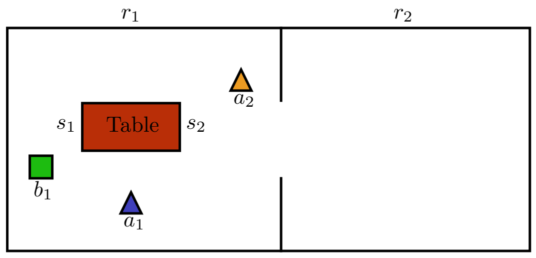


Figure 2: Initial state of a simple TABLEMOVER instance.

The precondition ensures that we are in the application phase, that a^i was previously selected, and that all concurrency constraints in the precondition of a^i hold. The effect is to apply all conditional effects of a^i, where each condition $C_{|F} \cup C_{|A}/F_{act}$ is generated from C by substituting each action $b^j \in A$ with active-b^j. Agent i is also marked as done to prevent a^i from being applied a second time.

The action end-a^i resets auxiliary fluents to their original value, and is defined as follows:

$$\begin{aligned}
\text{pre} &= \{\text{reset}, \text{done-agent}(i), \text{active-}a^i\}, \\
\text{cond} &= \{\emptyset \rhd \{\text{free-agent}(i), \neg\text{done-agent}(i), \neg\text{active-}a^i\}\}.
\end{aligned}$$

The precondition ensures that we are in the reset phase and that a^i was previously selected and applied (due to done-agent(i)). The effect is to make agent i free to select actions again, and to mark a^i as no longer selected.

Again, by inspection we can see that the total number of actions in A' is given by $|A'| = 4 + 3\sum_i |A^i| = O(|A|)$.

Properties

Figure 2 shows an example instance of TABLEMOVER in which the goal is for agents a_1 and a_2 to move block b_1 from room r_1 to room r_2. An example concurrent plan that solves this instance is defined as follows:

```
1 (to-table a1 r1 s2)(pickup-floor a2 b1 r1)
2 (putdown-table a2 b1 r1)
3 (to-table a2 r1 s1)
4 (lift-side a1 s2)(lift-side a2 s1)
5 (move-table a1 r1 r2 s2)(move-table a2 r1 r2 s1)
6 (lower-side a1 s2)
```

In this plan, agent a_2 first puts the block on the table, and then a_1 and a_2 concurrently lift each side of the table and move the table to room r_2. Finally, a_1 lowers its side of the table, causing the table to tip and the block to fall to the floor.

The following sequence of classical actions in A' can be used to simulate the first joint action of the concurrent plan:

```
(select-phase )
(select-to-table a1 r1 s2)
(select-pickup-floor a2 b1 r1)
(apply-phase )
(do-pickup-floor a2 b1 r1)
(do-to-table a1 r1 s2)
(reset-phase )
(end-to-table a1 r1 s2)
(end-pickup-floor a2 b1 r1)
(finish )
```

We show that the compilation is both sound and complete.

Theorem 1 (Soundness)**.** *A classical plan π' that solves Π' can be transformed into a concurrent plan π that solves Π.*

Proof. When fluent free is true, the only applicable action is select-phase. The only way to make free true again is to cycle through the three phases and end with the finish action.

During the selection phase, a subset of actions $a^1, \ldots, a^k$ are selected, causing the corresponding agents to be busy. Because of the precondition free-agent(i) of the finish action, each selected action a^i has to be applied in the application phase, and reset in the reset phase. The resulting simulated joint action is given by $b = \{a^1, \ldots, a^k\}$.

The precondition of b holds since the precondition of each a^i on fluents in F is checked in the selection phase, during which no fluents in F change values. The concurrency constraints of a^i are checked in the application phase when all actions have already been selected. This also ensures that the conditional effects of a^i are correctly applied. Finally, auxiliary fluents are cleaned in the reset phase. Hence the joint action b satisfies all concurrency constraints and is correctly simulated by the corresponding action subsequence of π'.

Let π be the concurrent plan composed of the sequence of joint actions simulated by the plan π'. Since π' solves Π', the goal condition G holds at the end of π', implying that G also holds at the end of π. This implies that π solves Π. □

Theorem 2 (Completeness)**.** *A concurrent plan π that solves Π corresponds to a classical plan π' that solves Π'.*

Proof. Let $b = \{a^1, \ldots, a^k\}$ be a joint action of the concurrent plan π. We can use a sequence of actions in A' to simulate b by selecting, applying and resetting each action among $a^1, \ldots, a^k$. Since b is part of π, its precondition and concurrency constraints have to hold, implying that the precondition and concurrency constraints of each atomic action hold. Hence the action sequence is applicable and results in the same effect as b. By concatenating such action sequences for each joint action of π, we obtain a plan π'. Since π solves Π, the goal condition G holds at the end of π, implying that G holds at the end of π'. This implies that π' solves Π'. □

Extensions

The basic compilation checks concurrency constraints in the application phase. Here we describe an extension that checks negative concurrency constraints in the selection phase, allowing a classical planner to identify inadmissible joint actions as early as possible, reducing the branching factor.

Assume that action a^i has a negative concurrency constraint $\neg a^j$. As before, we can simulate this constraint using the fluent $\neg$active-a^j. However, a^j may be selected *after* a^i in the selection phase, in which case $\neg$active-a^j holds when selecting a^i. To prevent inadmissible joint actions from being selected, we introduce additional fluents in the set F':

- For each action $a^i \in A^i$ in the action set of agent i, a fluent req-neg-a^i which indicates that a^i cannot be selected.

We now redefine the action select-a^i as follows:

$$\begin{aligned}
\mathrm{pre} &= \{\mathsf{select}, \mathsf{free\text{-}agent}(i), \neg\mathsf{req\text{-}neg\text{-}}a^i\} \cup \mathsf{pre}(a^i)_{|F} \\
&\cup \{\neg\mathsf{active\text{-}}b^j : \neg b^j \in \mathsf{pre}(a^i)\}, \\
\mathrm{cond} &= \{\emptyset \rhd \{\mathsf{busy\text{-}agent}(i), \neg\mathsf{free\text{-}agent}(i), \mathsf{active\text{-}}a^i\}\} \\
&\cup \{\emptyset \rhd \{\mathsf{req\text{-}neg\text{-}}b^j : \neg b^j \in \mathsf{pre}(a^i)\}\}.
\end{aligned}$$

To select a^i, req-neg-a^i has to be false. For each negative concurrency constraint $\neg b^j$ of a^i, action select-a^i adds fluent req-neg-b^j, preventing b^j from being selected after a^i.

With this extension, we only need to check *positive* concurrency constraints (i.e. required concurrency) in the application phase. We also redefine end-a^i such that fluents of type req-neg-a^i are reset in the cleanup phase, using the opposite effect of select-a^i. The initial state and goal condition do not change since the new fluents are always false while no joint action is simulated.

The second extension is to impose a bound C on the number of atomic actions selected in the selection phase, resulting in a classical planning problem $\Pi'_C = \langle F'_C, A'_C, I'_C, G'_C \rangle$. The fluent set $F'_C \supseteq F'$ extends F' with fluents count(j), $0 \leq j \leq C$. Counter parameters are added to the select and reset actions so that they can respectively increment and decrement the value of the counter. Crucially, no select action is applicable when $j = C$, preventing us from selecting more than C actions. The benefit is to reduce the branching factor by disallowing joint actions with more than C atomic actions.

We leave the following proposition without proof:

Proposition 3. *The compilation Π'_C that includes both proposed extensions is sound.*

Note that the compilation Π'_C is not complete. For instance, consider a concurrent multiagent plan that contains a joint action involving 4 atomic actions. If $C < 4$, then the concurrent multiagent plan cannot be converted into an equivalent classical plan without exceeding the bound C.

Experimental Results

We tested our compilations in two different sets of domains: centralized multiagent domains from the CoDMAP-15 competition, and four domains (MAZE, TABLEMOVER, WORKSHOP and BOXPUSHING) that require concurrency[1].

In each domain, we used three variants of our compilations: unbounded joint action size, and joint action size bounded by $C = 2$ and $C = 4$. In all variants, we used the extension that identifies negative concurrency constraints in the selection phase. The resulting classical planning problems were then solved using Fast Downward (Helmert 2006) in the LAMA setting (Richter and Westphal 2010).

All experiments ran on Intel Xeon E5-2673 v4 @ 2.3GHz processors. They had a time limit of 30 minutes and a memory limit of 8 GB.

[1]The code of the compilation and the domains are available at https://github.com/aig-upf/universal-pddl-parser-multiagent.

CoDMAP Domains

Although the centralized multiagent benchmark domains of CoDMAP involve multiple agents, none of these domains require concurrency between agents, i.e. their instances can be solved by sequences of atomic actions. Instead, the purpose of CoDMAP was to solve problems involving privacy over predicates, constants and objects.

As our approach and CoDMAP planners differ in their purpose, we decided to compare the former against the classical planner it uses to solve the compilation (Fast Downward). Since CoDMAP domains can be sequentially solved, Fast Downward can be directly applied on them. Furthermore, we will be able to see how the results of solving a problem sequentially compare to the results if the same problem is solved when concurrency is allowed. In addition, the coverage of Fast Downward is almost as high as the winner of the centralized track at CoDMAP-15, ADP (Crosby, Rovatsos, and Petrick 2013), which solved 222 instances.

We compare the results of our compilation with those obtained by running Fast Downward (FD) directly on the given instances. We wanted to test the following hypotheses:

1. Using our compilations it is possible to solve approximately the same number of instances as a classical planner that ignores the multiagent nature of the problem.
2. The plans resulting from solving the compiled problems are implicitly more compressed since atomic actions are grouped into joint actions.

The domains forming this set of experiments were manually modified to add the appropriate negative concurrency constraints. Otherwise, the instances would violate our assumption that the triggered effect $\mathsf{eff}(s, b)$ of a joint action is well-defined, producing invalid plans. For instance, an atomic action adding a fluent f and another action deleting f could have been part of the same joint action. Therefore, note that the complexity of the tasks increases with respect to the original tasks that had no concurrency constraints.

Table 1 shows the results for this set of experiments. There are four different metrics:

- Coverage: number of instances solved.
- Time: average number of seconds taken to find a solution.
- Plan length: number of actions in the plan, corresponding to the number of joint actions for our compilations.
- #Actions: total number of actions instantiated by FD.

From the table, we observe that the planner with the largest coverage is FD (219, 91.25%). The compilation with the largest coverage is the variant whose joint action size is bounded by 2 (158, 65.83%), followed by the unbounded variant (143, 59.58%) and the variant whose bound is set to 4 (140, 58.3%). Thus, although all variants solve less instances, they are not far from the results obtained by FD for many domains. As expected, when concurrency is not required to solve a MAP, the auxiliary fluents and additional copies of actions (corresponding to the three phases of a joint action simulation) introduced by our compilations do not pay off in solution efficiency, resulting in a smaller coverage and a larger time to solve the compiled instances.

Regarding plan length, the compilations always result in shorter solutions than FD. Since FD outputs a sequential plan with no joint actions, it has no way of compressing the plans, unlike the compilations in which atomic actions are grouped into joint actions.

As stated in the previous section, the number of actions of the compilation with unbounded joint action size is approximately $3|A|$, and the number of actions of the compilation with joint action size bounded by C is approximately $(C + 1)|A|$, which is reflected quite well in the number of instantiated actions (results vary somewhat because of the reachability tests performed by FD during grounding).

Domains with Required Concurrency

In this section, we first give a brief description of the domains that were included in the experiments, and the other algorithms that were used for comparison.

The MAZE domain (Crosby 2014) consists of a grid of interconnected locations. Each agent in the maze must move from an initial location to a target location. The connection between two adjacent locations can be one of the following:

- Door: can only be traversed by one agent at a time. Some are initially locked, and are unlocked by pushing a specific switch, placed anywhere in the maze.
- Bridge: can be crossed by multiple agents at once, but is destroyed after the first crossing.
- Boat: can only be used by two or more agents in the same direction.

The WORKSHOP domain is a new domain in which the objective is to perform inventory in a high-security storage facility. It has the following characteristics:

- To open a door, one agent has to press a switch while another agent simultaneously turns a key.
- To do inventory on a pallet, one agent has to use a forklift to lift the pallet while another agent examines it (for security reasons, labels are located underneath pallets).
- There are also actions for picking up a key, entering or exiting a forklift, moving an agent, and driving a forklift.

The BOXPUSHING domain (Brafman and Zoran 2014) consists in a grid of interconnected locations. Agents must push the boxes from one location to another. A different number of agents is needed depending on the box size: one for small boxes, two for mediums, and three for large boxes.

The algorithm that we use for comparison is that of Crosby, Jonsson, and Rovatsos (2014) (which we refer to as CJR), who define concurrency constraints in the form of affordances on subsets of objects. For example, the affordance on the subset of objects $\{\text{location}, \text{boat}\}$ in the MAZE domain is $[2, \infty]$, representing that at least two agents have to row the boat between the same two locations at once. Even though their algorithm cannot handle the concurrency constraints of CMAPs, the MAZE and WORKSHOP domains can be reformulated using their concurrency constraints.

The concurrency constraints of CJR are not as expressive as those of Kovacs because:

Domain	N	Coverage				Time (s.)				Plan length				# Actions			
		2	4	∞	FD	2	4	∞	FD	2	4	∞	FD	2	4	∞	FD
BLOCKSWORLD	20	7	2	4	**20**	759.5	-	-	**0.2**	**32.1**	-	-	32.8	6848	12323	4110	**1270**
DEPOT	20	13	10	9	**17**	202.9	246.4	223.9	**58.3**	30.6	15.7	**14.9**	44.0	10100	18176	6061	**2007**
DRIVERLOG	20	18	17	18	**20**	67.3	58.8	73.7	**26.1**	21.1	**20.5**	25.2	35.6	38416	69145	23051	**7386**
ELEVATORS08	20	9	8	10	**20**	13.8	12.5	9.5	**0.2**	31.0	**30.3**	36.4	65.1	10779	19399	6469	**2155**
LOGISTICS00	20	**20**	**20**	**20**	**20**	1.9	2.9	212.1	**0.0**	30.3	**28.0**	30.1	50.2	1781	3202	1070	**318**
ROVERS	20	**20**	**20**	19	**20**	45.2	75.2	20.9	**0.1**	46.5	47.4	**42.9**	56.8	18314	32962	10990	**2609**
SATELLITES	20	19	17	19	**20**	82.8	128.0	32.2	**1.0**	**32.6**	35.5	34.2	55.9	45106	81188	27065	**8122**
SOKOBAN	20	0	0	0	**18**	-	-	-	**32.3**	-	-	-	**54.1**	3319	5970	1993	**663**
TAXI	20	**20**	**20**	**20**	**20**	1.3	2.6	0.7	**0.0**	14.8	**14.7**	**14.7**	18.7	544	975	328	**108**
WIRELESS	20	2	2	2	**4**	-	-	-	-	-	-	-	-	15644	28156	9388	**3128**
WOODWORKING08	20	14	8	4	**20**	290.0	256.0	-	**0.9**	22.4	**11.4**	-	46.1	17406	31327	10445	**3447**
ZENOTRAVEL	20	16	16	18	**20**	87.2	125.6	164.8	**1.5**	**23.6**	24.1	34.2	46.9	67586	121652	40553	**13502**
Total	240	158	140	143	**219**												

Table 1: Summary of results for CoDMAP domains. Planners "2" and "4" are compilations having joint action size bounded by 2 and 4 respectively, while "∞" is the variant with unbounded joint action size. "FD" is Fast Downward directly applied to the given instances (no compilation involved). N is number of instances; time and length are averages over all instances solved, for all planners that solved at least 5 instances. The total number of actions is an average over all instances.

- Actions cannot be used in conditional effects, so their algorithm cannot solve instances of TABLEMOVER.
- To represent concurrency constraints on multiple action templates in PDDL, they have to be defined on the same subset of objects. In contrast, the constraints of Kovacs can be defined on arbitrary pairs of actions.

Furthermore, CJR does not separate the atomic action selection from the atomic action application. This is a big problem since one of the atomic actions can delete the precondition of other atomic actions, thus canceling the formation of the joint action. For example, in the MAZE domain, the action for crossing a bridge requires that the bridge exists, and destroys the bridge as an effect. Therefore, as this approach does not separate the selection from the application, this action can be done just by one agent at a time (and not by infinite agents as the problem states). The same occurs in the BOXPUSHING domain. Instances where a medium or a large box must be moved cannot be solved with this approach because the first agent to "push" the box will move it. Thus, the box location precondition for the other agent(s) does not hold, so the box is not moved in the end.

As for our compilations, we used Fast Downward in the LAMA 2011 setting to solve the instances produced by CJR.

Another algorithm we could have used for comparison is MAFS (Brafman and Zoran 2014). Unlike our approach and Crosby, Jonsson, and Rovatsos's, it is a distributed approach that preserves privacy and that has been shown to work in the BOXPUSHING domain. However, we have not tested it as we have not had access to the code to perform experiments.

Table 2 shows the results for the four domains. To provide an idea of how each planner behaves as a function of the number of agents, the table shows for each domain the same metrics for different numbers of agents. In the case of BOXPUSHING with three agents, as the version with the bound set to 2 cannot solve instances with large boxes, there are separate results for the cases where the number such boxes is 0, and the cases where it is greater than 0.

In terms of coverage, the unbounded compilation (∞) performs the best (93, 69.92%). The variant bounded to 2 and the variant bounded to 4 have similar coverage: 82 (61.65%) and 89 (66.92%) respectively. The former is clearly affected by the fact it cannot solve the instances requiring the concurrent action of 3 agents in the BOXPUSHING domain. Finally, CJR is the approach with the worst coverage (17, 12.78%). It performs reasonably well in MAZE in spite of its limitation regarding the action for crossing a bridge. On the other hand, its results are not very good in WORKSHOP, and it cannot solve instances from the TABLEMOVER and BOXPUSHING domains. Furthermore, the higher the number of agents, the worse the coverage becomes because problems are harder to solve (the number of available actions grows).

Regarding execution time, the unbounded compilation and the compilation bounded to 2 are the fastest. The higher the number of agents, the longer it takes to compute a plan.

In the case of plan length (i.e. number of joint actions), we observe that the plans obtained with our approach are shorter than the ones obtained with CJR. CJR obtains worse results because it explicitly builds joint actions only if their constituent joint actions are associated with a concurrency constraint. Thus, any action that appears out of a joint action, can be considered as a joint action of size 1. In contrast, our approach allows to combine atomic actions arbitrarily, so it already compresses the solution while planning.

In summary, it is not clear which variant is the best since they highly depend on the problem. In general, the version bounded to 2 works well, but it is useless if concurrency is required between 3 or more agents. The version bounded to 4 has a similar coverage to the unbounded one, but mainly because there are not instances requiring concurrency between more than 4 agents. Thus, we believe that the unbounded option is the most convenient since its performance is not far from the best in all domains, and it is more general.

Finally, we have performed preliminary scalability results in the BOXPUSHING domain. In these experiments we checked the results when $n \in \{1, \ldots, 10\}$ agents are required to move a box from a room r_1 to a neighbor room

Domain	N	Coverage				Time (s.)				Plan length				# Actions			
		2	4	∞	CJR	2	4	∞	CJR	2	4	∞	CJR	2	4	∞	CJR
MAZE	20	**13**	8	6	11	351.9	435.2	**144.4**	192.8	47.2	22.0	**11.7**	77.3	41723	69368	**27900**	156886
$a = 10$	10	**8**	6	5	7	243.6	564.8	**169.1**	225.5	48.3	25.0	**12.2**	79.6	39909	67417	**26155**	119374
$a = 15$	10	**5**	2	1	4	**525.2**	-	-	-	**45.4**	-	-	-	43989	71807	**30080**	194397
TABLEMOVER	24	**15**	12	**15**	-	**263.3**	336.5	341.0	-	**58.7**	59.0	61.5	-	7487	13127	**4667**	-
$a = 2$	12	10	10	**11**	-	**103.8**	226.4	214.6	-	63.5	**62.0**	64.5	-	3450	6154	**2098**	-
$a = 4$	12	**5**	2	4	-	**558.2**	-	-	-	**49.0**	-	-	-	11524	20100	**7236**	-
WORKSHOP	20	**15**	13	13	6	132.3	298.6	**51.8**	629.0	35.7	37.0	**32.5**	63.5	18002	31000	11502	**5425**
$a = 4$	10	**8**	**8**	**8**	5	42.1	261.6	**36.6**	587.3	**37.3**	43.9	**37.3**	65.8	7772	13621	4847	**2351**
$a = 8$	10	**7**	5	5	1	235.5	357.8	**76.0**	-	33.9	26.0	**24.8**	-	28231	48378	18157	**8499**
BOXPUSHING	69	39	56	**59**	-	**26.8**	79.9	63.9	-	**9.4**	11.0	10.2	-	3075	5360	**1932**	-
$a = 2$	21	**21**	**21**	**21**	-	**14.1**	15.6	16.2	-	10.5	10.6	**10.3**	-	2099	3775	**1261**	-
$a = 3$	48	18	35	**38**	-	**41.5**	118.5	90.2	-	**8.2**	11.2	10.2	-	3502	6054	**2226**	-
$l = 0$	21	**18**	17	**18**	-	**41.5**	64.7	53.3	-	**8.2**	8.3	8.3	-	3373	5887	**2116**	-
$l > 0$	27	-	18	**20**	-	-	169.2	**123.5**	-	-	13.9	**12.0**	-	3602	6184	**2312**	-

Table 2: Summary of results for domains requiring concurrency. Planners "2" and "4" are compilations having joint action size bounded by 2 and 4 respectively, while "∞" is the variant with unbounded joint action size. CJR is the compilation proposed by Crosby, Jonsson, and Rovatsos (2014). a is the number of agents, N is number of instances; time and length are averages over all instances solved, for all planners that solved at least 5 instances. For BOXPUSHING, l is the number of large boxes. The total number of actions is an average over all instances (solutions are not required to get this metric).

r_2. As n grows, the time required by FD for grounding increases due to the memory requirements. For $n = 6$, FD needs 6 seconds to find a solution; for $n = 7$, it needs 110 seconds; and for $n > 7$ it surpasses the memory limit.

Related Work

Several other authors consider the problem of concurrent multiagent planning. Boutilier and Brafman (2001) describe a partial-order planning algorithm for solving MAPs with concurrent actions, based on their formulation of concurrency constraints, but do not present any experimental results. CMAP (Borrajo 2013) produces an initial sequential plan for solving a MAP, but performs a post-processing step to compress the sequential plan into a concurrent plan.

Jonsson and Rovatsos (2011) present a best-response approach for MAPs with concurrent actions, where each agent attempts to improve its own part of a concurrent plan while the actions of all other agents are fixed. However, their approach only serves to improve an existing concurrent plan, and is unable to compute an initial concurrent plan. FMAP (Torreño, Onaindia, and Sapena 2014) is a partial-order planner that also allows agents to execute actions in parallel, but the authors do not present experimental results for MAP domains that require concurrency.

The planner of Crosby, Jonsson, and Rovatsos (2014) is similar to ours in that it also converts CMAPs into classical planning problems. The authors only present results from the MAZE domain, and concurrency constraints are defined as affordances on object sets that appear as arguments of actions. These concurrency constraints are not as flexible as those of Boutilier and Brafman (2001), since the latter can model any arbitrary concurrency constraint between pairs of actions, as well as unidirectional constraints that only affect one of the two actions. Moreover, affordances are not used to define concurrency constraints in conditional effects.

Brafman and Zoran (2014) extended the MA-STRIPS modeling language to support concurrency constraints, and the MAFS multiagent distributed planner to solve this kind of problems while preserving privacy. They examined the scalability of their approach in the BOXPUSHING domain.

Compilations from multiagent to classical planning have also been considered by other authors. Muise, Lipovetzky, and Ramirez (2015) proposed a transformation to respect privacy among agents. The resulting classical planning problem was then solved using a centralized classical planner as in our approach. Besides, compilations to classical planning have also been used in temporal planning, obtaining state-of-the-art results in many of the International Planning Competition domains (Jiménez, Jonsson, and Palacios 2015).

Conclusion

This work makes several contributions to concurrent planning. A common framework is introduced for different planning forms. We focused on the relation between concurrent and multiagent planning. We proposed a sound and complete method for compiling CMAPs into classical planning problems. The method does not need an exponential number of actions to represent the problem; instead, the number of resulting actions is linear in the description of the CMAP while respecting explicit concurrency constraints.

In future work, it would be interesting to explore strategies for encouraging concurrent actions that involve several agents (i.e. strategies for finding shorter concurrent plans). Furthermore, privacy preserving is a central topic on multiagent planning; thus, this approach could be combined with suitable privacy-preserving mechanisms in the future.

Acknowledgments

This work has been supported by the Maria de Maeztu Units of Excellence Programme (MDM-2015-0502).

References

Borrajo, D. 2013. Plan Sharing for Multi-Agent Planning. In *DMAP 2013 - Proceedings of the Distributed and Multi-Agent Planning Workshop at ICAPS*, 57–65.

Boutilier, C., and Brafman, R. I. 2001. Partial-Order Planning with Concurrent Interacting Actions. *J. Artif. Intell. Res. (JAIR)* 14:105–136.

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 28–35.

Brafman, R. I., and Zoran, U. 2014. Distributed Heuristic Forward Search with Interacting Actions. In *Proceedings of the 2nd ICAPS Distributed and Multi-Agent Planning workshop (ICAPS DMAP-2014)*.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A Single-Agent Approach to Multiagent Planning. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 237–242.

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*.

Crosby, M. 2013. A temporal approach to multiagent planning with concurrent actions. *PlanSIG*.

Crosby, M. 2014. *Multiagent Classical Planning*. Ph.D. Dissertation, University of Edinburgh.

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

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Jiménez, S.; Jonsson, A.; and Palacios, H. 2015. Temporal Planning With Required Concurrency Using Classical Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*.

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*.

Komenda, A.; Stolba, M.; and Kovacs, D. L. 2016. The International Competition of Distributed and Multiagent Planners (CoDMAP). *AI Magazine* 37(3):109–115.

Kovacs, D. L. 2012. A Multi-Agent Extension of PDDL3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, 19–27.

Maliah, S.; Brafman, R. I.; and Shani, G. 2017. Increased Privacy with Reduced Communication in Multi-Agent Planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, 209–217.

Muise, C.; Lipovetzky, N.; and Ramirez, M. 2015. MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning. In *Competition of Distributed and Multiagent Planners*.

Nardi, D.; Noda, I.; Ribeiro, F.; Stone, P.; von Stryk, O.; and Veloso, M. 2014. RoboCup soccer leagues. *AI Magazine* 35(3):77–85.

Nissim, R., and Brafman, R. I. 2014. Distributed Heuristic Forward Search for Multi-agent Planning. *J. Artif. Intell. Res.* 51:293–332.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Sheh, R.; Schwertfeger, S.; and Visser, A. 2016. 16 years of robocup rescue. *KI - Künstliche Intelligenz* 30(3):267–277.

Stolba, M.; Fiser, D.; and Komenda, A. 2016. Potential Heuristics for Multi-Agent Planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, 308–316.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: Distributed cooperative multi-agent planning. *Appl. Intell.* 41(2):606–626.

Tozicka, J.; Jakubuv, J.; and Komenda, A. 2014. Generating Multi-Agent Plans by Distributed Intersection of Finite State Machines. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 1111–1112.

Counterplanning using Goal Recognition and Landmarks

Alberto Pozanco, Yolanda E-Martín, Susana Fernández, Daniel Borrajo
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganés (Madrid). Spain
apozanco@pa.uc3m.es, yescuder@inf.uc3m.es, sfarregu@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

In non-cooperative multi-agent systems, agents might want to prevent the opponents from achieving their goals. One alternative to solve this task would be using counterplanning to generate a plan that allows an agent to block other's to reach their goals. In this paper, we introduce a fully automated domain-independent approach for counterplanning. It combines; goal recognition to infer an opponent's goal; landmarks' computation to identify subgoals that can be used to block opponents' goals achievement; and classical automated planning to generate plans that prevent the opponent's goals achievement. Experimental results in several domains show the benefits of our novel approach.

Introduction

In non-cooperative multi-agent systems, agents might want to prevent the opponents from achieving their goals. This task has been named counterplanning (Carbonell 1981). Examples of non-cooperative multi-agent domains where this approach can provide great benefits are police controls, cyber security, or real-time strategy games, where this ability has been identified as one of the major challenges for Artificial Intelligence (Ontañón et al. 2013). Most previous counterplanning approaches are based on domain-dependent solutions, such as rule-based systems (Carbonell 1978; Rowe 2003), or Hierarchical Task Networks (HTN) (Willmott et al. 2001).

Recently, there has been increasing interest in the study and generation of agents capable of reasoning about their own and opponents' goals as well as their environment (Cox 2007). Some works follow the Goal-Driven Autonomy (GDA) process, which integrates a diverse set of AI components such as HTN planning or explanation generation (Molineaux, Klenk, and Aha 2010; Weber, Mateas, and Jhala 2010). Other works combine goal recognition and reasoning on actions, applying those techniques in domains such as identifying terrorist activity (Jarvis, Lunt, and Myers 2005), air combat (Borck et al. 2015), real-time strategy games (Kabanza et al. 2010), or cyber security (Boddy et al. 2005; Edelkamp et al. 2009; Sarraute, Buffet, and Hoffmann 2012; Obes, Sarraute, and Richarte 2013; Hoffmann 2015). Again, these approaches are domain-dependent. On the goal recognition side, they use plan (Kabanza et al. 2010), rules (Carbonell 1978) or behavior (Borck et al. 2015) libraries to detect their opponent's goals. On the action reasoning side, they use stored policies (Carbonell 1981), ask for human guidance following a mixed-initiative paradigm (Jarvis, Lunt, and Myers 2005), or require heavy knowledge engineering processes such as HTN based approaches (Willmott et al. 2001).

In this paper we present a fully automatic domain-independent approach for counterplanning. This approach is based on: goal recognition, landmarks, and classical automated planning. Goal recognition aims to infer an agent's plan or goals from a set of observations. In general, the observed agent can be cooperative or competitive. We use this technique to infer an opponent's goals. Fact landmarks are propositions that must be true in all valid solution plans (Hoffmann, Porteous, and Sebastia 2004). We use landmarks to identify subgoals that can be used to block the opponent's goal achievement. Classical automated planning aims to generate a sequence of actions, namely a plan, which achieves some goals from an initial state. We use it to generate plans that prevent the opponent's goal achievement.

The main idea of this novel approach is to: (1) quickly identify the actual opponent's goal g using planning-based goal recognition techniques; (2) compute the set of landmarks involved in the achievement of g; (3) select a *counterplanning landmark*, which is the first landmark where the opponent could be blocked; and (4) generate a plan to achieve the counterplanning landmark, and therefore to block the opponent's goal achievement. This approach shows how an opponent can be effectively blocked in different non-cooperative domains.

The rest of the paper is organized as follows. In the next section we review the basic notions of classical planning, goal recognition, and landmarks. Then we introduce our fully automatic domain-independent counterplanning approach, which includes the quick detection of goals using goal recognition, and the landmark's computation to identify relevant counterplanning landmarks. Finally we present an empirical study and discuss future work.

Background

Automated Planning

Automated Planning is the task of choosing and organizing a sequence of actions such that, when applied in a given initial state, it results in a goal state (Ghallab, Nau, and Traverso 2004). Formally, a single-agent STRIPS planning task can be defined as a tuple $\Pi = \langle F, A, I, G \rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals. Each action $a \in A$ is described by a set of preconditions (pre(a)), which represent literals that must be true in a state to execute an action, and a set of effects (eff(a)), which are the literals that are added (add(a) effects) or removed (del(a) effects) from the state after the action execution. The definition of each action might also include a cost $c(a)$ (the default cost is one). The execution of an action a in a state s is defined by a function γ such that $\gamma(s, a) = (s \setminus \text{del}(a)) \cup \text{add}(a)$ if pre(a)$\subseteq s$, and s otherwise (it cannot be applied). The output of a planning task is a sequence of actions, called a plan, $\pi = (a_1, \ldots, a_n)$. The execution of a plan π in a state s can be defined as:

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \ldots, a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases}$$

A plan π is valid if $G \subseteq \Gamma(I, \pi)$. The plan cost is commonly defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$. We will use the function PLANNER(Π) to refer to an algorithm that computes a plan π from a planning task Π.

Goal Recognition

Goal Recognition is the task of inferring another agent's goals through the observation of its interactions with the environment. The problem has captured the attention of several computer science communities (Albrecht et al. 1997; Geib and Goldman 2009; Sukthankar et al. 2014). Among them, planning-based goal recognition approaches have been shown to be a valid domain-independent alternative to infer agents' goals (Ramírez and Geffner 2009; 2010; 2011; Pattison and Long 2010; E-Martín, R-Moreno, and Smith 2015; Vered and Kaminka 2017; Pereira, Oren, and Meneguzzi 2017). Ramírez and Geffner [2010] developed an approach that assumes observations are actions, and formally defined a planning-based goal recognition problem as:

Definition 1 (Goal Recognition Problem) *A goal recognition problem is a tuple $T = \langle P, \mathcal{G}, O, Pr \rangle$ where $P = \langle F, A, I \rangle$ is a planning domain and initial conditions, $\mathcal{G}$ is the set of possible goals G, $G \subseteq F$, $O = (o_1, ..., o_m)$ is an observation sequence with each o_i being an action in A, and Pr is a prior probability distribution over the goals in $\mathcal{G}$.*

The solution to a goal recognition problem is a probability distribution over the set of goals $G \in \mathcal{G}$ giving the relative likelihood of each goal. In this work we assume that Pr is uniform. We will use the function RECOGNIZEGOALS($F, A, I, \mathcal{G}, O$) to refer to an algorithm that solves the goal recognition problem. This function returns a list of tuples in the form of $\langle$goal, probability$\rangle$ for each goal in $\mathcal{G}$.

Landmarks

In Automated Planning, landmarks were initially defined as sets of propositions that have to be true at some time in every solution plan (Hoffmann, Porteous, and Sebastia 2004). Formally:

Definition 2 (Fact Landmark) *Given a planning task $\Pi = \langle F, A, I, G \rangle$, a formula $L_\Pi \in F$ is a fact landmark of Π iff L_Π is true in some state along all valid plans executions that achieve G from I.*

This definition was later extended to include action landmarks (Richter and Westphal 2010). We will use the function EXTRACTLANDMARKS(F, A, I, G) to refer to an algorithm that computes a set of landmarks $\mathcal{L}_\Pi$ from a planning task Π.

Domain-Independent Counterplanning

We first formalize the two actors involved in a counterplanning problem as planning agents.

Definition 3 (Seeking agent) *A seeking agent ϕ is an agent that has an associated planning task $\Pi_\phi = \langle F_\phi, A_\phi, I_\phi, G_\phi \rangle$, and pursues its goal G_ϕ by following a plan π_ϕ computed from Π_ϕ.*

Definition 4 (Preventing agent) *A preventing agent α is an agent that has an associated planning task $\Pi_\alpha = \langle F_\alpha, A_\alpha, I_\alpha, G_\alpha \rangle$.*

G_α is initialized to $\emptyset$. Then, Algorithm 1 (described later) computes a set of goals to be used for the counter-planning task. There can be varied relations between Π_ϕ and Π_α, and the information that one agent has from the other. For instance, the actions that both agents can perform could be the same $A_\phi = A_\alpha$, or totally different $A_\phi \cap A_\alpha = \emptyset$. They could also have different or equal observations of the world. In this work, we make the following assumptions:

- ϕ's model is known by α except for its goal G_ϕ. In most real-world domains that we have selected for potential applications (e.g. police control, cyber security, strategy games, ...), both F_ϕ, I_ϕ, and A_ϕ can be assumed to be known;
- as in most literature on goal reasoning, α knows a set of potential goals, $\mathcal{G}_\phi \subseteq 2^{F_\phi}$, ϕ could be trying to achieve;
- deterministic action outcomes and full observability of those actions by α;
- both agents can follow optimal or suboptimal strategies to reach their goals;
- both agents stick to their plans. In other words, they do not replan or change their goals during execution; and
- the temporal duration of an action $a_i \in A$ is determined by its cost $c(a_i)$.[1]

Since both agents operate in a common environment, the execution of their actions affects the shared environment. Therefore, we assume any state of the environment s can be defined in terms of the set of propositions F_e ($s \subseteq F_e$), such

[1]In this paper, we assume unit action costs.

that $F_\phi \cup F_\alpha \subseteq F_e$. Additionally, some propositions must be in $F_\phi \cap F_\alpha$, i.e. they will be observable and modifiable by both agents. The individual execution of actions by any of the two agents in F_e will be based on the respective action sets. Hence, the execution of an action a ($a \in A_\phi \cup A_\alpha$) in a state s is defined using the previous $\gamma(s, a)$. Furthermore, the joint execution of one action per agent in the same time step t can be defined as follows.

Definition 5 (Joint execution of two actions) *Given two actions $a_\phi \in A_\phi$ and $a_\alpha \in A_\alpha$ and an environment state $s \subseteq F_e$, the joint execution of both actions at a time step t results in a new state given by*

$$\gamma_{\phi,\alpha}(s, a_\phi, a_\alpha) = \begin{cases} \gamma(\gamma(s, a_\alpha), a_\phi) & \textit{if } a_\phi \textit{ not mutex with } a_\alpha \\ \gamma(s, a_\alpha) & \textit{otherwise} \end{cases}$$

Similarly, the joint execution of two plans $\Gamma_{\phi,\alpha}(s, \pi_\phi, \pi_\alpha)$ can be defined by the iteration of the joint execution of actions of those plans using $\gamma_{\phi,\alpha}(s, a_\phi, a_\alpha)$. For simplicity, in this paper we assume that the preventing agent always executes its action first when two actions are mutex. We define two mutex actions as follows.

Definition 6 (Mutex actions) *Two actions a_x, a_y executed at a time step t are mutex if any literal in eff(a_x) deletes (adds) any literal in pre(a_y) or if any literal in eff(a_x) deletes (adds) a literal that is added (deleted) in eff(a_y).*

Using these definitions, we can now formally describe a counterplanning task.

Definition 7 (Counterplanning task) *A counterplanning task is defined by a tuple $CP = \langle \Pi_\phi, \Pi_\alpha, \mathcal{G}_\phi, O_\phi \rangle$ where Π_ϕ is the planning task of ϕ, Π_α is the planning task for the preventing agent, $\mathcal{G}_\phi$ is the set of sets of goals that ϕ can potentially pursue, and $O_\phi = (o_1, \ldots, o_m)$ is a set of observations by α of the execution of a plan $\pi_\phi = (o_1, \ldots, o_m, a_{m+1}, \ldots, a_k)$ that solves Π_ϕ.*[2]

We assume that ϕ generates a plan π_ϕ to solve its planning task Π_ϕ prior to counterplanning, and that plan (as well as its corresponding goals) is unknown for α. Then, at some time step m of the execution of π_ϕ (where m can range from 1 to k, the length of π_ϕ), given all observed actions from the execution of π_ϕ, α has to infer the ϕ agent goals (from $\mathcal{G}_\phi$) and generate a solution to a counterplanning task, namely a counterplan.

Definition 8 (Counterplan) *Given ϕ agent plan $\pi_\phi = (a_{m+1}, \ldots, a_k)$, a plan $\pi_\alpha = (a_1, \ldots, a_n)$ is a valid counterplan for $\pi_\phi = (a_{m+1}, \ldots, a_k)$ if the joint execution of π_α and π_ϕ does not allow ϕ to achieve the goals in G_ϕ; $G_\phi \not\subseteq \Gamma_{\phi,\alpha}(s, \pi_\phi, \pi_\alpha)$.*

Our approach to solve counterplanning tasks assumes that α can delete (or add in the case of negated literals) some proposition that ϕ *needs* in order to achieve its goals. There could be different definitions for *needed literals*. We use planning landmarks in this work. Therefore, we impose two constraints: the seeking agent ϕ and the preventing agent α share some propositions, $F_\phi \cap F_\alpha \neq \emptyset$; and at least one action a in α model, $a \in A_\alpha$, must delete (add) at least one of ϕ's plan landmarks.

[2]We have changed the notation o_i for a_j in the π_ϕ plan to differentiate between observations and future actions.

Algorithm 1 shows the high-level algorithm used to solve a counterplanning task from the perspective of α. The algorithm first solves a goal recognition problem using RECOGNIZEGOALS given a planning domain, initial conditions, a set of candidate goals $\mathcal{G}_\phi$, and a set of observations O_ϕ. It returns T_ϕ, a probability distribution over the set of candidate goals set $\mathcal{G}_\phi$ in the form of tuples $\langle$goal, probability$\rangle$. Then, the initial state of ϕ, I_ϕ, is updated with the given observations by advancing the state from the initial I_ϕ and applying all actions corresponding to the observations in O_ϕ. Next, we select the set of most probable goals' sets $G\prime_\phi$ from T_ϕ. For each goal $g \in G\prime_\phi$, we extract the landmarks of the new ϕ planning task using EXTRACTLANDMARKS. This computation will return the set of common landmarks among all the most probable sets of goals, $\mathcal{L}_{\Pi_\phi}$. Figure 1 shows an example of that computation in a navigation domain. If there are not common landmarks, the counterplanning task cannot be performed. Otherwise, the algorithm selects the set of counterplanning landmarks $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$ in EXTRACTCPLANDMARKS. This process will be explained in detail later. As before, if there are not counterplanning landmarks, the counterplanning task cannot be performed. Otherwise, one of the landmarks in $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$ is negated and returned as the preventing agent's goal G_α in SELECTGOAL. Finally, a plan π_α is computed to achieve that goal such that it prevents ϕ from achieving its goals. In the next section we discuss how we select G_α from $\mathcal{L}_{\Pi_\phi}$.

Algorithm 1 DOMAIN-INDEPENDENT COUNTERPLANNING

```
Inputs: $\Pi_\phi, \Pi_\alpha, \mathcal{G}_\phi, O_\phi$
Outputs: $\pi_\alpha$
 1: $T_\phi \leftarrow$ RECOGNIZEGOALS($F_\phi, A_\phi, I_\phi, \mathcal{G}_\phi, O_\phi$)
 2: $I_\phi \leftarrow$UPDATE($I_\phi, A_\phi, O_\phi$)
 3: $\mathcal{L}_{\Pi_\phi} \leftarrow F_\phi$
 4: $G\prime_\phi \leftarrow$ goal($\arg\max_{t \in T_\phi}$ probability($t$))
 5: $\pi_\alpha \leftarrow \emptyset$
 6: for $g \in G\prime_\phi$ do
 7:     $\mathcal{L}_{\Pi_\phi} \leftarrow \mathcal{L}_{\Pi_\phi} \cap$ EXTRACTLANDMARKS($F_\phi, A_\phi, I_\phi, g$)
 8: if $\mathcal{L}_{\Pi_\phi} \neq \emptyset$ then
 9:     $\mathcal{L}_{\Pi_\phi,\Pi_\alpha} \leftarrow$EXTRACTCPLANDMARKS($\Pi_\phi, \Pi_\alpha, \mathcal{L}_{\Pi_\phi}$)
10:     if $\mathcal{L}_{\Pi_\phi,\Pi_\alpha} \neq \emptyset$ then
11:         $G_\alpha \leftarrow$SELECTGOAL($\Pi_\phi, \Pi_\alpha, \mathcal{L}_{\Pi_\phi,\Pi_\alpha}$)
12:         $I_\alpha =$UPDATE($I_\alpha, A_\alpha, O_\phi$)
13:         $\pi_\alpha \leftarrow$ PLANNER($\Pi_\alpha = (F_\alpha, A_\alpha, I_\alpha, G_\alpha)$)
14: return $\pi_\alpha$
```

Selecting goals from landmarks

Given a set of common landmarks L_{Π_ϕ}, two questions arises: (1) how many of those fact landmarks could be deleted (added) by α's model of the world (domain), so ϕ cannot achieve them?; and (2) within those facts that α can delete (add), which one should it become its goal G_α to ef-

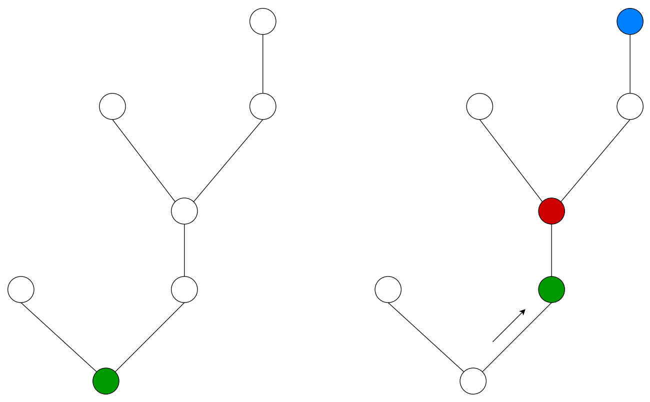


Figure 1: Navigation example. The green node illustrates the current position of the agent. The rest of the nodes represent a possible goal. The red node indicates a common landmark among the remaining goals after observing an action (moving to the right). The blue node refers to the actual goal of the seeking agent.

fectively stop ϕ from achieving its goal? The first question brings us to the following definition:

Definition 9 Counterplanning landmark *Given the set of fact landmarks from Π_ϕ, $\mathcal{L}_{\Pi_\phi}$, a landmark $l_i \in \mathcal{L}_{\Pi_\phi}$ is a counterplanning landmark for α if $\exists a \in A_\alpha$ with $l_i \in eff(a)$. If l_i is a positive literal, l_i must be in $del(a)$. If l_i is a negative literal, l_i must be in $add(a)$.*

All the fact landmarks that comply with this condition are added to the counterplanning landmarks set of both agents $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$. We will refer to this process as EXTRACTCPLANDMARKS($\Pi_\phi, \Pi_\alpha, \mathcal{L}_{\Pi_\phi}$).

The second question requires further analysis. Given the definition of a fact landmark, α only has to delete (add) a fact landmark of Π_ϕ to prevent the seeking agent from achieving its goal. The problem therefore turns into selecting a single goal from $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$ to become the next goal for α, G_α. In most counterplanning domains, the earlier we discover the opponent's intentions (and thus stop him/her from achieving its goal), the better. This is a common characteristic in: (1) cyber security domains where we want to detect an intruder as soon as possible; (2) real-time strategy games where we want to defeat our enemy in the shortest possible time; or (3) a medical domain where we want to prevent the disease from spreading at the earliest time. Thus, this type of problems presents some temporal aspects that we need to take into account. In particular, it is not useful for α to pursue a counterplanning fact landmark that ϕ is going to achieve before α can avoid it. We define this temporal subproblem as finding the *First Counterplanning Landmark*, FCL. Algorithm 2 shows the high-level algorithm used to find it. For each counterplanning landmark l_i, an optimal plan is computed for ϕ and α. It is done optimally to ensure that the returned values correspond to the shortest time (cost) when both agents could reach that subgoal.

If the cost (duration) of achieving $\neg l_i$ by α solving Π_α is smaller than the cost (duration) of achieving l_i by ϕ, solving Π_ϕ, and there is no other landmark l_j with smaller cost, then $\neg l_i$ becomes FCL. In other words, l_i is the first landmark in Π_ϕ that α can achieve before ϕ. Therefore, the new G_α will be the negated FCL, since we want α to avoid ϕ achieving FCL. As a reminder, we are performing a one-step counterplanning episode. If ϕ performs some actions to re-achieve FCL or change its goals, then we assume α would have to start a new counterplanning episode.

Algorithm 2 SELECT GOAL

```
Inputs: $\Pi_\phi, \Pi_\alpha, \mathcal{L}_{\Pi_\phi,\Pi_\alpha}$
Outputs: FCL
 1: FCL $\leftarrow \emptyset$
 2: FCLCost $\leftarrow 0$
 3: for $l_i$ in $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$ do
 4:     $\pi_\phi \leftarrow$ PLANNER($\Pi_\phi = \langle F_\phi, A_\phi, I_\phi, l_i \rangle$)
 5:     $\pi_\alpha \leftarrow$ PLANNER($\Pi_\alpha = \langle F_\alpha, A_\alpha, I_\alpha, \{\neg l_i\} \rangle$)
 6:     if $c(\pi_\phi) >= c(\pi_\alpha)$ then
 7:         if $c(\pi_\phi) <$ FCLCost then
 8:             FCLCost $\leftarrow c(\pi_\phi)$
 9:             FCL$\leftarrow \neg l_i$
10: return FCL
```

Example

To illustrate our approach, let us consider a simple domain where a terrorist has committed an attack in the center of a city. Figure 2 shows the road network for this problem. The city police (α) knows that the terrorist (ϕ) wants to leave the city by either $G1$ (airport), $G2$ (train station), or $G3$ (bus terminal); so, $\mathcal{G}_\phi = \{G1, G2, G3\}$. The police has control over some cameras located at key points around the city (represented as nodes in Figure 2). The police actions consist of stopping the terrorist by setting a control at any of those points (A_α). So, the police wants to: (1) quickly know where the terrorist wants to go ($G\prime_\phi$); and (2) stop him/her as soon as possible to avoid panic breaking out. When the cameras observe that the terrorist is at L1 ($O_1 = a_1 \in A_\phi$), the police guesses that his/her goal is to reach $G1$ ($G\prime_\phi$) by doing goal recognition. The police only has resources to set one control. It knows that the terrorist must pass through L1, L2 and L3 to reach the airport. Although these four spots are counterplanning landmarks $\mathcal{L}_{\Pi_\phi,\Pi_\alpha}$, the police can only set the control at L2, L3 and $G1$ before the terrorist reaches those places. Finally, the police goal G_α will be to set the control at L2 since it is the FCL; i.e. the first spot where the terrorist can be effectively stopped.

Experiments and Evaluation

We empirically evaluate our approach on the new previously described TERRORIST domain as well as in other domains usually used in goal recognition works such as LOGISTICS, EASY IPC GRID, BLOCKS, and INTRUSION DETECTION. Each domain and problem conforms Π_ϕ. Additionally, in order to perform counterplanning, for each domain we have generated a new counterplanning domain, which defines the planning task Π_α for α. The classical domain and the counterplanning domain comply with the requirements mentioned in Section .

						HSP*$_f$				LAMA				Greedy LAMA			
Domain	$\|\mathcal{G}_\phi\|$	$\|\pi_\phi\|$	$\|\pi_\alpha\|$	$\|\mathcal{L}_{\Pi_\phi}\|$	%Obs	Q	Q_t	E	Pe	Q	Qt	E	Pe	Q	Qt	E	Pe
LOGISTICS	6	9.4	1	12.3	10	0.6	4.2	0.7	0.3	0.6	5.0	0.7	0.3	0.6	2.6	0.6	0.2
					30	0.6	7.5	0.7	0.4	0.6	10.7	0.7	0.4	0.7	2.6	0.8	0.4
					50	0.7	10.0	0.8	0.5	0.7	16.3	0.8	0.5	0.7	2.7	0.8	0.5
					70	0.9	14.3	0.6	0.8	0.9	28.8	0.6	0.8	0.9	2.8	0.7	0.8
TERRORIST	4	3.8	1	3.8	10	0.5	3.8	0.6	0.5	0.6	2.8	0.6	0.5	0.6	2.6	0.6	0.5
					30	0.6	4.0	0.8	0.7	0.6	2.9	0.8	0.7	0.6	2.8	0.8	0.7
					50	0.6	5.0	0.8	0.8	0.6	3.0	0.8	0.8	0.6	2.9	0.8	0.8
					70	0.9	5.1	0.9	1.0	0.9	3.3	0.9	1.0	0.9	3.0	0.9	1.0
	10	5.6	1	4.1	10	0.3	283.2	0.5	0.5	0.3	211.4	0.5	0.5	0.3	210.4	0.5	0.5
					30	0.5	287.2	0.6	0.8	0.4	235.0	0.6	0.8	0.4	227.3	0.6	0.8
					50	0.6	307.1	0.4	0.8	0.6	248.6	0.4	0.8	0.6	269.6	0.4	0.8
					70	0.6	325.8	0.4	1.0	0.6	321.1	0.4	1.0	0.6	322.7	0.4	1.0
INTRUSION DETECTION	6	4.3	1	4.8	10	1.0	0.5	1.0	0.4	1.0	1.0	1.0	0.4	1.0	0.7	1.0	0.4
					30	1.0	0.4	1.0	0.7	1.0	1.0	1.0	0.7	1.0	0.7	1.0	0.7
					50	1.0	0.4	1.0	0.7	1.0	1.0	1.0	0.7	1.0	1.0	1.0	0.7
					70	1.0	0.3	1.0	0.9	1.0	0.8	1.0	0.9	1.0	1.0	1.0	0.9
BLOCKS	10	8.6	1	17.6	10	0.2	836.6	0.4	0.2	0.2	482.1	0.4	0.2	0.2	155.3	0.4	0.2
					30	0.3	910.7	0.6	0.5	0.3	531.9	0.6	0.5	0.3	175.3	0.6	0.5
					50	0.5	980.3	0.8	0.5	0.5	602.7	0.8	0.5	0.6	195.8	0.9	0.6
					70	0.8	1070.3	0.7	0.8	0.8	655.3	0.7	0.8	0.8	207.8	0.7	0.7
EASY IPC GRID	4	12.6	4.5	6.3	10	0.3	5.6	0.1	1.0	0.3	1.9	0.1	1.0	0.1	1.7	0.1	0.8
					30	0.3	7.3	0.3	0.6	0.3	2.0	0.3	0.6	0.3	2.0	0.4	0.78
					50	0.1	9.8	0.3	0.7	0.1	2.8	0.3	0.7	0.1	2.5	0.4	0.8
					70	0.3	15.3	0.0	∞	0.3	3.5	0.0	∞	0.3	3.1	0.0	∞

Table 1: Comparison of the counterplanning approach in five domains using optimal and approximated goal recognition methods. Figures shown are all averages over the set of problems as explained in the text. The metrics measured are: size of $\mathcal{G}_\phi$, length of plans for each agent, number of landmarks $|\mathcal{L}_{\Pi_\phi}|$, percentage of observations, goal recognition accuracy Q and its time Q_t, counterplanning accuracy E and penalty value Pe.

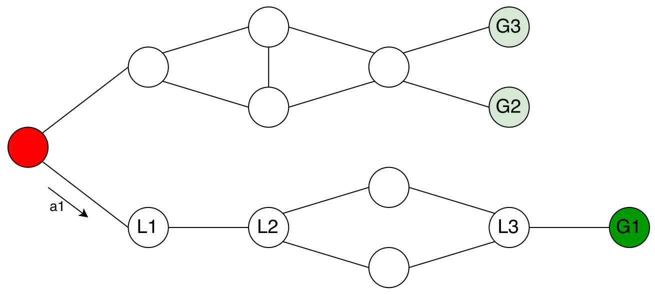


Figure 2: Terrorist capture domain. The red circle represents the initial position of the terrorist. Green circles represent his/her possible goals. Arrow a1 indicates the terrorist first observed action. L1, L2 and L3 refer to the landmark points that the terrorist has to pass through in order to reach $G1$.

For each classical domain, we generate 10 random problems for ϕ. All the problems have the same number of objects. However, each problem has a different actual goal for ϕ, which is hidden for α. Their details are explained below.

- LOGISTICS. ϕ can deliver any package to any destination. It can do it by driving either trucks or planes. α can break a truck or a plane to interrupt the delivery.
- TERRORIST. ϕ may want to get to any point in the map. It can do it by navigating through points that are connected. α can set a control at a point so that they can arrest ϕ.
- INTRUSION DETECTION. ϕ may want to perform a set of attacks to a pool of computers. It can do it by performing hacking actions like delete logs or gain root access. α can perform administration actions like encrypt the information or change the root password to a computer to prevent ϕ from conducting its attack.
- BLOCKS. ϕ may want to put any of block on top of another. It can do it by picking up and stacking blocks that are not painted. α can paint a block so ϕ can not pick it up and preventing it from achieving its goal.
- EASY IPC GRID. ϕ may want to get to any cell in the grid. It can do it by navigating through cells without door or, if it has the right key, by opening them. α can navigate through those cells without needing a key, and can steal the key or change the lock of a door to block ϕ's plan.

The set of candidate goals $\mathcal{G}_\phi$ always consists of a 20% of all the possible goals in each problem. Therefore, bigger $\mathcal{G}_\phi$

sets for the same domain mean bigger problems. In particular, for the TERRORIST domain with a problem map of 20 nodes, $\mathcal{G}_\phi$ consists of 4 random destinations; when the map has 50 nodes, $\mathcal{G}_\phi$ consists of 10 possible destinations. The hidden goal, i.e., the current goal of ϕ, G_ϕ, that is unknown to α, is always on $\mathcal{G}_\phi$.

The set of observed actions was taken to be a subset of the plan solution π_ϕ, ranging from 10% of the actions, up to 70% of the actions. We did not include tests where the observed sequence is higher than 70% because our counterplanning approach degrades rapidly. The reason for this is that the number of counterplanning landmarks decreases as the number of observed actions increases.

Our fully automatic domain-independent counterplanning approach works with any combination of goal recognition and classical planning approaches. For purposes of these tests, we have selected the following configuration of goal recognition techniques and planners. For the goal recognition part of our counterplanning technique, we have tested the aforementioned domains and problems using the Ramírez and Geffner [2010] approach with different planners. The planners are HSP*$_f$ (Haslum 2008), an optimal planner; and LAMA (Richter, Westphal, and Helmert 2011), a satisficing planner. LAMA is used in two modes: as a *greedy* planner that stops after the first plan is found GREEDY LAMA; and as an *anytime* planner that reports the best plan found in a given time window LAMA. The planning times for all the planners were set to 1800 seconds. In all the domains π_ϕ is computed using GREEDY LAMA. For optimal plan computations of FCL, we use HSP*$_f$. All the experiments were ran on a Ubuntu machine with Intel Core 2 Quad Q8400 running at 2.66 GHz.

Table 1 summarizes the experimental results. For each planner, each row shows average performance over the 10 problems in each domain. Each column represents different measures of quality and performance:

- $|\mathcal{G}_\phi|$: number of goals in the candidate goal's set.
- $|\pi_\phi|$, $|\pi_\alpha|$: average plan length cost for each agent.
- $|\mathcal{L}_{\Pi_\phi}|$: number of landmarks of the seeking agent planning task.
- %Obs: percentage of the actions of π_ϕ in O_ϕ. Higher percentages of observations mean that more actions of ϕ's plan have already been observed by α and, thus, executed by ϕ.
- Q: fraction of times that the actual goal G_ϕ was found to be the most likely goal $G\prime_\phi$. In our experiments, if $G\prime_\phi$ consist of more than one goal, we select the one with more counterplanning landmarks as the most likely goal. Ideally, $Q = 1$.
- Q_t: average time in seconds taken for solving the goal recognition problems.
- E: fraction of times that α executing π_α succeeds in stopping ϕ in achieving its goals. Ideally, $E = 1$.
- Pe: penalty value computed as the number of steps in π_ϕ that are successfully performed divided by the length of π_ϕ. This penalty value represents the cost paid by π_α at each time step that has not stopped π_ϕ. Lower values of Pe indicate better performance, ideally $Pe = 0$.

As we can see in all the domains, the higher percentage of observations, the higher Q values, as expected. The goal recognition task becomes easier as more actions have been observed (as reported in other goal recognition works). Regarding E, the fraction of times that α blocks ϕ achieving its goals is clearly related to Q. Guessing the opponent's goal right usually involves more opportunities to block it. This is the case of INTRUSION DETECTION, where our counterplanning approach performs well. However, there are some cases in which we can badly guess ϕ's goal and still block its goal achievement (Q value is lower than E). This happens when our analysis of the goal recognition process identifies a common landmark (to stop ϕ's plan), but selects a wrong goal as in some BLOCKS instances.

The value of E is also closely related to the percentage of observations. Lower percentage values allow α to find many landmarks where to effectively block ϕ. On the other hand, if most of the actions in π_ϕ have already been observed (i.e. executed by the seeking agent), there will be just a few counterplanning landmarks to prevent ϕ from achieving its goal. This is also connected with the penalty values Pe. Lower percentage of observations imply that, if the opponent can be blocked, it could be done farther from the goal than if 50% or more of the plan has already been observed.

The number of landmarks of ϕ's planning task affects the counterplanning results. Domains with a higher number of landmarks will have a higher number of potential counterplanning landmarks where to block the opponent. The experiments confirm this aspect: domains such as LOGISTICS and BLOCKS are more likely to have more landmarks and the penalty values are smaller than in the other domains which just have a few landmarks.

Regarding planners' performance, GREEDY LAMA seems to achieve the best overall results both in terms of quality (Q, E, Pe) and time (Q_t). Since LAMA is an anytime planner, sometimes it takes more time to complete the goal recognition process than the optimal planner HSP*$_f$. However, all planners scale poorly to bigger problem instances where $\mathcal{G}_\phi$ increases. This entails worse goal recognition performance Q and, therefore, worse counterplanning performance E and P. We could speed-up our technique by computing plan cost estimations instead of actual plans in order to improve the performance. That would allow its use in real-time environments.

Summarizing, the best scenario for our counterplanning technique (high E values and low Pe values) would be when the preventing agent guesses the seeker's actual goal ($Q = 1$) with a low percentage of observed actions (very soon) and there is a high number of landmarks in the seeker's planning task which the preventing agent can delete (add).

Conclusions and Future Work

We have presented a novel fully automatic domain-independent approach for counterplanning, which is based on classical planning techniques. We have formally defined the counterplanning task involving two planning agents: an agent that seeks to achieve some goals; and an agent that tries to prevent its opponent from achieving its goals. To successfully block an agent in a domain-independent way, we: (1) recognize the opponent's goals by observing its performed actions; (2) identify the counterplanning landmarks of its planning task; and (3) generate a sequence of actions to block its goal achievement process as soon as possible. Results show the benefits of our approach on preventing the opponent from achieving its goals in several domains. Its performance depends on the ability of the preventing agent to quickly infer the hidden goal, and the number of landmarks of the seeker's planning task.

In this work, we assume we are given the candidate goals for the goal recognition process (as in the usual literature on goal recognition). Future work would consist on relaxing this assumption and consider F_ϕ as $\mathcal{G}_\phi$. We also assume unit action costs. In future work, we would generalize our approach by considering non-unit action costs. Additionally, a natural extension to this work would be to assume a seeking agent capable of changing his/her plans and goals. It seems to be also possible to extend our approach to deal with noisy observations and uncertainty on the seeking agent's behavior.

Acknowledgements

This work has been partially supported by MINECO projects TIN2014-55637-C2-1-R and TIN2017-88476-C2-2-R.

References

Albrecht, D. W.; Zukerman, I.; Nicholson, A. E.; and Bud, A. 1997. Towards a bayesian model for keyhole plan recognition in large domains. In *User Modeling*, 365–376. Springer.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21.

Borck, H.; Karneeb, J.; Alford, R.; and Aha, D. W. 2015. Case-based behavior recognition in beyond visual range air combat. In *FLAIRS Conference*, 379–384.

Carbonell, J. G. 1978. Politics: Automated ideological reasoning. *Cognitive Science* 2(1):27–51.

Carbonell, J. G. 1981. Counterplanning: A strategy-based model of adversary planning in real-world situations. *Artificial Intelligence* 16(3):295–329.

Cox, M. T. 2007. Perpetual self-aware cognitive agents. *AI magazine* 28(1):32.

E-Martín, Y.; R-Moreno, M. D.; and Smith, D. E. 2015. A fast goal recognition technique based on Interaction estimates. In *IJCAI*.

Edelkamp, S.; Elfers, C.; Horstmann, M.; Schröder, M.-S.; Sohr, K.; and Wagner, T. 2009. Early warning and intrusion detection based on combined ai methods. In *ICAPS*. Citeseer.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101–1132.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. *In Proceedings of the 2008 IPC*. Sydney, Australia.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Hoffmann, J. 2015. Simulated penetration testing: From" dijkstra" to" turing test++". In *ICAPS*, 364–372.

Jarvis, P. A.; Lunt, T. F.; and Myers, K. L. 2005. Identifying terrorist activity with ai plan recognition technology. *AI Magazine* 26(3):73.

Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A. R.; and Irandoust, H. 2010. Opponent behaviour recognition for real-time strategy games. *Plan, Activity, and Intent Recognition* 10(05).

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-driven autonomy in a navy strategy simulation. In *AAAI*, 1548–1554.

Obes, J. L.; Sarraute, C.; and Richarte, G. 2013. Attack planning in the real world. *arXiv preprint arXiv:1306.4044*.

Ontañón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game AI research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4):293–311.

Pattison, D., and Long, D. 2010. Domain independent goal recognition. In *Stairs 2010: Proceedings of the Fifth Starting AI Researchers Symposium*, volume 222, 238.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. AAAI Press.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI. Morgan Kaufmann Publishers Inc*, 1778–1783.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *In Twenty-Fourth AAAI Conference on Artificial INtelligence (AAAI-10)*, 1121–1126.

Ramírez, M., and Geffner, H. 2011. Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In *IJCAI*, 2009–2014.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *International Planning Competition*, 117–124.

Rowe, N. C. 2003. Counterplanning deceptions to foil cyber-attack plans. In *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, 203–210. IEEE.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In *AAAI*.

Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, activity, and intent recognition: theory and practice*. Newnes.

Vered, M., and Kaminka, G. A. 2017. Heuristic online goal recognition in continuous domains. In *IJCAI*, 4447–4454.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010. Applying goal-driven autonomy to starcraft. In *AIIDE*.

Willmott, S.; Richardson, J.; Bundy, A.; and Levine, J. 2001. Applying adversarial planning techniques to Go. *Theoretical Computer Science* 252(1-2):45–82.

ICAPS Proceedings of the 6th Workshop on Distributed and Multi-Agent Planning (DMAP-2018)
In Proceedings of the 14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Madrid, Spain, June 2018.

An Approach for Autonomous Multi-rover Collaboration for Mars Cave Exploration: Preliminary Results

Tiago Vaquero and **Martina Troesch** and **Steve Chien**
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract

Mars caves are promising targets for planetary science and human shelter. Exploring these environments would pose several challenges, including limited communication, lack of sunlight, limited vehicles lifetime that would not allow humans in the loop, and a totally unknown environment. Mission to these underground environments would required levels of autonomy, coordination and collaboration never been deployed before in rovers. In this paper we propose a multi-rover coordination algorithm and experimental framework for cave exploration missions. We describe preliminary experimental results with this coordination algorithm in a realistic simulated cave. We analyze rover coordination performance in different environmental settings and provide insights on potential opportunities for enhanced autonomy with AI planning and scheduling.

1. Introduction

Exploration of planetary caves is becoming an active research topic in the planetary science community and a promising scientific target for autonomous robotic explorers. Mars in particular offers exciting opportunities for (1) human settlements, (2) understanding the planet's evolution, and (3) the search of extraterrestrial life. Caves present the most mission effective habitat alternative for future human exploration, offering a stable, UV-shielding, meteoric-shielding environment (Boston et al. 2003), as well as access to minerals, gases and ice. Equally important, caves may preserve valuable information about the planet's history and evolution. Specifically, they offer stable physio-chemical environments, trapped volatiles, secondary mineral precipitation and microbial growth, which are expected to preserve bio-signatures and provide a record of past climate (Boston et al. 2005; 2004). Moreover, caves can potentially host water deposits which, through interaction with volcanic heat and minerals, could have created a favorable environment to microbial life preservation. What makes planetary caves even more attractive is that they are quite abundant. Mars for example has more than 2000 cave-related features identified, commonly associated with lava tubes, which provides a variety of promising targets for future exploration missions.

Robotic exploration missions on Mars would provide unique science opportunities for the cognitive and robotics communities, however, they present several challenges. Communicating with a rover into any of these caves and transmitting science data out is in itself a hard technical problem. Without a link to the surface, a rover would not be able to go far into the cave without losing contact with a base station. Moreover, because sunlight is not available in the cave, a mission is likely to last only a few days since the rovers will rely exclusively on battery power. Given limited communication, power and mission duration (just days), it is impractical to wait for humans' commands and feedback like in current Mars operations. For example, current MSL operations requires humans in the loop to plan sequences of actions for each sol based on downlinked data (Gaines et al. 2016). Those challenges alone require rovers far more autonomous than the existing surface rovers, for their environment is quite unknown and their communication with Earth is extremely limited, if at all.

Autonomy in multi-rover coordination is a key mission enabler that would help rovers to map and explore as much of the cave as efficiently as possible. With their very limited lifetime, rovers cannot wait for large parts of each day to receive directions from ground/Earth. The need for such multi-asset coordination was identified in recent studies in Mars cave exploration (Dubowsky et al. 2005; Kesner et al. 2007; Husain et al. 2013; Thangavelautham et al. 2014) and in Mars surface exploration (Clement and Barrett 2003; Yliniemi, Agogino, and Tumer 2014). The AI community has recently started to look into coordination techniques to map and explore Mars cave environments (Husain et al. 2013). One traditional approach would be to use a centralized task allocation and communication architecture to coordinate the rovers during exploration (Chien et al. 2000; Clement, Durfee, and Barrett 2007). However, this approach becomes unfeasible in a realistic cave environment due to intermittent, unreliable communication, as well as the high cost of communication power associated with the centralized scheme. Some existing work explores distributed techniques to coordinate vehicles to maintain connectivity between a base robot and a mobile explorer at all times in more controlled environments (auf der Heide and Schneider 2008; Stump, Jadbabaie, and Kumar 2008). These approaches can be leveraged to address subsurface missions, but they would need to be contextualized to environments with high likelihood of connectivity loss between rovers (sometimes done proactively by rovers to increase science utility) and unknown density and geometry of obstacles. Research on multi-rover coordination under these challenging constraints is in its infancy.

In this work, we propose a multi-rover coordination strategy for cave exploration that aims to send rovers as deep into the cave as possible while also maximizing science data sent out to a surface base station. The proposed *Dynamic Zonal Relay with Sneakernet Relay Algorithm* is a two step algorithm. The first phase of the algorithm (Dynamic Zonal) drives each rover to a designated zone along the length of the cave, while maintaining communication distance between neighboring rovers. Each rover only takes science data in its designated zone and transmits it to the neighboring rover in the direction of the base station. Once at the end of its zone, the rover stops and becomes a relay point. The dynamic part of this algorithm is that if a rover is no longer operable, the other rovers would re-distribute the zones to maintain communication and characterization of the environment. The next step of the algorithm (Sneakernet Relay) would allow the rovers to acquire science data further in the cave by driving beyond the communication distance (intentional communication lost) and driving between neighboring rovers to relay the data out of the cave. We implement a simulation framework that (1) allows different multi-rover mission configurations, as well as (2) supports the measurement, evaluation, visualization and analysis of rover performance. We present preliminary results on the rovers and algorithm performance in a realistic simulated cave environment and rover configuration, including power and communication constraints, and science instruments and navigation system specification. The results provide initial insights to future mission design space, direction for algorithm improvements, as well as interesting opportunities for task planning and scheduling that would improve rover coordination, operation, communication and science return.

This paper is organized as follows. We first describe the multi-rover coordination problem for Mars cave exploration we address in this work and the particular elements of the mission. We then present the Dynamic Zonal Relay with Sneakernet Relay Algorithm in more detail. Next, we provide experimental results from a set of simulated Mars Cave exploration scenarios, in which a team of four rovers explore a realistic-size cave with varying obstacle densities. We analyze the performance of the coordination algorithms with respect to a score based on cave coverage transmitted out of the cave, mission life span, distribution of energy and time spent in different rover activities. Finally, we discuss the results, potential roles for AI planners and schedulers, as well as future directions.

2. Example Problem Definition

Among the several mission challenges related to deploying and controlling a set of rovers in a Mars cave, in this work we focus on the hypothetical problem of autonomously coordinating multiple rovers to (1) map and characterize a Martian cave as far into the cave as possible from the entrance, and (2) to transmit as much science data collected by the rovers' instruments as possible out of the cave to a lander (base station), which will then take care of transmitting it to scientists on Earth. Figure 1 illustrate a Martian lave tube structure, with the lander positioned at the entrance and a set of science rovers exploring the cave interiors. We provide more details and constraints on the cave environment and rover platform in what follows.

2.1 Cave Environment

In this paper we focus on Martian caves associated with lava tubes. Due to Mars' lower gravity, Martian lava tubes are much larger than Earth lava tubes. Herein we target caves that are approximately 100 meters wide and potentially hundreds of meters deep, with a skylight entrance formed from a collapsed cave ceiling as illustrated in Figure 1. We assume that the terrain in the interior of the cave is unknown a priori.

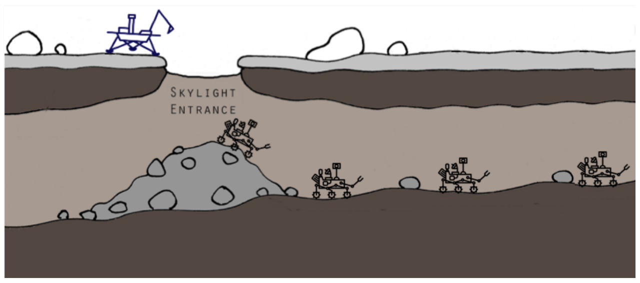


Figure 1: Illustration of a hypothetical multi-rover coordination problem in Mars cave exploration. Rovers not to scale. Credit: Figure adapted from the Wikimedia Commons, Longitudinal cross-section of a martian lava tube with skylight.

Cave walls are a quite interesting science target for NASA/JPL scientists. They can provide critical constraints on lava temperature and cooling history, leading to insights into Martian magmatic processes and differentiation. Thus, in our coordination problem the rovers should try to safely remain as close to the walls as possible to characterize wall properties and facets.

2.2 Conceptual Autonomous Rovers

We consider a set of homogenous rovers that are assumed to be successfully deployed at the bottom of the cave through the skylight entrance. The problem of deploying the rovers into the cave, although interesting, is not in the scope of this work. The focus is in the exploration and coordination problem while in the cave where communication is limited.

The rovers are equipped with a battery module, mobility components, a communication component (antennas), and a science component with a set of key science instruments. Those components allow each rover to perform the following actions:

- **Ping** (communication component): a rover can send a ping to all rovers within communication range to detect the vehicles around it. Rovers (including the base rover) in the communication range respond with their position and status update. A ping process has a specific duration (e.g., 2 seconds) and also a power consumption rate known a priori. Communication range and ping duration are provided in the antenna specs.
- **Drive** (mobility components): to navigate the environment safely, each rover is able to detect obstacles within a radius (e.g. 5 meters) in 360 degrees. The cave map is stored during exploration - given that the focus of this work is not on mapping and localization per se, we assume that the knowledge about the map and coverage becomes available to all the rovers as they explore the cave. The velocity of the vehicle and power consumption during driving is known and given by the mobility specification.
- **Science** (science component): each rover has the same set of science instruments partitioned in three categories: *primary instruments*, *secondary instruments*, and *periodic*

instruments. Each one of these instruments has its own specs for power consumption, data volume generated by each reading and the sensing duration.

- **Transfer** (communication component): transfer is a collaboration task in which the sender first sends a transfer request to a target/receiver rover. The receiver then informs the sender when it is available to receive data. Once that confirmation is received, the sender transfers the data to the target rover. When the data is successfully transferred, the receiving rover sends a confirmation and the task is completed. The duration of the actual data transfer between two rovers (lander and/or science rover) is determined by the antenna specs, the data volume and the distance from each other (bandwidth). The bandwidth can be modeled with an arbitrary function (see Discussion). For the simulations presented in this paper, we model bandwidth as a step-wise function of distance between communicating vehicles. For example between 0-5 meters rovers can transfer data at 11.0 Mbps, between 5-10 meters at 5.5 Mbps, between 10-15 meters at 2.0 Mbps, and between 15-25 meters at 1.0 Mbps. Power consumption rates are also known and are constant during communication, regardless of distance.

In additional to the above action specification, we list below some of the key assumptions on the exploration problem:

1. All actions consume energy from the battery component, which is a limited resource. If the battery drains out, the rover becomes non-operational.
2. We consider a constant hotel load that represents the energy consumption to keep the rover operational. We frame any cognitive process (e.g., decision making, path planning computation) as part of this constant consumption.
3. Each rover can only execute one action at a time, except sending and responding to pings. In the science case, only one instrument can be used at a time.
4. Communication model does not consider the shape, texture, material of the cave or proximity to walls. (This is actually already being incorporated in our models, but will be left for future publications.)
5. Communication is possible only up to a fixed distance between rovers, where the lander has a longer fixed range.
6. Rover can fail during exploration, which means that the coordination has to account for reconfiguration.
7. In this work we are not modeling acceleration or slippage in the motion model.
8. Finally, each rover does have a memory component for data science storage, but the memory capacity is large enough to handle days or weeks worth of data.

3. Approach

We propose a multi-rover coordination strategy for cave exploration that aims to send rovers as deep into the cave as possible while also maximizing data sent out to a surface base station.

The rovers explore the cave using the *Dynamic Zonal Relay with Sneakernet Relay Algorithm*, which is a two phase algorithm, starting with (1) *Dynamic Zonal Relay* and expanding with (2) *Sneakernet Relay*. One of the main aspects of this algorithm is the use of spatial zones to determine the state of the rover. Each zone is a distinct section of the cave based on distance from the lander, as shown in Figure 2.

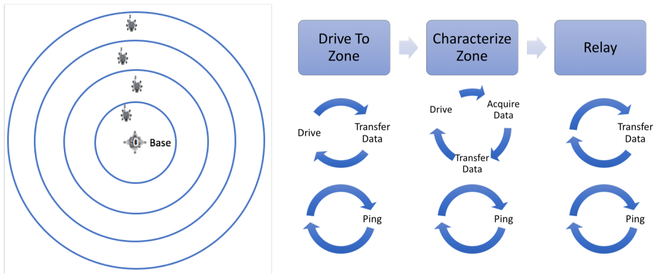


Figure 2: Zones based on the distance from the conceptual lander or base station (left). Nominal state transitions for the Dynamic Zonal Relay phase (right).

3.1 Dynamic Zonal Relay

The first phase, *Dynamic Zonal Relay*, assigns the rovers to designated, adjacent zones that keep the rovers within communication range of their immediate neighbors. The algorithm is *dynamic* in that if any rover becomes inactive (i.e., no longer communicating due to some kind of failure or running out of battery), the other rovers dynamically readjusts the zone assignments.

While *driving* to its assigned zone, the rover maintains a safe communication distance with its neighboring rovers and relays any science data that has been transferred to it to its neighbor in the direction of the lander. When in its zone, the rover moves along the length of the cave, continuing to maintain communication distance, while *characterizing* the cave. The rover sends acquired science data to the neighboring rover closest to the lander. Once at the end of its zone, the rover becomes a *relay* point. In this state, the rover transfers any remaining science data that it has collected, as well as any science data that has been transferred to it, to its neighbor closest to the lander.

A diagram of the nominal state transitions for the Dynamic Zonal Relay phase is shown in Figure 2. The diagram also shows that the rovers perform periodic pings to the other rovers to share status information, such as position, and to keep track of which rovers are still active.

In the case that a rover becomes inactive, the surrounding rovers readjust depending on their position relative to the inactive rover. Rovers closer to the lander would not need to adjust their zones; however, they need to be made aware of the new configuration. Rovers deeper into the cave need to adjust their zone closer to the other rovers to re-establish a continuous line of communication across all rovers. Since the rovers do not know how much science data the inactive rover was able to acquire and transfer out of its zone (if it was already characterizing its zone), all rovers that moves into a new zone re-characterize the entire zone.

3.2 Sneakernet Relay

Once all of the data that was collected during the Dynamic Zonal phase has been passed to the lander, the rovers transition to the *Sneakernet Relay* phase. During this phase, the rover furthest into the cave is designated as the lead rover (e.g., *Rover4* in a team of four rovers) and the others are designated as relayers (e.g., *Rover1*, *Rover2* and *Rover3* in the team of four rovers). The lead rover is now tasked with characterizing the next zone, which means that one of the relayers is no longer in communication range of one of its neighbors, meaning that it must *sneakernet*. Increased sneakernet distance is assigned in order, starting with the rover closest to the lander (e.g., *Rover1* in our example), as the lead rover characterizes more zones.

The sneakernetting process is composed of cycles, where a sneakernet cycle consists of each rover incrementally increasing its sneakernetting distance. A cycle is further broken down into stages that are repeated with each assignment of increased distance: (1) extension/replacement and characterization, (2) relay, and (3) confirmation. Except at the beginning of the Sneakernet Relay phase, stage (1) and stage (3) occur simultaneously. Figure 3 helps to illustrate the evolution of the Sneakernet Relay phase, with line 1 showing the positions of the rovers for a three rover mission configuration at the end of the Dynamic Zonal Relay phase.

The beginning of a cycle is triggered by the rover closest to the lander (*Rover1*) beginning the *extension/replacement and characterization* stage, as shown in Figure 3 line 2. The initiator of this stage (in this case, the rover closest to the lander) moves forward to the relay position of its neighbor. This triggers the neighbor rover to move forward to the relay position of the rover in front of it, and so on, until the lead rover. When the lead rover is triggered, it moves forward and characterizes the next section of the cave, which is the same distance as that of the relay distance of the previous rover (the distance between the leader's neighbor and the neighbor's neighbor).

When the lead rover has finished collecting new data (line 3), the *relay* stage is initiated. The lead rover begins by moving within communication range of the rover following it and transferring all of its data. After the transfer, the transferring rover remains where it is while the receiving rover moves to communication range of its neighbor in the direction of the lander and transfers all of the data, and so on, until the rover closest to the lander transfers all of the data out of the cave. In Figure 3, line 4 shows the first rover requiring to move in order to transfer the data to the lander.

The transfer of all of the data to the lander triggers the next stage, *confirmation*. The rover closest to the lander now moves back to its neighbor inside the cave, confirms that the transfer was successful, and returns to its previous relay position (line 5). The next rover then moves deeper into the cave to its neighbor and reports the confirmation and returns to its relay position, and so on for all of the rovers until the confirmation reaches the lead rover. During this stage, the next rover to initiate extension moves to its next relay position during the confirmation process, triggering all rovers to move deeper into the cave as a cascading sequence of extension and confirmation, such as on line 6. In line 7, we see the lead rover moving ahead and characterizing a zone the same distance as that of the relay distance of the previous rover (distance between *Rover2* and *Rover1*, as described previously, requiring the lead rover to sneakernet on line 8.

The remainder of Figure 3 shows the repetition of these stages, until line 13, which shows the positions of the rovers after the second cycle is initiated by the first rover (*Rover1*).

To remain robust to rover failures during the Sneakernet Relay phase when the rovers are no longer in communication range, the rovers rely on timeouts to estimate how long they should wait for a neighbor to initiate the next phase. If a timeout is reached, they will try to find its neighbor in the direction of the lander to re-establish the relay chain. If a relayer reaches a timeout waiting for its peer deeper in the cave, it will then act as the leader.

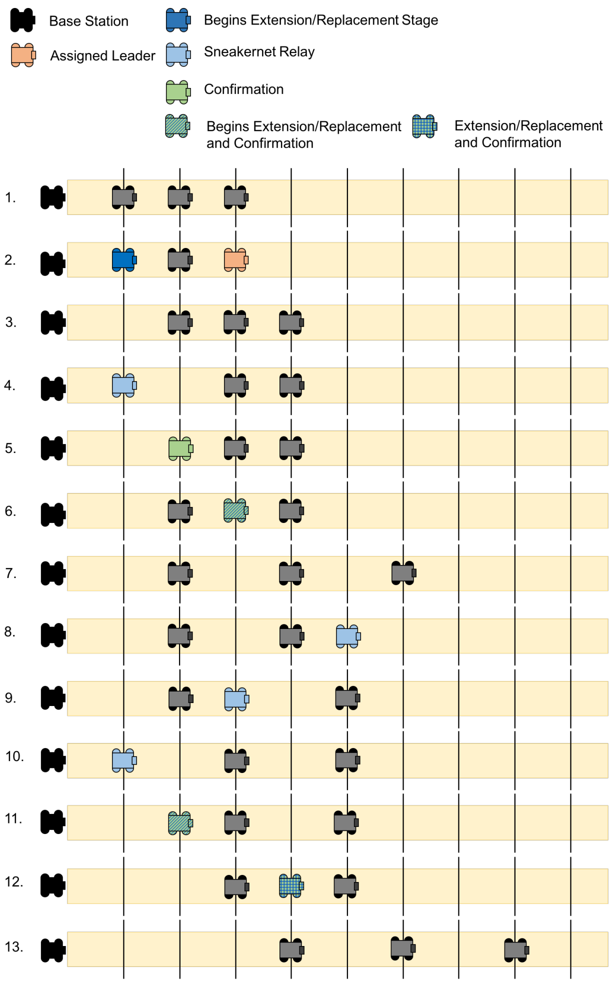

Base Station
Begins Extension/Replacement Stage
Assigned Leader
Sneakernet Relay
Confirmation
Begins Extension/Replacement and Confirmation
Extension/Replacement and Confirmation
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.

Figure 3: The movement of the rovers during the Sneakernet Relay phase of the algorithm for the first cycle.

4. Experiments in Simulation

The Dynamic Zonal Relay with Sneakernet Relay Algorithm was tested in a simulation framework using the Robot Operating System (ROS) to model the communication between the rovers as well as to model the different rover components (e.g. the science instruments, driving and navigation, etc.) and the cave.

A configuration with four rovers (*Rover1* through *Rover4*) and a base station was used, as illustrated in Figure 1, for the experiments. This configuration is based on preliminary cost and payload analysis of similar classes of missions. The cave model used is a model of the Cassone Cave (Santagata), scaled approximately twelve times so that the width is around 70 m, which is shown in Figure 4. The cave model is made up of approximately 350,000 triangular facets, with an average size 1.16 m^2.

Each rover is assumed to have an identical suite of instruments, partitioned in the three aforementioned categories: primary, consisting of a LiDAR to characterize the walls, facets and structure of the cave; secondary, including a color camera and a spectrometer; and periodic instruments, including a thermometer, radiation detector, and hygrometer. Primary and secondary instruments are used based on movement of the rover, whereas the periodic instruments are used based on a regular, timed cadence (in this case, every 60 minutes). A summary of the assumed instrument parameters is shown in Table 1.

Table 1: Assumed Instrument parameters

	Power (Watts)	Data Volume (Mb)	Sensing Duration (s)
LiDAR	10.0	1344	5.0
Color Camera	5.0	150.0	1.0
Spectrometer	10.0	14.4	660.0
Thermometer	1.0	0.0008	5.0
Radiation Detector	1.0	0.0008	5.0
Hygrometer	2.0	0.0008	5.0

The communication range was limited to 25 meters between rovers and 75 meters to the lander. Pings to communicate position and status were performed once per minute. The assumed power for communication was 4.0 Watts.

In this work we incorporate a simple approach for rover navigation and selection of the region to be explored in the cave. Each rover computes its path through the cave map using the A* algorithm, where *Rover1* through *Rover3* move toward the rovers ahead of them, while the leader, *Rover4*, uses a frontier detection algorithm (Yamauchi 1997) to move towards unexplored regions of the map into the cave. In this experiment, rovers traverse the environment at 0.005 m/s and a 5-meter range is used for obstacle detection and mapping. It is assumed that driving requires 14.0 Watts of power.

We also model a hotel load (the amount of power required for a rover to remain operational, such as basic heating and health monitoring) of 5.0 Watts.

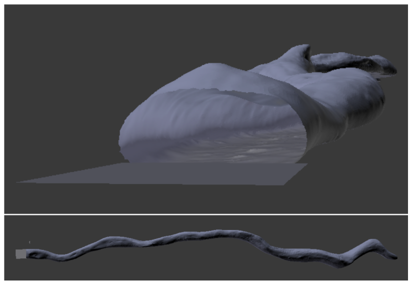


Figure 4: Simulated cave front and top views. Model of the Cassone Cave (Santagata), scaled approximately twelve times.

To test a perfect scenario where the cave has no obstacles and the rovers are able to function until they run out of energy, one experiment was performed with zero obstacles in the cave and no random dying of the rovers. Two further experiments were performed, again with rovers able to function until they run out of energy, with random obstacle densities of 10% and 20% to show how the simulation can evaluate the success of a mission where there are rocks and debris throughout the cave. In order to show the robustness of the algorithm to loss of rovers, another experiment was run with zero obstacles and a random chance of rovers dying during the run.

As a comparison, an experiment was performed with a single rover using the Dynamic Zonal Relay with Sneakernet Relay algorithm, where the rover extends by a single zone (in this case 20 m) at each step. No obstacles were used for this experiment.

To evaluate the different runs, a scoring function based on the cave characterization data transferred out of the cave was used. For remote instruments (such as cameras), the score, s_{remote}, is the area of the triangular facets covered in the cave model based on the position of the rover, the field of view of the instrument, and any restrictions on far and near clip planes or normal angle of the facet, which is summarized in Eq. 1 for a data acquisition instance data$_i$.

$$s_{remote}(\text{data}_i) = \sum_{\text{cave facets, } f} \text{area}(f), \text{ if } f \text{ visible} \quad (1)$$

For in-situ instruments (e.g. temperature sensors), the score, $s_{in-situ}$, depends on both position and time of the data. For these types of measurements, the value of the data decreases if it is taken at almost the same position and time, therefore the score is a function that decays based on the position and time of any previously taken data of that type. Given a max time of T before a facet can receive a full score again, $s_{in-situ}$ is determined by Eq. 2 for a data acquisition instance data$_i$.

$$s_{in-situ}(\text{data}_i) = \sum_{\text{cave facets, } f} \text{area}(f) * d, \text{ if } f \text{ visible,} \quad (2)$$

where,

$$d = \begin{cases} 1 & \text{if } f \text{ last measured } > T \text{ seconds ago} \\ e^{-(T-\Delta t)/T} & \text{otherwise} \end{cases}$$

and visibility is based on a sphere with a fixed radius instead of a field of view and clip planes.

This results in a total score, $score$, defined by Eq. 3, where only data that is transferred out of the cave is scored.

$$\begin{aligned} score = & \sum_{\text{remote data acquisitions, } i} s_{remote}(\text{data}_i) \\ + & \sum_{\text{in-situ data acquisitions, } j} s_{in-situ}(\text{data}_j) \end{aligned} \quad (3)$$

5. Simulation Results

In what follows we present the results from the single rover and the multi-rover experiments using the simulator. A comparison of the results are shown in Table 2.

5.1 Single Rover

The single rover was able to explore up to 100 meters into the cave; however it was only able to transfer data from up to 80 meters. This can be seen from Figure 5, which shows the depth into the cave that the rover travelled with respect to time; the rover was not able to make it back close enough to the lander after characterizing up to 100 meters to transfer its most recently collected data (i.e., data collected between 80-100 meters).

The percentage of time that the rover spent performing different activities is show in Figure 6, which demonstrates that the amount of time required to drive and transfer the data is quite significant, especially with respect to the amount of

Table 2: Comparison of simulated experiments

	Max Lifetime (days)	Max Transferred Data Distance (m)	Score	Data Volume Transferred (GB)	Data Volume Un-Transferred (GB)	Rover Death	Death Time (days)
Single Rover	1.59	80	1122.76	4.70	1.39		
0% Obstacles	2.99	100	3828.27	6.44	1.02		
10% Obstacles	3.00	100	4285.45	6.76	1.39		
20% Obstacles	3.41	45	3347.68	4.34	0.0000077		
Random Death	2.69	75	2452.43	5.20	2.41	Rover4, Rover1	0.19, 1.94

time spent acquiring the data (labeled "Science"). However, when looking at the percentage of energy required, Figure 7, the transfers make less of an impact, whereas driving continues to have the greatest impact.

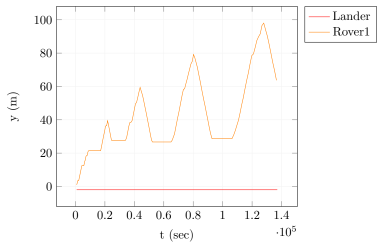


Figure 5: Simulated depth of the rover into the cave (y position) with respect to time.

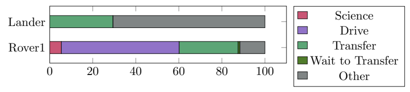


Figure 6: Percentage of time spent on different activities.

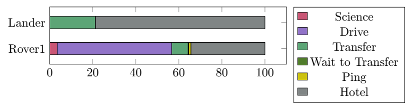


Figure 7: Percentage of energy required to perform different activities.

5.2 Four Rovers

Figure 8 shows an example of the motion of the rovers expanding and sneakernetting in the four rover configuration with zero obstacles and no random death.

The percent breakdown of activities in terms of time is displayed in Figure 9, where it is shown that transferring data (either receiving or sending) takes a large portion of a rover's time, increasing for the rovers closer to the lander. In fact, comparing Figure 6 and Figure 9, *Rover1* spends approximately as much time transferring as the single rover. However, the largest amount of time is spent performing "other" activities, which includes pings and idle time. Unlike in the single rover scenario, with multiple rovers there are times that the state transition of a rover depends on the actions and states of the surrounding rovers, meaning that the rover must wait idly, which is why the *Other* time is so high in Figure 9 compared to Figure 6.

Figure 10 shows the energy distribution for the four rovers. Like the single rover scenario, in terms of specific activities, driving takes the most amount of energy. However, in the multi-rover scenario, each rover spends much less energy performing science activities, with *Rover4* using the most energy on science, as expected since it characterized more zones. Although a small percentage of the overall energy required by the rovers, in Figure 10, it can be seen that it is not an insignificant source of energy usage.

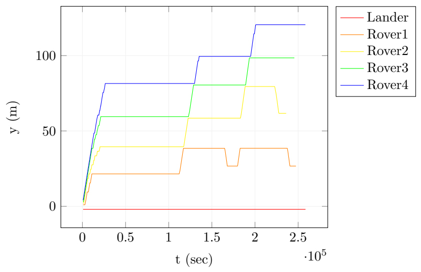


Figure 8: Simulated depth of the rovers into the cave (y position) with respect to time for a four rover sneakernet configuration with zero obstacles and no chance of random death.

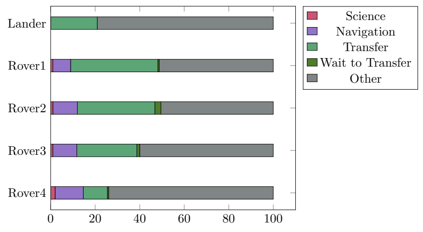


Figure 9: Percentage of time spent on different activities for the four rovers and the lander in the simulation.

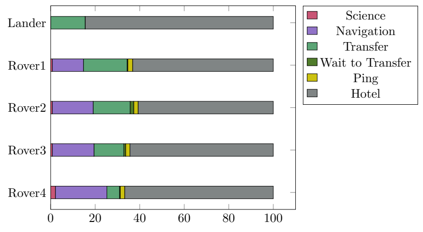


Figure 10: Percentage of energy required to perform different activities for the four rovers and the lander in the simulation.

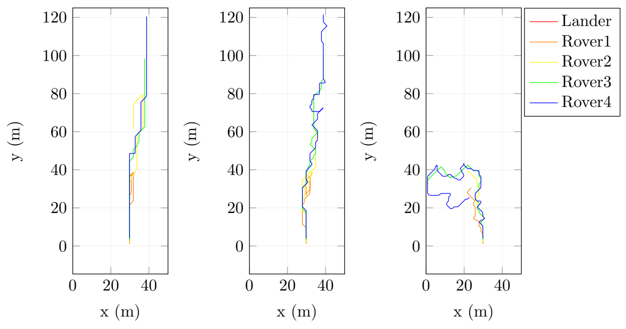


Figure 11: Paths of the rovers with 0% obstacle density (left), 10% obstacle density (center) and 20% obstacle density (right) with no random rover death.

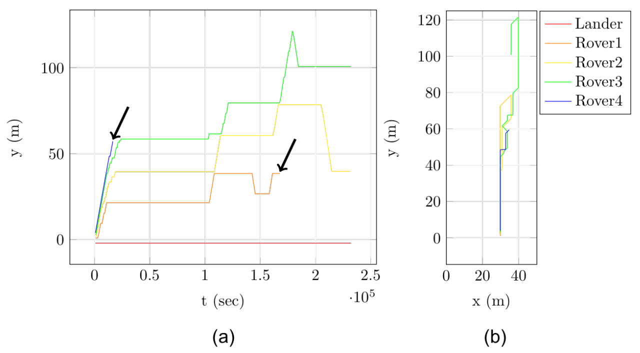


Figure 12: a) Simulated depth of the rover into the cave (y position) with respect to time. The arrows point out times and locations of rover deaths. b) Paths of the rovers in the random dying rovers experiment.

With more obstacles, the paths of the rovers, shown in Figure 11 for 0%, 10%, and 20% obstacle densities and no random death, become less straight and aligned with the cave wall. In fact, with 20% obstacle density, the rovers are not able to find a path that stretches beyond 45 meters into the cave, and the rovers begin exploring farther from the cave wall. This means that *Rover4* (the leader) never reaches its zone and does not take any primary or secondary data.

The experiment with random rover death shows a scenario where *Rover4* dies just before *Rover3*, its immediate follower, finishes characterizing its zone, and *Rover1* dies after the first transfer of the first sneakernet expansion data. From Figure 12 (a), we see that *Rover3* seemlessly becomes the new leader and leads the way during the algorithm's Sneakernet Relay phase. As shown in Figure 3, *Rover3* expands by its neighbors relay distance.

We also see a timeout begin in Figure 12 with *Rover2*. When *Rover2* sneakernets back toward *Rover1* to relay the data, it is not able to locate *Rover1*, therefore *Rover2* waits at the location it last saw *Rover1* for a timeout duration (which in this scenario, *Rover2* does not live long enough to finish). Although there are only three rovers, they are able to collect data beyond 100 meters, but are only able to live long enough to transfer out data up to 75 meters into the cave.

Figure 12 (b) shows the paths of the rovers for the experiment with random rover deaths, which looks very much like the 0% obstacle paths in Figure 11, as expected.

6. Discussion

The simulation shows that a single rover can successfully characterize up to 80 meters along a cave wall (given no obstacles) if it does not encounter any problems before running out of battery; however, this is a large assumption given the unknown environment of the cave. The sneakernet results with randomly dying rovers shows the robustness of the Dynamic Zonal Relay with Sneakernet Relay algorithm to rover loss. Furthermore, with the survival of all rovers for the duration of the battery charge, science data from deeper into the cave can be transferred out to the lander than in the single rover case.

Although the experiment with 20% obstacle density showed the rovers unable to reach as deep into the cave as other scenarios, it is interesting to note the large score. This is due to the fact that as the rovers move farther away from the cave wall, the field of view of the remote instruments is able to capture a larger section of the cave model facets. This shows an interesting trade-off that can be made between remaining close to the wall, and therefore characterizing smaller features of the cave and reaching deeper distances, versus moving away from the wall and characterizing larger sections.

In our exploration approach, the sequencing of science actions is predefined based on scientist team inputs. Nevertheless, an automated and opportunistic sequencing of science actions could provide a higher science utility. Onboard data analysis and science goals and instrument prioritization techniques are described in (Chien et al. 2016). Castano et al. (2007) describes the Onboard Autonomous Science Investigation System (OASIS), an autonomous system that is capable of analyzing imagery to generate new science tasks for execution both in simulation and on a test rover. Wettergreen et al. (2014) shows the capability of autonomous sample location selection and adaptive path planning on a rover in a deployment to the Atacama Desert. Woods et al. (2009) demonstrates the feasability of autonomous opportunistic science with autonomous instrument placement for contact science. All of these algorithms and techniques would support desirable autonomous behavior. Moreover, our proposed approach has room for improvement with respect to the rovers responsible for relaying data. More opportunistic decision making approaches would allow relayers to potentially perform additional science tasks while also managing the task of relaying data out of the cave.

Coordination of data transfer is also an opportunity for cognitive systems. As opposed to waiting for a target rover to be available to receive data, a scheduling system could support a more efficient data transfer coordination between rovers (Clement and Barrett 2003) - assuming they can share their status and activities. The communication model and respective ranges have a great impact on this coordination. We are working on incorporating a stochastic communication model in which bandwidth degrades as a function of distance and does not have a hard constraint on the maximum distance (e.g., 25-meter max range). That provides opportunities for rovers to establish a comm link in greater distance and provides options to route data science out of the cave through different rovers. A more realistic package management during communication would make the routing problem even more interesting, in which science data could be partitioned into smaller pieces and sent to different rovers over time depending on bandwidth variations. Here we assume that data packages would be able to be properly combined at the target asset (e.g., lander or an orbiter). Such stochastic models would also create scenarios in which rovers are physically close but with a poor or unexisting communication link.

7. Conclusion

In this paper we proposed a multi-rover coordination algorithm for Mars Cave exploration. A simulation framework was created to evaluate the performance of the algorithm and to study mission configurations to explore design options for future missions to underground cavities in other planets and moons. We utilized realistic cave settings and vehicle specs to generate an initial evaluation of the feasibility of the multi-rover approach for science data collection. We also discussed opportunities for AI planning and scheduling techniques to augment rover autonomy and efficiency with respect to science utility.

This is an ongoing research project with several promising immediate next steps and future directions. In the short-term we will investigate the impact on rover performance when increasing action concurrency. More specifically, in the simulation we will allow rovers to transfer data while navigating the environment and doing science. We will also incorporate data routing techniques with the aforementioned stochastic communication model we are integrating. We are also interested in augmenting the proposed algorithm to help rovers to better coordinate data transfer and to balance data relay and science tasks.

8. Acknowledgments

Portions of this work were performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. We thank the valuable inputs from other JPL team members, including Jay Wyatt, Joseph Lazio, Jay Gao, Abigail Freaman, Julie Castillo-Rogez, Sebastian Herzig, Konstantin Belov, and Amos Byon.

References

auf der Heide, F. M., and Schneider, B. 2008. Local strategies for connecting stations by small robotic networks. In Hinchey, M.; Pagnoni, A.; Rammig, F. J.; and Schmeck, H., eds., *Biologically-Inspired Collaborative Computing*, 95–104. Boston, MA: Springer US.

Boston, P.; Frederick, R.; Welch, S.; Werker, J.; Meyer, T.; Sprungman, B.; Hildreth-Werker, V.; Thompson, L.; and Murphy, D. 2003. Human utilization of subsurface extraterrestrial environments. *Gravitational and Space Biology Bulletin* 26(2).

Boston, P.; Frederick, G.; Welch, S.; Werker, J.; Meyer, T.; Sprungman, B.; Hildreth-Werker, V.; Murphy, D.; and Thompson, S. 2004. System Feasibility Demonstrations of Caves and Subsurface Constructed for Mars Habitation and Scientific Exploration. Technical report, USRA Reports, NASA Institute for Advanced Concepts.

Boston, P.; Spilde, M.; Northup, D.; Melim, L.; Soroka, D.; Kleina, L.; Lavoie, K.; Hose, L.; Mallory, L.; Dahm, C.; Crossey, L.; and Schelble, R. 2005. Cave biosignature suites: microbes, minerals, and mars. *Astrobiology* 1(1):25–55.

Castano, R.; Estlin, T.; Anderson, R. C.; Gaines, D. M.; Castano, A.; Bornstein, B.; Chouinard, C.; and Judd, M. 2007. Oasis: Onboard autonomous science investigation system for opportunistic rover science. *Journal of Field Robotics* 24(5):379–397.

Chien, S.; Barrett, A.; Estlin, T.; and Rabideau, G. 2000. A comparison of coordinated planning methods for cooperating rovers. In *International Conference on Autonomous Agents (Agents 2000).*

Chien, S.; Thompson, D. R.; Castillo-Rogez, J.; Rabideau, G.; Bue, B.; Knight, R.; Schaffer, S.; Huffman, W.; and Wagstaff, K. L. 2016. Agile science - a new paradigm for missions and flight software.

Clement, B. J., and Barrett, A. C. 2003. Continual coordination through shared activities. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, 57–64. New York, NY, USA: ACM.

Clement, B.; Durfee, E.; and Barrett, A. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research* 28:453–515.

Dubowsky, S.; Iagnemma, K.; Liberatore, S.; Lambeth, D. M.; Plante, J. S.; and Boston, P. J. 2005. A concept mission: Microbots for largescale planetary surface and subsurface exploration. *AIP Conference Proceedings* 746(1):1449–1458.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016. Productivity challenges for mars rover operations. In *Proceedings of 4th Workshop on Planning and Robotics (PlanRob)*, 115–125. London, UK.

Husain, A.; Jones, H.; Kannan, B.; Wong, U.; Pimentel, T.; Tang, S.; Daftry, S.; Huber, S.; and Whittaker, W. L. 2013. Mapping planetary caves with an autonomous, heterogeneous robot team. In *IEEE Aerospace Conference*, 1–13.

Kesner, S. B.; Plante, J. S.; Boston, P. J.; Fabian, T.; and Dubowsky, S. 2007. Mobility and power feasibility of a microbot team system for extraterrestrial cave exploration. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 4893–4898.

Santagata, T. Università degli Studi di Modena e Reggio Emilia. Inside the Glacier Project.

Stump, E.; Jadbabaie, A.; and Kumar, V. 2008. Connectivity management in mobile robot teams. In *ICRA*, 1525–1530. IEEE.

Thangavelautham, J.; Robinson, M. S.; Taits, A.; McKinney, T.; Amidan, S.; and Polak, A. 2014. Flying, hopping pit-bots for cave and lava tube exploration on the moon and mars. In *The 2nd International Workshop on Instrumentation for Planetary Missions*.

Wettergreen, D.; Foil, G.; Furlong, M.; and Thompson, D. R. 2014. Science autonomy for rover subsurface exploration of the atacama desert. *AI Magazine* 35(4):47–60.

Woods, M.; Shaw, A.; Barnes, D.; Price, D.; Long, D.; and Pullan, D. 2009. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics* 26(4):358–390.

Yamauchi, B. 1997. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA'97*, 146–151.

Yliniemi, L.; Agogino, A.; and Tumer, K. 2014. Multi-robot coordination for space exploration. *AI Magazine* 35(4):61–74.

DMRR: Dynamic Multi-Robot Routing for Evolving Missions

Mihai-Ioan Popescu, Olivier Simonin, Fabrice Valois
INSA Lyon, CITI-INRIA Laboratory
University of Lyon
F-69621 Villeurbanne, France
{mihai-ioan.popescu, olivier.simonin, fabrice.valois}@insa-lyon.fr

Gabriela Czibula
Department of Computer Science
Babeş-Bolyai University
Cluj-Napoca, Romania
gabis@cs.ubbcluj.ro

Anne Spalanzani
INRIA Rhône-Alpes
University of Grenoble
Grenoble, France
anne.spalanzani@inria.fr

Abstract

The paper proposes Dynamic Multi Robot-Routing (DMRR), as a continuous adaptation of the multi-robot target allocation process (MRTA) to new discovered targets. There are few works addressing dynamic target allocation. Existing methods are lacking the continuous integration of new targets, handling its progressive effects, but also lacking dynamicity support (e.g. parallel allocations, participation of new robots). The present paper proposes a framework for dynamically adapting the existing robot missions to new discovered targets. Missions accumulate targets continuously, so the case of a saturation bound for the mission costs is also considered. Dynamic saturation-based auctioning (DSAT) is proposed for allocating targets, providing lower time complexities (due to parallelism in allocation). Comparison is made with algorithms ranging from greedy to auction-based methods with provable sub-optimality. The algorithms are tested on exhaustive sets of inputs, with random configurations of targets (for DMRR with and without a mission saturation bound). The results for DSAT show that it outperforms state-of-the-art methods, like standard sequential single-item auctioning (SSI) or SSI with regret clearing.

1 Introduction

As robot missions have become reality, a growing number of applications requires teams of mobile robots to autonomously accomplish missions incorporating task groups. *Multi-robot Task Allocation* (MRTA) has been widely studied in collaborative multi-robot planning, and generally for multi-agent coordination. Key applications include search and rescue operations (Wei, Hindriks, and Jonker 2016; Beck et al. 2016), multi-robot exploration (Tovey et al. 2005) or patrolling (Pippin, Christensen, and Weiss 2013). As underlined by an extensive MRTA survey of 2015 (Khamis, Hussein, and Elmogy 2015), despite the large number of papers in the task allocation domain, there is only few work concerning subjects such as dynamic task allocation or allocation of complex tasks in multi-robot systems.

In practice, targets are found progressively and robots carry missions that evolve in time. Therefore, a recent challenging problem is how to dynamically and efficiently adapt the robots and their work to new targets, in order to let robots continue or extend their missions (while keeping the envisaged objectives).

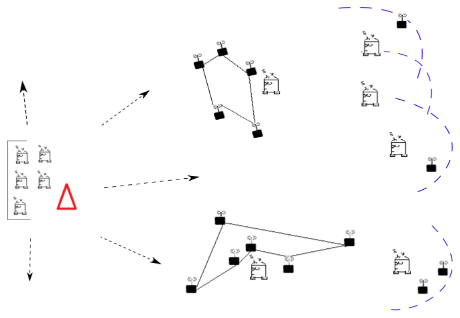


Figure 1: Robots deployed for executing missions on targets and for progressive target discovery. Targets are represented by the black dots and missions by the cyclic paths. Robots move on these paths between their allocated targets.

Multi-robot routing (MRR) (Lagoudakis et al. 2005) has often been the main testbed for MRTA scenarios, being at base part of the location routing problems (Toth and Vigo 2014). The MRR problem assumes a set of static targets and robots with known locations. The goal is then to assign the targets to the robots while optimizing an objective function. Under typically considered objectives, MRR is a difficult combinatorial optimization problem – NP-hard (Lagoudakis et al. 2005) – therefore heuristics have been studied in order to provide solutions, though not optimal, but in feasible time. For the rest of the paper, as in MRTA literature, target allocation and task allocation are used interchangeably.

Solving MRR problems has been addressed using market-based approaches like combinatorial, parallel or sequential auctions (Koenig et al. 2006; Koenig, Keskinocak, and Tovey 2010), but also through other optimization based techniques, mainly TSP-based optimizations (Mosteo, Montano, and Lagoudakis 2009; 2008). Lately, approaches were proposed for combining auctions with methods such as clustering (Heap and Pagnucco 2011; Elango, Nachiappan, and Tiwari 2011) or matroid theory techniques (Williams, Gasparri, and Ulivi 2017). Aspects of dynamic task allocation have been envisaged by few works. These include the consideration of dynamic clusters in auctions (Heap and Pagnucco 2012), the re-auctioning of uninitiated tasks (Nanjanath and Gini 2010) (both presuming delays, e.g. caused by communication loss or dynamic obstacles), or more recently, prediction methods for search and rescue operations (Wei, Hindriks, and Jonker 2016). However, these approaches are lacking support for dynamic target allocation (like the use of parallel allocations). The algorithms require numerous allocation rounds to finish, trading off time complexities for solution quality. This does not cope with dynamic allocation needs, like repeated algorithm execution. Also, existing methods do not handle effects of continuous growths of the sets of targets (e.g. a robot may have its missions saturated, due to resource constraints).

This paper proposes *Dynamic Multi-Robot Routing* (DMRR) as an incremental dynamic adaptation of the MRR process to new targets. DMRR assumes that targets are continuously discovered by a group of exploration robots, and then integrated in the missions of the working robots (see Figure 1). Hence, robot missions continuously grow in size. No prior information is considered regarding target locations, since the environment is presumed to be dynamic, with a high degree of uncertainty.

After presenting the background and related work (Section 2), a framework is proposed around DMRR, which formalizes the problem, contains a mathematical model and complexity proofs (Section 3). The MRR objectives are redefined for the dynamic case, while asymptotic objectives and target coverage maximization are envisaged as well (DMRR remains NP-hard, under the MRR objectives). Because of the continuous mission growth, missions may get saturated, so resource limitation is taken into account in a version of DMRR, called DMRR-Sat. This problem assumes a so-called mission saturation bound (i.e. a bound for the mission cost). Consequently, proofs for DMRR-Sat hardness are provided.

Finally, Section 4 proposes *dynamic saturation-based auctioning* (DSAT) for allocation of the new discovered targets at each time step. DSAT encapsulates an auctioning algorithm introduced here as *inverse-SSI*. It combines elements of parallel and sequential auctions, providing better time complexities than state-of-the-art solutions. The algorithm is described along with its complexity analysis and experiments compare the DSAT and inverse-SSI with state-of-the-art auction algorithms such as standard single-item auctions (SSI (Lagoudakis et al. 2005)) or SSI with regret clearing (Zheng et al. 2008). Computations are performed for an extensive set of scenarios (Section 5). We discuss empirical results, showing that DSAT and inverse-SSI perform better than existing state-of-the-art methods.

2 Background and Related Work

Multi-robot routing considers a set of mobile robots R and static targets T, whose locations are known in the two-dimensional plane. A cost is considered for traveling between two locations on the map and all robots can communicate between them without errors.

Definition 2.1 (MRR). *The multi-robot routing problem (Lagoudakis et al. 2005) consists in finding allocations of targets T to robots R (and paths to visit these targets), such that a team objective function is optimized.*

MRR has been the standard testbed for multi-robot task allocation problems. Originally part of the vehicle routing problems, VRP (Pillac et al. 2013), MRR differs from VRP on at least two fundamental things. First, it does not rely on a predefined path graph (but considers movement between any two locations in the area). Second, for the dynamic case, targets might not be discarded after one visit, but remain part of the robot mission. Hence, the problem objective is dependent on the whole mission, not only on the robot's current position. Throughout this paper, like in usual MRTA terminology, task allocation is equally referred as target allocation (in literature, this often depends on the context). Task is used more to emphasize the action (which may have an ending time), and it can be seen as a treatment of a target.

Taxonomies regarding MRTA (Gerkey and Matarić 2004; Korsah, Stentz, and Dias 2013) have been widely used to classify task allocation problems. Thereby, (Gerkey and Matarić 2004) classifies MRTA problems into categories: single-task (ST) or multi-task (MT) robots, single-robot (SR) or multi-robot (MR) tasks, instantaneous assignment (IA), or time-extended assignment (TA) (w.r.t. the target allocation). MRR, as well as dynamic MRR, proposed in this paper, are part of the MT-SR taxonomy, since a task can be assigned to only one robot at a time. However, MRR has often been treated in IA scenarios, whereas DMRR is rather TA, since the process is dynamic and communication is limited, so the assignation of targets is done in time.

A more recent taxonomy (Korsah, Stentz, and Dias 2013) classifies MRTA problems also based on the dependencies between tasks or robots: no-dependencies (ND), inter-schedule dependencies (ID), cross dependencies (XD) and complex dependencies (CD). DMRR falls into the ID class, along with other well-known NP-hard problems like m-TSP[1] or MRR, since the cost of treating a target (e.g. time, travel distance) depends on the other targets treated before.

To our knowledge, so far no work considered target allocation for ongoing evolving missions, nor offered a framework which correlates the target discovery and the dynamic adaptation to these new targets. Paper (Tardioli et al. 2010) offers a framework treating three problems simultaneously: multi-task allocation, cooperative navigation while maintaining the multi-hop robot network connected, and ensuring link quality for real-time communication. However, this considers robot clusters may be assigned to one task (i.e. MR tasks), and the task allocation plan is performed w.r.t. the connectivity maintenance constraint.

In the last years, MRTA techniques have been used for online planning in scenarios such as warehouse commissioning problems (Claes et al. 2017) or search and rescue operations with uncertainty of tasks (introduced as UMRTA) (Beck et al. 2016), where the information about the tasks relies on probability distributions.

Solving MRTA problems has been addressed mainly using market-based techniques and optimization-based approaches. Market-based techniques use the concept of auctions (Koenig, Keskinocak, and Tovey 2010; Lagoudakis et

[1]Multiple Traveling Salesman Problem, NP-hard

al. 2004; Choi, Brunet, and How 2009), considered state-of-the-art decentralized solutions in MRTA. The auctions require that robots can communicate and use their own information to bid for a task they may probably execute. The negotiation process chooses a winner robot, which the task is assigned to. Optimization-based techniques include deterministic and stochastic methods such as clustering (Janati et al. 2017), graph search (Kartal et al. 2016), or trajectory-based optimizations (Mosteo, Montano, and Lagoudakis 2009; 2008). Recently, auctions were combined with techniques such as clustering (Heap and Pagnucco 2011; Elango, Nachiappan, and Tiwari 2011), or matroid theory techniques (Williams, Gasparri, and Ulivi 2017).

Auction-based methods for MRR solving include combinatorial, sequential or parallel auctions (Koenig, Keskinocak, and Tovey 2010). Combinatorial auctions require exponential amounts of time to compute solutions which minimize the objective. For this, robots bid on all possible combinations of targets. Parallel auctions allocate the targets in linear time single-round auctions, robots bidding for every target only. Sequential single-item auctions (SSI (Koenig et al. 2006)) are a trade-off between combinatorial and parallel auctions, providing in a reasonable time results with guaranteed sub-optimality. Lately, parallel simulations of SSI auctions have been studied (Kishimoto and Nagano 2016). Improvements of SSI auctions include SSI auctions with roll-outs, bundle-bids or regret clearing (Koenig, Keskinocak, and Tovey 2010). In comparison with standard SSI, these auction methods trade-off between better running time and solution quality.

Dynamic task allocation gained attention in the last years, though few works emerged. For example, in search and rescue operations has been addressed with prediction methods (Wei, Hindriks, and Jonker 2016) executed by single-task robots. However, dynamicity consists in interleaving exploration with target retrieval, and the problem is part of the ST-SR-TA taxonomy (one target per robot).

Late results for MRR include the work of (Heap and Pagnucco 2011), which lets robots sequentially bid on task clusters rather than one task at a time. Clusters are formed before the auction, using k-means clustering, and the empirical results consist in lower team costs than SSI auctions (Koenig et al. 2006) (in a similar runtime). As improvement for dynamic allocation, (Heap and Pagnucco 2012) performs cluster-based reallocation once a robot finished one of its allocated tasks. It extends the previous work of sequential single-cluster auctions (SCC) (Heap and Pagnucco 2011), and combines it with ideas of repeated auctioning for post-initial allocation (Nanjanath and Gini 2010). The latter algorithm allows robots to exchange tasks if this improves the overall team objective. The re-plan is performed after every treatment of a task and the empirical results show the final allocation is close to optimal. However, dynamicity is related only to possible delays, caused by communication loss or dynamic obstacles.

K-means clustering with auctions has been proposed in (Elango, Nachiappan, and Tiwari 2011), for balancing the task allocation among all robots. This technique searches to evenly distribute work loads between robots, and however diverges from typical goals of MRR auction methods. None of the previous approaches considers the existence of targets in ongoing missions when beginning the actual allocation, nor any dynamic adaptation of robot missions to new discovered targets. Algorithms require many iterations for allocation (usually one for every target), not always appropriate for dynamic repetitive allocations.

3 Dynamic Multi-Robot Routing (DMRR)

The following definition extends the multi-robot routing (MRR) problem of (Lagoudakis et al. 2005), to evolving missions (growing groups of targets). It introduces *Dynamic Multi-Robot Routing* (DMRR) as a continuous adaptation of MRR to new targets discovered in time. For this, time steps are denoted by the *superscript* t throughout all the paper.

3.1 DMRR Definition

Let $F{=}R{\cup}E$ be a finite fleet composed of robots $R{=}\{r_1, r_2, \dots, r_n\}$ executing missions on their own set of static targets (T_{r_i}) and robots E exploring the environment for target discovery. Let $T{=}\{t_1, t_2, \dots, t_m\}{=}\cup_i T_{r_i}$ be the set of total targets under missions and $T_E{=}\{t_{m+1}, t_{m+2}, \dots, t_{m+p}\}$ be the set of new discovered targets. The locations of robots R, targets T_E and T are known. Finally, let c_{ij} be the cost of moving between two locations i and j in both directions (can be related to measures like energy, distance, travel time, etc.).

Definition 3.1 (DMRR). *The objective of DMRR (at every time new targets T_E are discovered) is to find an allocation of targets $T{\cup}T_E$ to the robots in $R{\cup}S$ (with $S{\subseteq}E$) and paths to treat these targets s.t. an objective function is optimized.*

After each allocation, targets from T_E become part of T, since they are allocated to robot missions. Robots from E can pass to R, in time: at a given time step, allocations can be made for $R^{t+1}{=}R^t{\cup}S$, where $S{\subseteq}E^t$, while exploration robots become $E^{t+1}{=}E^t{\setminus}S$. Until becoming part of R, robots that explore are simply used for updating the set of discovered targets (T_E). Hence, these robots' positions are not priori necessary, if not being subject to allocation. Throughout the paper, the equivalent time dependent notations are $R^t, E^t, T^t_{r_i}, T^t$ and T^t_E; for readability, these are used only when emphasizing time is necessary.

Using the notations of MRR(Lagoudakis et al. 2005), the team objective is written as follows: at any moment in time,

$$\min_A f(g(r_1, A_1), g(r_2, A_2), \dots, g(r_n, A_n)); \qquad (1)$$

consequently, let the asymptotic objective be defined as:

$$\lim_{t\to\infty} \min_{A^t} f(g(r_1, A^t_1), g(r_2, A^t_2), \dots, g(r_n, A^t_n)) = opt, \qquad (2)$$

where f and g measure the performances of the whole team and of each robot, respectively. $A^t{=}\{A^t_1, A^t_2, \dots, A^t_n\}$ is a partition of the targets $T^t{\cup}T^t_E$, where A^t_i is allocated to robot r_i, at moment t. Because the allocation process is repetitive in time, the team objective can be optimized on the short term as well as on the long term (when $t \rightarrow \infty$). So the limit *opt* represents an optimum which f may converge to.

The role of exploration robots is to provide positions of new discovered targets, and to participate to missions when necessary (for example, when the missions of robots R are saturated, e.g. cannot include more targets). Unless explicitly stated otherwise, any robot subject to target allocation refers to a mission robot (not exploration one).

The robot missions might not have an ending time, and targets may be repeatedly visited. Therefore, when the robot position is required, one may use the average of robot's target positions (i.e. centroid of its target cluster). Nevertheless, one may consider the robot is at the position of its last allocated target (MRR assumption).

At moment $t{=}0$, there may be no targets under missions ($T^0{=}\emptyset$, which in fact corresponds to the actual MRR problem), or even no new targets ($T^0{=}T_E^0{=}\emptyset$), if robots did not start the exploration. Then, at any time step t, the set of new targets T_E can change, because of discovered targets, or because of dynamically allocated ones. Like this, targets from T_E always move to T, in time, after their allocation.

In literature, when addressing the MRR problem, several objectives are usually considered, including:

MINSUM: Minimizing the sum of path costs over all robots.
MINMAX: Minimizing the maximum path cost over all robots.

The path cost of a robot r is the cost of visiting all the targets of a cluster S, and let it be denoted by $c(r, S)$ (may be related to energy, distance, time, etc.). Mission costs may also include the time spent on treating targets. Therefore, our discussion makes abstraction of the target treatment cost. Typically for MRR problems, the above objectives concern just the targets in the T_E set only (so the costs $c(r_i, A_i)$, with $A_i{\subseteq}T_E$). For DMRR, A^t is a partition of the $T^t{\cup}T_E^t$, so let the same objectives be defined as follows: at any moment t,

$$\text{MINSUM}_D : \min_{A^t} \sum_i c(r_i, A_i^t), \tag{3}$$

$$\text{MINMAX}_D : \min_{A^t} \max_i c(r_i, A_i^t), \tag{4}$$

and the asymptotic objectives be defined as:

$$\text{MINSUM}_\infty : \lim_{t\rightarrow\infty} \min_{A^t} \sum_i c(r_i, A_i^t) = opt_{\text{MINSUM}}, \tag{5}$$

$$\text{MINMAX}_\infty : \lim_{t\rightarrow\infty} \min_{A^t} \max_i c(r_i, A_i^t) = opt_{\text{MINMAX}}. \tag{6}$$

The $c(r_i, A_i)$'s represent the costs of executing robot mission r_i on the targets $A_i \subseteq T{\cup}T_E$. The asymptotic objectives consider what happens in time with these costs (after a certain amount of steps or an undetermined period of time). In particular, the sum or the average of the path costs may converge towards an optimal value. Under the above objectives, MRR is NP-hard (Lagoudakis et al. 2005), so DMRR is hard to solve as well, being at least as complex as MRR.

Theorem 3.1. *1. There is no polynomial algorithm solving the DMRR, under any of the* MINSUM$_D$, MINMAX$_D$, MINSUM$_\infty$ *or* MINMAX$_\infty$ *objectives (unless P=NP).*

Proof. Consider that $T^0{=}\emptyset$ and T_E^0 contains the new discovered targets. Solving DMRR for time step $t{=}0$ equals to solving the MRR problem. So MRR reduces to DMRR. □

3.2 DMRR with Saturation (DMRR-Sat)

In the context of dynamic adaptation, since the robot mission evolves, the tasks may not have an ending time (e.g. in patrolling, robot execution does not finish). That is why, as robot mission grows in size (e.g. costs of handling new targets), the mission can get saturated. The saturation can be due to robot or target constraints (e.g. robot energy, target visit frequency). The problem, namely DMRR-Sat, considers a saturation bound for each robot mission (in particular, a bound for the path costs $c(r_i, T_{r_i})$). As a consequence, the robots that explore (E) may become part of the robots that treat the targets (R), e.g. when the already ongoing missions get saturated.

Considering the reallocation of all existing targets every time new targets are discovered is extremely time consuming in practice, since the allocation input would be $|T|$, whereas $|T|{\gg}|T_E|$. In addition, it can break the ongoing missions and their related constraints (e.g. for targets which depend on continuous robot execution, like in patrolling), requesting resource consumption for reestablishing all missions. Hence, in DMRR-Sat, only the targets T_E^t are allocated at time t.

Definition 3.2 (DMRR-Sat)**.** *Consider the setting of the DMRR problem. Let Sat be a saturation bound for any robot mission, of the same type as the cost function c. Consider that exploration robots (E) can execute missions on targets from T_E^t at any time t. The DMRR-Sat problem consists in finding allocations of T_E^t to R and robot paths that optimize the team objective f, while respecting constraint $c(r_i, T_{r_i}^t) \leq Sat$, for every robot r_i.*

Since in DMRR-Sat the already allocated targets are not subject to new reallocations, in the objectives (1) and (2) function f becomes

$$f(g(r_1, T_{r_1}{\cup}B_1), g(r_2, T_{r_2}{\cup}B_2), \ldots, g(r_n, T_{r_n}{\cup}B_n))$$

and for objectives (3), (4), (5) and (6), the costs become $c(r_i, T_{r_i}^t{\cup}B_i^t)$, where $\{B_i\}_{i:1,n}$ is a partition of T_E only. Let these objectives be denoted by MINSUM$_{D-Sat}$, MINMAX$_{D-Sat}$, MINSUM$_{\infty-Sat}$ and MINMAX$_{\infty-Sat}$. For example, (3) becomes

$$\text{MINSUM}_{D-Sat} : \min_{B^t} \sum_i c(r_i, T_{r_i}^t{\cup}B_i^t).$$

Because the missions can get saturated and the fleet F is finite, an intuitive objective to consider is that robots in R cover as many targets as possible (the MAXTAR objective). The following objectives are considered for DMRR-Sat:
MINSUM$_{D-Sat}$, MINMAX$_{D-Sat}$,
MINSUM$_{\infty-Sat}$, MINMAX$_{\infty-Sat}$ (after a certain amount of steps),
MAXTAR: $\max_{B^t} \sum_t \sum_i |T_{r_i}^t{\cup}B_i^t|$.
These objectives are subject to the mission saturation constraint $c(r_i, T_{r_i}^t) \leq Sat$.

Theorem 3.2. *There is no polynomial algorithm solving the DMRR-Sat under the above objectives, unless P=NP.*

Proof. The proof shows that DMRR reduces to DMRR-Sat. Let T and T_E be the sets of targets of DMRR problem which needs to be solved. Consider a large enough saturation bound, (e.g. greater than any $c(r_i, T \cup T_E)$ or even $Sat = \infty$). At every time step of DMRR we solve DMRR-Sat for $T' = \emptyset$ and $T'_E = T \cup T_E$, which gives the solution for DMRR. Hence, reduction time is constant. Using Th.3.1, the proof is complete (for the first four objectives).

For the MAXTAR objective, we show that the NP-complete TSP reduces to DMRR-Sat. Let S be the set of targets for the TSP problem. Let all the finite precision real values be scaled to integers. Consider the interval $[Left, Right]=[0, TSP_{sol}]$, where TSP_{sol} is any solution of TSP computed ad-hoc in $O(|S|)$ time. Fix $Sat = \lfloor (Left + Right)/2 \rfloor$ and solve DMRR-Sat for $T=\emptyset$ and $T_E=S$; if MAXTAR $=|S|$ then $Left \leftarrow Sat$, otherwise $Right \leftarrow Sat$. Continue the binary search until $Left=Right$, which is the actual TSP optimal solution. This takes $\log(TSP_{sol})$ steps, so the reduction is linear in the size of the input (and DMRR-Sat is NP-hard under MAXTAR). □

MAXTAR considers that missions must integrate as many targets as possible (intuitively speaking, robots should treat the maximum number of tasks they can). This also minimizes the number of mission robots ($|R|$), resulting in a maximization of the number of exploring robots ($|E|$), which again motivates the objective.

4 Target Allocation Heuristics

Like stated in the related work section, auction-based methods represent the state-of-the-art decentralized solutions for MRR problems. However, existing solutions do not handle allocations of targets for missions that continuously grow and get saturated. Before proposing the dynamic saturation-based auction (DSAT), this section describes sequential and parallel single-item auction methods, like SSI or PSI auctioning. DSAT method combines advantages of sequential and parallel single-item auctions, that are presented in what follows.

4.1 Sequential and Parallel Auctions

Sequential Single-item Auction (SSI) In standard SSI auctions (Koenig et al. 2006), every robot submits the lowest bid among all targets it can bid on. Then, the lowest bid is chosen among all robots, and the corresponding target is acquired by that robot. Once a robot wins a target, it updates its location to the one of the target just got. The auction continues until all the targets have been allocated.

Sequential Single-item Auction with Regret Clearing (SSI-rc) SSI auctions with regret clearing (Zheng et al. 2008) use same mechanism as standard SSI, with one exception: the winner determination phase allocates the target with the largest regret to the robot that made the smallest bid on it. The regret of a target is the difference between its two lowest bids. This time, the robots send the bids for all the targets, in the bidding phase.

Ordered Single-item Auction (OSI) The auction-based greedy approach introduced in (Schneider et al. 2015) assumes the targets are placed in an ordered list. Each target is offered to all robots, one in every auction round. The robot with the smallest bid receives the target and in addition, it updates its own location with the one of the target it just won.

Parallel Single-item Auction (PSI) The method called parallel single-item auction (Koenig et al. 2006) allocates all the targets in one round. The robots submit all their bids, and each target goes to the robot that made the smallest bid on it. This is the fastest approach in terms of running times, but the solution quality is usually the worst, since no path cost is considered in the bidding.

4.2 Dynamic Saturation-based Auction (DSAT)

The coordination system that this paper proposes (DSAT auctioning) combines benefits of sequential and parallel single-item auctions. It allows both single or multiple target allocation in one auction round. DSAT allocation can handle scenarios in which mission can continuously grow in size, and targets can lose candidate robots due to high robot mission costs. While trying to maximize the target coverage, DSAT copes with ideas of parallel auctions (for achieving better running times) and SSI auctions (for objectives such as MINMAX or MINSUM).

At the beginning, all new targets are initially unallocated. The locations of the new targets T_E are known. For any mission robot r, its location is known, and it may be approximated based on the positions of its targets T_r. The location of an exploration robot is required only if the auctioning decides to form a new mission in the end, because of already saturated ones. Robots bid on the targets with an estimated allocation cost. This cost may usually consider the path cost of robot's already allocated targets. The cost can be an approximative or an optimal path cost. Usually, computing the cost of the optimal path can be expensive (e.g. TSP solution). So the robot computes final path minimization when the allocation is finished. Thereby, when bidding for a target t, an approximative path cost is computed by robot r : $c(r, T_r \cup \{t\})$ (usually computed as the sum of distances between allocated targets).

The auction computations can be run by each robot individually (decentralized), or centralized, by an auctioneer robot. The auctioneer would then deliver the results after every auction round. In the decentralized case, a robot listens to the others and memorizes all the other robot bids. Therefore, it can perform the rest of the computations and determine the winners by itself. Like this, all robots execute the algorithm simultaneously and know which target has been allocated to which robot. Because of the decentralized behavior, more messages are passed between robots, but this avoids robots being dependent on an auctioneer.

DSAT Algorithm Description (Alg. 1) The algorithm starts with robots submitting their bids for every target. A robot submits a "not_a_candidate" bid if it estimates that integrating a target would result in its mission cost exceeding the saturation bound. Upon the arrival of all robot bids, the bid lists $B_i=\{b_i^1, b_i^2, \ldots, b_i^{k_i}\}$ are formed for every tar-

Algorithm 1: DSAT-Auction(R, E, T_E)

```
1  Input: $R$: the set of mission robots
2         $E$: the set of exploration robots
3         $T_E$: the set of new discovered targets
4  Output: $T_E=\emptyset$: All targets are allocated
5  for each robot $r \in R$ do
6      for each target $t \in T_E$ do
7          submit bid$(r,t)$ if $c(r, T_r \cup \{t\}) \leq Sat_r$;
8          else submit "not_a_candidate";
9  for each target $t_i \in T_E$ do
10     form the bid list $B_i$;
11     sort$(B_i)$;
12 group the $t_i$'s by their $|B_i|$ (nb. of candidate robots);
13 let $k := \max_i |B_i|$ and $G_k := \{t_i \in T_E | k = |B_i|\}$
14 while $T_E \neq \emptyset$ and $k \geq 1$ do
15     // one round of inverse SSI
16     $T' :=$ InverseSSI$(G_k)$ (for targets with $|B_i|=k$);
17     $T_E := T_E \setminus T'$;
18     update robot bids in every $B_i$ (and update $|B_i|$);
19     regroup $t_i$'s by their $|B_i|$ and recompute $k$;
20 if $k = 0$ then
21     // form new missions for targets with $|B_i| = 0$
22     choose robots $E' \subseteq E$ for missions on $T_E$;
23     run DSAT-Auction$(E', E \backslash E', T_E)$;
```

Algorithm 2: InverseSSI(G_k)

```
1  // sorted lists $B_i=\{b_i^1, b_i^2, \ldots, b_i^k\}$ are already formed
2  // $G_k$ is the target group with $|B_i| = k, \forall t_i \in G_k$
3  $G'_k := G_k$; $Alloc := \emptyset$;
4  while $G'_k \neq \emptyset$ do
5      let $t_i \in G'_k$;
6      $Q := \{t_j \in G'_k \mid r(b_j^1) = r(b_i^1)\}$;
7      $t := \arg\min_{t_j \in Q} b_j^1$;
8      $Alloc \cup= \{t\}$;  $R' \cup= \{r(t)\}$;
9      $T_{r(t)} \cup= \{t\}$;  $G'_k \setminus= Q$;
10 for each robot $r \in R'$ do
11     for each target $t \in T_E$ do
12         submit bid$(r,t)$ if $c(r, T_r \cup \{t\}) \leq Sat_r$;
13         else submit "not_a_candidate";
14 return $Alloc$;
```

get $t_i \in T_E$, and then they are sorted. This helps for faster identification of the smallest bid of a target (b_i^1) when robots update their bids, at the end of an auction round.

DSAT auctioning continues with a clustering-like phase, where targets are grouped based on the size of the B_i lists (i.e. the number of their candidate robots; Lines 12 and 13). Then, the targets with the biggest number of candidate robots are auctioned first (Line 13).

The winer determination and the new bidding phase are performed in what is defined as inverse-SSI auction (and explained below, in Algorithm 2). After one call of the InverseSSI function (i.e. one round of inverse-SSI), the new bids are received by all robots. Each robot then updates the bids in the B_i lists (Line 18). Robots can submit "not_a_candidate" bids, so targets can lose or gain candidate robots. Thereby, the sizes of the candidate lists are updated, so targets get regrouped while the maximum $|B_i|$ is recomputed on the way (Line 19).

When the auction finishes, targets with zero candidates are re-auctioned to form new missions. A set S of exploration robots is selected from E (e.g. the closest robots to T_E centroid). These robots become the input set R in the re-auctioning process. At the end, they might all be mission robots ($R^{t+1}=R^t \cup S$ and $E^{t+1}=E^t \backslash S$), unless there exists a robot $r \in S$, with $T_r=\emptyset$.

Inverse-SSI Auction (Alg. 2) The technique defined by this paper as *inverse-SSI auction* is used as part of the dynamic saturation-based auction, but also as a standalone algorithm. It consists of the robot bidding and winner determination phases. The mechanism is somehow an inverse of the standard SSI auction reasoning. The bids are evaluated from the target perspective and a conflict resolution is performed for the targets which prefer the same robot. However, it differs from standard SSI by its parallelism, since inverse-SSI permits allocating multiple targets in one round. For example, it may be the case that all targets are allocated in one auction round, or to the contrary, taking $|T|$ rounds to complete the auction.

Bidding Phase: The auctioning starts with all robots bidding on all targets. In each of the next auction rounds, only the robots which won in the previous round are submitting their new bids (Line 10). The new bids take into account the new cost of the robot missions, estimated upon the last target allocation.

Winner Determination: Targets are grouped according to the robot of their smallest bid: $r(b_i^1)$, where $t_i \in T_E$. This is the robot that a target prefers, i.e. its first candidate. For targets which prefer the same robot (set Q, Line 6), the target with the smallest bid is selected and allocated for that robot (Line 7). After an allocation, all the targets in Q (Line 9) are discarded from the current auction round. This is because the robot they prefer changed its mission saturation value, after the allocation. Its new bids are used to update the target bid lists used in the next auction rounds.

As standalone algorithm, inverse-SSI is run on the set of targets T_E (main input parameter). It is then executed repeatedly (like standard SSI or SSI-rc), until no more allocations can be made.

DSAT Auction Complexity If a number of $|T|$ targets is subject to allocation, then the auctioning process of DSAT can finish in 1 up to $|T|$ rounds, so the best and worst case running times differ by a large amount of computations. This means DSAT best case time complexity reduces to one complete auction round $O(|T||R| \log |R|)$, as explained below, which outdoes best case complexities of SSI and SSI-rc ($O(|T||R| + |T|^2)$ and $O(|T|^2 \log |R|)$, respectively).

In the beginning, all robots bid on all targets, hence $|R|^2$ messages are transmitted between robots. Then, at most $|T||R|$ messages are sent for all the other rounds, since only the winning robots bid again. This gives a maximum of $|R|^2 + |T||R|$ messages for the whole DSAT auction (like for standard SSI).

If an auctioneer is used (to centralize messages and computations), the number of messages is decreased by a factor of $|R|$ (since no broadcast is needed among robots). Note that if all robots submit their bids at the end of an inverse-SSI round, then a total of $|T||R|^2$ messages are transmitted (the equivalent of SSI with regret clearing).

In what follows, a denotes the number of rounds performed by DSAT auction. The bidding lists B_i are formed and sorted in $|T||R| \log |R|$ steps (using MergeSort algorithm, for example). Grouping the targets by their B_i size takes $|T|$ steps, so a total of $a|T|$ for all DSAT rounds. To complete the winner determination, one round of inverse-SSI takes $2|T|$ time steps: selecting the targets which prefer the same robot and obtaining the minimum of their bids. In total, this requires $a|T|$ steps until DSAT finishes.

Updating a bid of a robot in the B_i lists can be done in time $O(|T| \log |R|)$, since the lists are sorted. For example, binary search can be used for removing the old bid and inserting the new bid in the B_i's. This is performed at most $|T|$ times during the whole auctioning, so the total complexity is $O(|T|^2 \log |R|)$. If required in the end of DSAT auctioning, exploration robots can be chosen randomly in constant time. However, depending on the objective of the DMRR-Sat, a polynomial-time heuristic can be used to compute an approximative cost C of a mission on T_E. Then, a number of $\lceil C/Sat \rceil$ robots from E can be chosen (e.g. robots closest to the T_E centroid).

In conclusion, the worst case time complexity of the whole DSAT algorithm is $O(|T|^2 \log |R|)$ and a number of $|T||R| + |R|^2$ messages are transmitted. This complexity equals both SSI and SSI-rc auctions in terms of running time, and the one of standard SSI for the number of transmitted messages. One may note that the complexity of the winner determination phase is linear in the number of bids submitted: $O(|T||R|)$.

5 Experimental Results

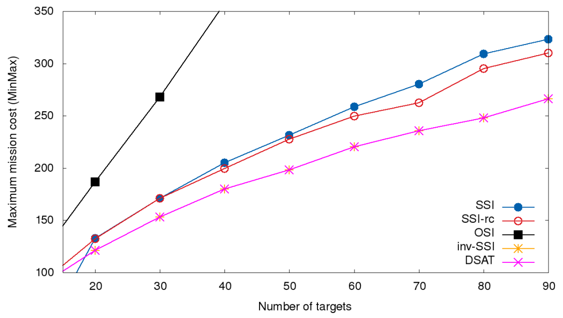


Figure 2: The maximum mission cost among all robots, over the number of targets discovered. No saturation bound is considered ($Sat = \infty$).

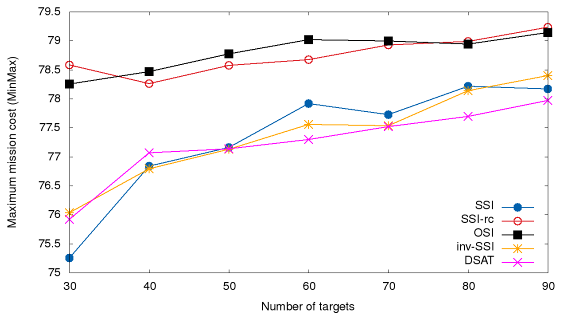


Figure 3: The maximum mission cost among all robots, for a mission saturation bound of 80.

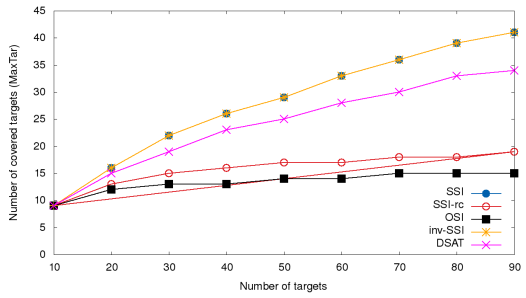


Figure 4: The number of covered targets in the area, for a mission saturation bound of 80. Evolution of costs w.r.t. the number of discovered targets.

Experiments are performed considering euclidean distances in the 2D plane. The targets are randomly generated using the uniform distribution, in an area of 100×100 units. The robots bid for the targets, taking always into account the cost of their mission. For simplicity, these costs are estimated using the sum of the traveling paths (euclidean distances), among the targets they own. The distances are computed in the order in which targets get allocated to the robot. A robot is allowed to bid for a target only if the approximated cost of its integration does not exceed the mission saturation bound. All the computations are performed for an extensive set of 100 random configurations of the targets in the area. All the metrics in this section are mean values obtained in the computations. At the beginning, 4 robots are deployed in the area. Targets are randomly generated, and robots start the auction algorithm for target allocation. Every time new targets are generated, the algorithms are run again. Due to the saturation bound, not all targets can be covered. Therefore, another 3 robots are added in the area, in time. This happens once the targets can not be all covered by the allocation (see Figure 4).

In the experiments, the performance of parallel single-item auction (PSI) is extremely bad. Hence, because of the big differences in the graphics, we leave it out and compare the other five algorithms.

Figure 2 shows the results obtained for the MinMax$_D$ objective, when no saturation bound is considered. DSAT and standalone inverse-SSI, which perform equally here,

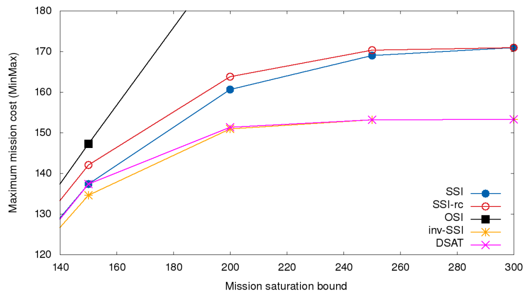


Figure 5: The behavior of the maximum mission cost w.r.t. different saturation values. The number of fixed targets: 30.

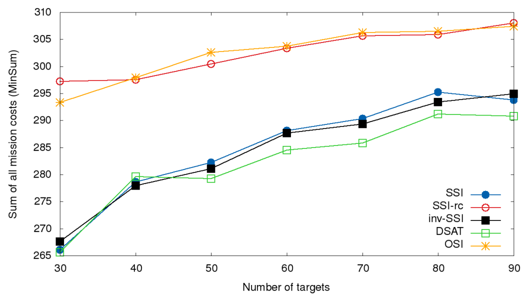


Figure 6: The sum of all the mission costs of all robots, w.r.t. number of discovered targets. Mission saturation bound: 80.

show lower mission costs than all the other algorithms. In this case, all targets are covered by the robots. The mean of MinMax_D for the 100 random target configurations shows a difference of $\approx$ 50 units between SSI and DSAT. This represents around 16% of the mission cost, DSAT outperforming all the other algorithms. Mission costs translate into robot resource usage, hence resource consumption is optimized among all robots. This result applies to DMRR and consequently to MRR, since no saturation bound was considered. Figure 3 shows the resulting MinMax_{D-Sat} for the same configurations, when a mission saturation bound of 80 is considered. Because the costs are now bounded, the values are more compact, and SSI, DSAT and inverse-SSI have almost the same performance (with a slight advantage for DSAT). The costs of OSI do not increase like before (Figure 2), because of the saturation bound; however less targets are covered by the algorithm (Figure 4). Mission saturations reduce the target coverage percentage. In Figure 4, SSI, inverse-SSI and DSAT outperform the other three algorithms with a difference of 50% of covered targets. Here, SSI and inverse-SSI cover the same number of targets.

When changing the saturation bound for the same configurations of targets, the allocations can oscillate, since new bids are allowed to participate. Results in Figure 5 show that DSAT and standalone inverse-SSI continue to outperform standard SSI and SSI with regret clearing, with a difference of $\approx$ 12% in the mission cost. This again results in lower resource usage for the robot missions.

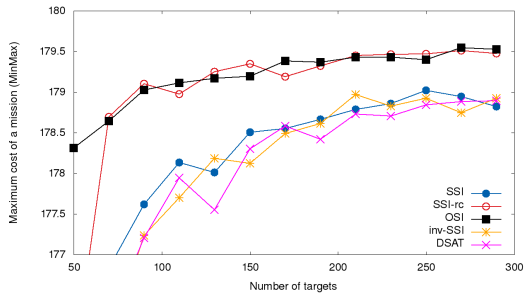


Figure 7: Convergence of the maximum mission cost ($\text{MinMax}_{\infty-Sat}$) towards the optimum ($Sat$=180). An exhaustive generation of targets is performed.

The sum of the mission costs obtained in the experiments can be observed in Figure 6. DSAT has a slight advantage in front of SSI and inverse-SSI (which perform equally), but all three algorithms outperform SSI-rc by a percentage of 5. For the $\text{MinMax}_{\infty-Sat}$ objective, Figure 7 shows the convergence of the algorithms towards the optimum $opt_{\text{MinMax}}=Sat=180$, as new targets are discovered in time. Experiments show that the maximum mission cost tends to the considered optimum, as new targets appear in the area.

6 Conclusion

The present paper exposed the challenges of dynamic target allocation and the drawbacks of the existing literature. It then proposed a framework which defines DMRR (Dynamic Multi-Robot Routing) and contains a mathematical model, complexity results and proofs, but also a collateral problem generated by the evolving missions which dynamically grow (DMRR-Sat). Asymptotic objectives for DMRR were also analyzed, even though assuming an undetermined (or infinite) time period makes them impractical for experiments. DSAT (Dynamic Saturation-based Auction) is proposed as solution for dynamic target allocation and robot coordination. It relies on a technique defined in the paper as inverse-SSI, which stands for both sequential and parallel allocations. Complexity analysis showed that DSAT and inverse-SSI provide running times that overdo the existing state-of-the-art, yet providing better solution qualities when tested (for both DMRR and DMRR-Sat). The experimental results performed on exhaustive sets of inputs show that DSAT and inverse-SSI outperform methods such as SSI or SSI-rc, especially for objectives such as MinMax or MinSum. These results apply also to classic MRR, since experiments were performed as well for DMRR without a saturation bound. Further work emerging from DMRR could consider problems such as dynamic reallocation of targets over time, dynamic reallocations with saturation bounds, or dynamic clustering in target allocation.

Acknowledgments

Funding for this project was provided by a grant from la Région Rhône-Alpes (France) and by the bilateral Franco-Romanian PHC Project “DRONEM”.

References

Beck, Z.; Teacy, L.; Rogers, A.; and Jennings, N. R. 2016. Online planning for collaborative search and rescue by heterogeneous robot teams. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 1024–1033. International Foundation for Autonomous Agents and Multiagent Systems.

Choi, H.-L.; Brunet, L.; and How, J. P. 2009. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics* 25(4):912–926.

Claes, D.; Oliehoek, F.; Baier, H.; and Tuyls, K. 2017. Decentralised online planning for multi-robot warehouse commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 492–500. International Foundation for Autonomous Agents and Multiagent Systems.

Elango, M.; Nachiappan, S.; and Tiwari, M. K. 2011. Balancing task allocation in multi-robot systems using k-means clustering and auction based mechanisms. *Expert Systems with Applications* 38(6):6486–6491.

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

Heap, B., and Pagnucco, M. 2011. Sequential single-cluster auctions for robot task allocation. *AI 2011: Advances in Artificial Intelligence* 412–421.

Heap, B. G. J., and Pagnucco, M. 2012. Repeated sequential auctions with dynamic task clusters. In *AAAI*.

Janati, F.; Abdollahi, F.; Ghidary, S. S.; Jannatifar, M.; Baltes, J.; and Sadeghnejad, S. 2017. Multi-robot task allocation using clustering method. In *Robot Intelligence Technology and Applications 4*. Springer. 233–247.

Kartal, B.; Nunes, E.; Godoy, J.; and Gini, M. 2016. Monte carlo tree search with branch and bound for multi-robot task allocation. In *The IJCAI-16 Workshop on Autonomous Mobile Service Robots*.

Khamis, A.; Hussein, A.; and Elmogy, A. 2015. Multi-robot task allocation: A review of the state-of-the-art. In *Cooperative Robots and Sensor Networks 2015*. Springer. 31–51.

Kishimoto, A., and Nagano, K. 2016. Evaluation of auction-based multi-robot routing by parallel simulation. In *ICAPS*, 495–503.

Koenig, S.; Tovey, C.; Lagoudakis, M.; Markakis, V.; Kempe, D.; Keskinocak, P.; Kleywegt, A.; Meyerson, A.; and Jain, S. 2006. The power of sequential single-item auctions for agent coordination. In *Proceedings of the national conference on artificial intelligence*, volume 21, 1625. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Koenig, S.; Keskinocak, P.; and Tovey, C. A. 2010. Progress on agent coordination with cooperative auctions. In *AAAI*, volume 10, 1713–1717.

Korsah, G. A.; Stentz, A.; and Dias, M. B. 2013. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research* 32(12):1495–1512.

Lagoudakis, M. G.; Berhault, M.; Koenig, S.; Keskinocak, P.; and Kleywegt, A. J. 2004. Simple auctions with performance guarantees for multi-robot task allocation. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, 698–705. IEEE.

Lagoudakis, M. G.; Markakis, E.; Kempe, D.; Keskinocak, P.; Kleywegt, A. J.; Koenig, S.; Tovey, C. A.; Meyerson, A.; and Jain, S. 2005. Auction-based multi-robot routing. In *Robotics: Science and Systems*, volume 5, 343C350. Rome, Italy.

Mosteo, A. R.; Montano, L.; and Lagoudakis, M. 2008. Multi-robot routing under limited communication range. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1531–1536. IEEE.

Mosteo, A. R.; Montano, L.; and Lagoudakis, M. G. 2009. Guaranteed-performance multi-robot routing under limited communication range. *Distributed Autonomous Robotic Systems* 8:491–502.

Nanjanath, M., and Gini, M. 2010. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems* 58(7):900–909.

Pillac, V.; Gendreau, M.; Guéret, C.; and Medaglia, A. L. 2013. A review of dynamic vehicle routing problems. *European Journal of Operational Research* 225(1):1–11.

Pippin, C.; Christensen, H.; and Weiss, L. 2013. Performance based task assignment in multi-robot patrolling. In *Proceedings of the 28th annual ACM symposium on applied computing*, 70–76. ACM.

Schneider, E.; Sklar, E. I.; Parsons, S.; and Özgelen, A. T. 2015. Auction-based task allocation for multi-robot teams in dynamic environments. In *Conference Towards Autonomous Robotic Systems*, 246–257. Springer.

Tardioli, D.; Mosteo, A. R.; Riazuelo, L.; Villarroel, J. L.; and Montano, L. 2010. Enforcing network connectivity in robot team missions. *The International Journal of Robotics Research* 29(4):460–480.

Toth, P., and Vigo, D. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.

Tovey, C.; Lagoudakis, M.; Jain, S.; and Koenig, S. 2005. The generation of bidding rules for auction-based robot coordination. *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III* 3–14.

Wei, C.; Hindriks, K. V.; and Jonker, C. M. 2016. Dynamic task allocation for multi-robot search and retrieval tasks. *Applied Intelligence* 45(2):383–401.

Williams, R. K.; Gasparri, A.; and Ulivi, G. 2017. Decentralized matroid optimization for topology constraints in multi-robot allocation problems. In *Robotics and Automation (ICRA), 2017*, 293–300. IEEE.

Zheng, S. K. X.; Tovey, C.; Borie, R.; Kilby, P.; Markakis, V.; and Keskinocak, P. 2008. Agent coordination with regret clearing. In *Proceedings of the Twenty-third (AAAI-08) Conference on Artificial Intelligence, 13-17 July 2008, Chicago, Illinois, USA: Machine Learning*, 101. AAAI Press.

Iterative Planning for Deterministic QDec-POMDPs

Sagi Bazinin
Ben-Gurion University
Beer-Sheva 84105, Israel
sagibaz@post.bgu.ac.il

Guy Shani
Ben-Gurion University
Beer-Sheva 84105, Israel
shanigu@cs.bgu.ac.il

Abstract

QDec-POMDPs are a qualitative alternative to stochastic Dec-POMDPs for goal-oriented planning in cooperative partially observable multi-agent environments. Although QDec-POMDPs share the same worst case complexity as Dec-POMDPs, previous research has shown an ability to scale up to larger domains while producing high quality plan trees. A key difficulty in distributed execution is the need to construct a joint plan tree branching on the combinations of observations of all agents. In this work, we suggest an iterative algorithm, IMAP, that plans for one agent at a time, taking into considerations collaboration constraints about action execution of previous agents, and generating new constraints for the next agents. We explain how these constraints are generated and handled, and a backtracking mechanism for changing constraints that cannot be met. We provide experimental results on multi-agent planning domains, showing our methods to scale to much larger problems with several collaborating agents and huge state spaces.

1 Introduction

In many real-world problems agents collaborate to achieve joint goals. For example, disaster response teams typically consist of multiple agents that have multiple tasks to perform, some of which require the cooperation of multiple agents. In such domains, agents typically have partial information, as they can sense their immediate surroundings only. As agents are often located in different positions and may possess different sensing abilities, their runtime information states differ. Sometimes, this can be overcome using communication, but communication infrastructure can be damaged, or communication may be costly and should be reasoned about explicitly.

In this setting it is common to plan for all agents jointly using a central engine. The resulting policy, however, is executed by the agents in a decentralized manner, and agent communication is performed only through explicit actions.

Decentralized POMDPs (Dec-POMDPs) offer a rich model for capturing such multi-agent problems (Bernstein et al. 2002), but Dec-POMDPs solvers have difficulty to scale up beyond small toy problems. Qualitative Dec-POMDP (QDec-POMDP) were offered as an alternative model, replacing the quantitative probability distributions over possible states with qualitative sets of states (Brafman, Shani, and Zilberstein 2013).

Although QDec-POMDPs share the same worst case complexity class as Dec-POMDPs, Brafman et al. have shown that a translation-based approach into contingent planning managed to scale to model sizes that could not be solved by Dec-POMDP algorithms. The policy for a QDec-POMDP can be represented as a joint policy tree (or graph), where nodes are labeled by joint actions of all agents, and edges are labeled by the possible joint observations following those actions. In a solution tree all the leaves correspond to goal states. The policy tree hence has a huge branching factor, limiting the ability to scale up to larger problems. One can extract single agent local policy trees from the joint policy tree, where each local tree has an exponentially smaller branching factor. In this paper we focus on deterministic QDec-POMDPs, where one can find solutions with a finite depth.

In many problems interactions between cooperating agents are limited to a number of key points. Each agent may be able to achieve a set of tasks that require no cooperation, while assisting other agents only in several collaborative tasks. This is illustrated in the box pushing domain (Figure 2), where light boxes can be pushed into place by a single agent, but a heavy box can only be pushed by two agents together.

In such cases, it may be useful to plan for each agent independently, creating a single agent plan tree, branching only on the observations of the specific agent. We suggest an iterative approach, which we call IMAP (iterative multi agent planning) where the central planning engine plans for one agent at a time. IMAP creates a local policy tree for each agent, instead of a joint policy tree for all agents.

When planning for a single agent IMAP assumes that other agents will be available to assist in required collaborations. These assumptions generate a set of conditional constraints on the behavior of other agents, that must be considered when planning for these agents. When a constraint cannot be satisfied, we backtrack to the agent that required the constraint.

In addition, agents attempt to perform tasks at the lowest cost, notifying all other agents of the cost for completing subgoals. Agents that manage to complete some tasks more cheaply, inform backward to previous agents, that re-plan again ignoring these tasks. Thus, our approach also contains a task allocation component that assigns tasks to agents to reduce the overall cost for completing all tasks.

For solving single agent problems, we compile the multi-agent QDec-POMDP into a single agent contingent planning problem. We use an off-the-shelf offline contingent planner (Komarnitsky and Shani 2016a) to generate a plan graph, and then extract the conditional constraints from that plan graph. Our method is sound, but incomplete, due to the greedy nature of our iterative process. Still, it scales up to very large QDec-POMDPs. We provide an empirical study, focusing on scaling up analysis, showing how our approach scales to very large domains, with multiple agents, many order of magnitudes beyond domains solvable by previous approaches.

2 Model Definition

We start with the basic definition of a flat-space QDec-POMDP, followed by a factored definition motivated by contingent planning model definitions (Bonet and Geffner 2014; Brafman and Shani 2016).

Definition 1. *A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple* $Q = \langle I, S, b_0, \{A_i\}, \delta, \{\Omega_i\}, O, G\rangle$ *where*

- *I is a finite set of agents indexed* $1, ..., m$.
- *S is a finite set of states.*
- *$b_0 \subset S$ is the set of states initially possible.*
- *A_i is a finite set of actions available to agent i and $\vec{A} = \otimes_{i\in I} A_i$ is the set of joint actions, where $\vec{a} = \langle a_1, ..., a_m\rangle$ denotes a particular joint action.*
- *$\delta : S \times \vec{A} \to 2^S$ is a non-deterministic Markovian transition function. $\delta(s, \vec{a})$ denotes the set of possible outcome states after taking joint action $\vec{a}$ in state s.*
- *Ω_i is a finite set of observations available to agent i and $\vec{\Omega} = \otimes_{i\in I}\Omega_i$ is the set of joint observation, where $\vec{o} = \langle o_1, ..., o_m\rangle$ denotes a particular joint observation.*
- *$\omega : \vec{A} \times S \to 2^{\vec{\Omega}}$ is a* non-deterministic *observation function. $\omega(\vec{a}, s)$ denotes the set of possible joint observations $\vec{o}$ given that joint action $\vec{a}$ was taken and led to outcome state s. Here $s \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.*
- *$G \subset S$ is a set of goal states.*

We do not assume here a finite horizon T, limiting the maximal number of actions in each execution. We focus, however, on deterministic outcomes and deterministic observations. In such cases a successful solution is acyclic, and there is hence no need to bound the number of steps. Extension to domains with non-deterministic outcomes with a bounded horizon is simple, but extensions to infinite horizon and non-deterministic outcomes is beyond the scope of this paper. We assume a shared initial belief, like most Dec-POMDP models.

Definition 2. *A factored QDec-POMDP is a tuple $\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G\rangle$ where I is a set of agents, P is a set of propositions, $\vec{A}$ is a vector of individual action sets, Pre is the precondition function, Eff is the effects function, b_0 is the set of initially possible states, and G is a set (conjunction) of goal propositions. The state space S consists of all truth assignments to P, and each state can be viewed as a set of literals.*

The precondition function Pre maps each individual action $a_i \in A_i$ to its set of preconditions, i.e., a set of literals that must hold whenever agent i executes a_i. Preconditions are local, i.e., defined over a_i rather than $\vec{a}$, because each agent must ensure that the relevant preconditions hold prior to executing its part of the joint action. We extend Pre to be defined over joint actions $\{\vec{a} = \langle a_1, .., a_m\rangle : a_i \in A_i\}$ (where $m = |I|$): $Pre(\langle a_1, .., a_m\rangle) = \cup_i Pre(a_i)$.

Brafman et al. (2013) define an effects function Eff mapping joint actions into a set of pairs (c, e) of conditional effects, where c is a conjunction of literals and e is a single literal, such that if c holds before the execution of the action e holds after its execution. Thus, effects are a function of the *joint* action rather than of the local actions, as can be expected, due to possible interactions between local actions.

We slightly refine these definitions to explicitly support independent and collaborative actions. For each local action a_i of agent i we define a set of local conditional effects $eff_l(a_i) = \{(c, e)\}$. In addition, for each subset of local actions of agents, $\{a_{i_1}, ..., a_{i_k}\}$ of agents $i_1, ..., i_k$ we define another set of collaborative conditional effects $eff_c(\{a_{i_1}, ..., a_{i_k}\}) = \{(c, e)\}$. When this set is empty, we say that the local actions are independent, when the set is not empty, we say that the subset of local actions are collaborative. We further require that collaborative subsets will be minimal, in that for two subsets A^1, A^2 such that $A^1 \subset A^2$, $eff_c(A^1) \neq eff_c(A^2)$. While the effects of collaborative actions are shared, the preconditions are not. Specifically, when applying a collaborative action each agent must only ensure that its own preconditions hold.

For every joint action $\vec{a}$ and agent i, $Obs(\vec{a}, i) = \{p_1, \ldots, p_k\}$, where $p_1, ..., p_k$ are the propositions whose value agent i observes after the joint execution of $\vec{a}$. The observation is private, i.e., each agent may observe different aspects of the world. We assume that the observed value is correct and corresponds to the post-action variable value.

While QDec-POMDPs allow for non-deterministic action effects as well as non-deterministic observations, we focus in this paper only on deterministic effects and observations, and leave discussion of an extension of our methods to non-determinism to future research. Additionally, our model assumes a shared initial belief state, as most Dec-POMDP models. The case where agents have different initial belief states is very important, as it corresponds to the situation in on-line planning, but is also challenging.

2.1 Policy Trees

We can represent the local plan of an agent i using a *policy tree* τ_i, which is a tree with branching factor $|\Omega|$. Each node of the tree is labeled with an action and each branch is labeled with an observation. To execute the plan, each agent performs the action at the root of the tree and then uses the subtree labeled with the observation it obtains for future action selection. If τ_i is a policy tree for agent i and o_i is a possible observation for agent i, then $\tau_{i_{o_i}}$ denotes the subtree that corresponds to the branch labeled by o_i.

An alternative to local trees is a global joint policy tree, where nodes are labeled by joint actions and edges are labeled by joint observations. We argue that constructing global trees directly is difficult, due to the need to consider all possible observation combinations together, and suggest an iterative construction of local policy trees as a more scalable approach.

Let $\vec{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_m \rangle$ be a vector of policy trees. We denote the joint action at the root of $\vec{\tau}$ by $\vec{a}_{\vec{\tau}}$, and for an observation vector $\vec{o} = o_1, \ldots, o_m$, we define $\vec{\tau}_{\vec{o}} = \langle \tau 1_{o_1}, \ldots \tau_{m_{o_m}} \rangle$. When executing a policy tree each agent i maintains its own local belief b_i — the set of possible states given the observations that i has observed during the execution. Initially, all agents set $b_i = b_0$, but following the different observations it is often the case that $b_i \neq b_j$ for two agents i and j. $tr(\vec{b}, \vec{o}, \vec{a})$ denotes the set of local beliefs after executing the joint action $\vec{a}$ and observing $\vec{o}$ starting from the local beliefs $\vec{b}$. To execute $\vec{\tau}$ we first consider the action $\vec{a}_{\vec{\tau}}$ in the context of the local beliefs. That is, each agent i must validate that $b_i \models pre(a_{\tau_i})$. If for some agent j, $b_j \not\models pre(a_{\tau_j})$ then the execution is not valid. Each agent i then executes a_{τ_i}, observes o_i, transitioning to one of the subtrees τ'_i of τ_i with a new local belief b'_i. We say that $\vec{\tau}$ is a valid set of policy trees if every possible execution for $\vec{\tau}$ starting from b_0 is valid. If the precondition of $\vec{a}_{\vec{\tau}}$ are met at the initial local beliefs $\langle b_0, ..., b_0 \rangle$, and for every possible joint observation $\vec{o}$, executing $\vec{\tau}_{\vec{o}}$ in the new local belief $tr(\vec{b}, \vec{a}_{\vec{\tau}}, \vec{o})$ is valid.

A local policy tree τ_i is valid if there exists a valid vector $\vec{\tau}$ of local policy trees containing τ_i. For example, a local policy tree τ_i is not valid if for some t, for two different branches of τ_i of length t, a collaborative action with some agent j appears in one branch but not in the other, and agent j cannot distinguish between the two branches. That is, some differentiating observations cannot be observed by j.

A set of policy trees $\vec{\tau}$ is called a *joint policy* if executing the policy trees starting from the initial belief b_0 results in a valid execution. A joint policy is called a *solution* if for all leaves in the tree $\bigcap_i b_i \models G$, i.e., the set of possible states given the joint local beliefs of the agents satisfy the goal.

Example 1. We now illustrate the factored QDec-POMDP model using a simple box pushing domain (Figure 1). In this example there is a one dimensional grid of size 3, with cells marked 1-3, and two agents, starting in cells 1 and 3. In each cell there may be a box, which needs to be pushed upwards. The left and right boxes are light, and a single agent may push them alone. The middle box is heavy, and requires that the two agents push it together.

We can hence define $I = \{1, 2\}$ and $P = \{AgentAt_{i,pos}, BoxAt_{j,pos}, Heavy_j\}$ where $pos \in \{1, 2, 3\}$ is a possible position in the grid, $i \in \{1, 2\}$ is the agent index, and $j \in \{1, 2, 3\}$ is a box index. In the initial state each box may or may not be in its corresponding cell — $b_0 = AgentAt_{1,1} \wedge AgentAt_{2,3} \wedge (BoxAt_{j,j} \vee \neg BoxAt_{j,j})$ for $j = 1, 2, 3$. There are therefore 8 possible initial states.

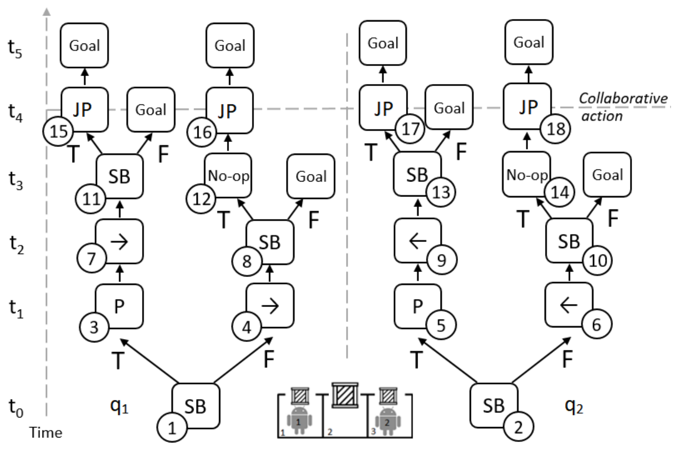


Figure 1: Illustration of Example 1 showing the box pushing domain with 2 agents and a possible set of local plan trees that produce a solution. Possible agent actions are sensing a box at the current agent location (denoted SB), moving (denoted by arrows), pushing a light box up alone (denoted P), jointly pushing a heavy box (denoted $JointPush$), and no-op.

The allowed actions for the agents are to move left and right, to push a light box up, or jointly push a heavy box up with the assistance of the other agent. There are no preconditions for moving left and right, i.e. $Pre(Left) = Pre(Right) = \emptyset$. For agent i to push up a light box j, agent i must be in the same place as the box. That is, $Pre(PushUp_{i,j}) = \{AgentAt'_{i,j} \neg Heavy_j, BoxAt_{j,j}\}$. For the collaborative joint push action the precondition is $Pre(JointPush_j) = \{AgentAt_{1,j}, AgentAt_{2,j}, Heavy_j, BoxAt_{j,j}\}$.

The moving actions transition the agent from one position to the other, and are independent of the effects of other agent actions, e.g., $Right_i = \{(AgentAt_{i,1}, \neg AgentAt_{i,1} \wedge AgentAt_{i,2}), (AgentAt_{i,2}, \neg AgentAt_{i,2} \wedge AgentAt_{i,3})\}$. The only joint effect is for the *JointPush* action — $Eff(PushUp_{1,2}, a_2)$ where a_2 is some other action, are identical to the independent effects of action a_2, while $Eff(PushUp_{1,2}, PushUp_{2,2}) = \{(\emptyset, \neg BoxAt_{2,2})\}$, that is, if and only if the two agents push the heavy box jointly, it (unconditionally) gets moved out of the grid.

We define sensing actions for boxes — $SenseBox_{i,j}$, with precondition $Pre(SenseBox_{i,j}) = AgentAt_{i,j}$, no effects, and $Obs(SenseBox_{i,j}) = BoxAt_{j,j}$. The goal is to move all boxes out of the grid, i.e., $\bigwedge_j \neg BoxAt_{j,j}$.

3 Iterative Construction of Policy Trees

We now describe our main contribution — a method to construct satisfying policy trees iteratively, which we call IMAP, for iterative multi agent planning. The first agent constructs an independent policy tree using only its own independent actions, and collaborative actions that it participates in, assuming that the other agents required to execute the collaborative actions will be available to assist. This single agent problem is solved by a contingent planner (Komarnitsky and Shani 2016a), returning a policy tree.

After the single agent policy tree is computed, the agent manipulates the tree to be a valid local policy tree in a QDec-POMDP. Then, the agent extracts constraints for other agents, encapsulating the assistance requirement of the computed policy.

We now create a new single agent problem for the next agent, containing the computed constraints. The next agent attempts to solve this constrained problem. If the agent succeeds, it again extracts constraints and passes them on. If the single agent contingent planning problem cannot be solved, however, the agent reports the failure back, with additional information, and the first agent must replan.

The process terminates when all agents agree on a set of local policy trees that achieve the goal. Then, we run a soundness check. The following subsections describe the various parts of this algorithm. For ease of exposition, we describe our methods below assuming time, or unit costs. Our methods are directly applicable to the case of varying costs.

A high level description of IMAP is presented in Algorithm 1.

Algorithm 1: Iterative Planning for QDEC-POMDP

```
Input: QDec-POMDP $\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G \rangle$
$A_c \leftarrow \emptyset$ (collaborative action constraints)
$i \leftarrow 1$ (current agent)
for each goal literal $g$, $G^T(g) \leftarrow \infty$
while $i < n$ do
    $\tau_i \leftarrow$ solve $\langle P_i, A_i^+, Pre_i, Eff_i, Obs_i, b_0, G \rangle$
    if $\tau_i$ is valid then
        Align and order collaborative actions in $\tau_i$
        $G_i^T \leftarrow$ goal achievement times in $\tau_i$
        if $\exists g \in G$ s.t. $G_i^T(g) < G^T(g)$ then
            $G^T(g) \leftarrow G_i^T(g)$
            $j \leftarrow$ the earliest agent that achieved $g$ s.t.
              $G_i^T(g) < G^T(g)$
            Undo all constraints by agents $j$ .. $i$
            $i \leftarrow j$
        for Collaborative action $a_c$ in $\tau_i$ at time $t$ do
            Add constraint on $a_c$ at time $t$ to $A_c$
        $i \leftarrow i + 1$
    else
        for constraint $c$ in $A_c$ by increasing time do
            $\tau \leftarrow$ Solve $\langle P_i, A_i^+, Pre_i, Eff_i, Obs_i, b_0, c \rangle$
            if $\tau$ is not valid then
                $c_f \leftarrow c$
                $t \leftarrow$ earliest time that $c_f$ could be achieved
        $j \leftarrow$ the agent that introduced $c_f$
        notify $j$ that $c_f$ can be achieved by time $t$
        Undo all constraints by agents $j$ .. $i$
        $i \leftarrow j$
```

3.1 Compilation to a Single Agent Problem

We now describe the creation of a single agent problem for the first agent 1. The planning problems for the next agents will be based on this compilation, adding constraints which we later describe. Given a factored QDec-POMDP $\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G \rangle$ we create for agent i a single agent problem $\langle P_i, A_i^+, Pre_i, Eff_i, Obs_i, b_0, G \rangle$. The observations Obs_i, the initial belief b_0 and the set of goals G remain as in the multi agent problem.

A_i^+ is the set of single agent i, containing all the independent actions in A_i, as well as all collaborative actions that i participates in. That is, for each minimal subset $\{a_{j_1}, ..., a_{j_k}\}$ such that $Eff_c(\{a_{j_1}, ..., a_{j_k}\}) \neq \emptyset$ and there exists l such that $a_{j_l} \in A_i$, we add a single agent action $a_{\{j_1,...,j_k\}}$. The created action has the same preconditions as a_{j_l} — the component of agent i in the joint action. This represents an assumption of agent i that the other agents that participate in the collaborative action will fulfill the needed precondition for the collaborative action to apply.

In addition, we identify a set P_i^- of non-constant propositions that none of the independent or collaborative actions of agent i can achieve, yet appear in a precondition of an action $a \in A_i^+$ or in the goal G. These are propositions which may be required for achieving the goal, yet cannot be produced by agent i. We remove all these propositions from the problem description. This represents an assumption of agent i that other agents will produce these propositions when needed.

We can now run a contingent planner on the compiled problem and obtain a single agent policy tree τ.

3.2 Adjustments to the Policy Tree

It may not be possible to construct a joint policy using τ, as we must ensure that all collaborating agents execute a collaborative action together. For example, consider the single agent policy tree for agent 1 in Figure 2. This tree is a solution for the compiled problem, assuming that agent 2 would assist in joint-push actions when needed. However, we can observe that in the left branch, the collaborative joint push action is at time t_4, while in the right branch the joint push is at time t_3. To be able to assist at different times, agent 2 must know whether agent 1 is at the right or left branch.

To handle this, we force all the instances of a collaborative action to occur at the same time in all branches, by adding *no-op* actions (Figure 2 right), creating a *leveled* policy tree.

Leveling the collaborative actions may be difficult when we have multiple collaborative actions in a branch. Given two collaborative actions a_1, a_2, if a_1 precedes a_2 in all branches, then we can level the execution time as before, by first leveling the tree for a_1, then leveling the tree for a_2. However, when a_1 precedes a_2 in one branch, and a_2 precedes a_1 in another branch, we cannot level both together.

There can be several ways to create a valid leveled policy tree. First, we can replan forcing the planner to decide on one ordering of all collaborative actions. There can be cases where this will make the problem unsolvable. We can also condition the difference on an observed variable that all collaborating agents can observe. We currently take the first

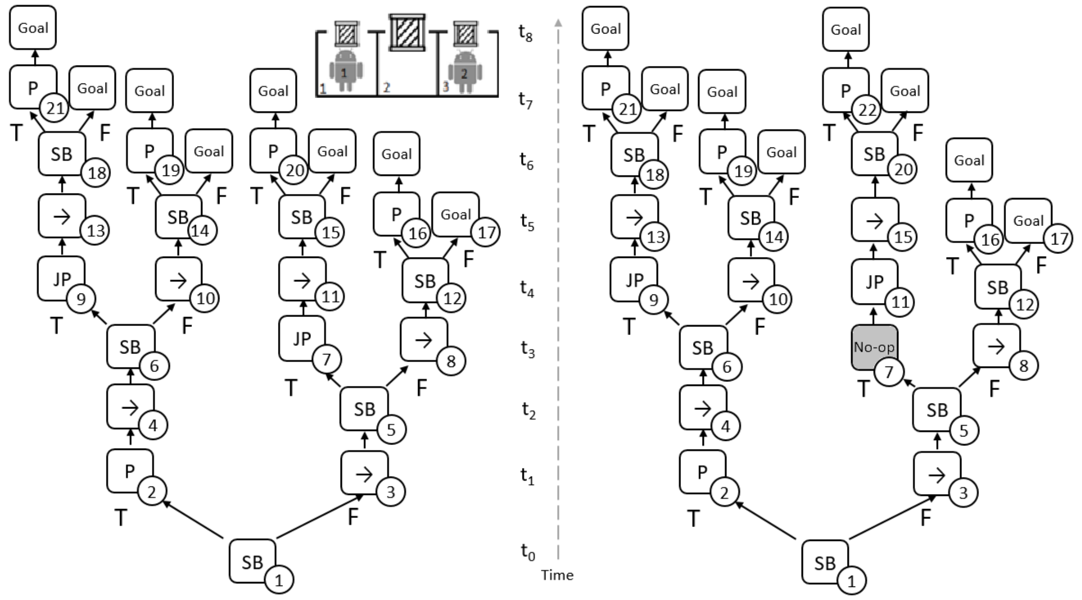


Figure 2: Policy tree (left) and an adjusted tree (right) for the compiled contingent problem of agent 1 in the simple box pushing domains. A *no-op* action (grayed) was inserted to level the JointPush action to be executed at time t_4

approach, and fail if the planner cannot order all collaborative actions in a consistent order of execution in all branches.

3.3 Extracting Constraints for Other Agents

Following the adjustments to the tree τ, resulting in tree τ_i for agent i, we now extract a set of constraints on the policy tree of other agents. We can extract two types of constraints — collaborative action constraints and missing preconditions constraints.

Collaborative Action Constraints For each collaborative action a_c executed at time t in policy tree τ_i, we add a constraint for all other agents that participate in a_c to also execute the action at time t. However, even though a_c is executed always at time t, in some branches it might be that a_c is not executed at all. Consider the tree in Figure 2. In two branches the agents do not jointly push the heavy box, because it is already at the target position. Hence, the constraint to jointly push the heavy box applies only in branches where the box is not initially at its target position.

The collaborative action constraints are hence *conditional* constraints, conditioned on the value of some observed variables. The collaborative action a_c must be executed only in branches where the value of the observed variables conforms to the conditioned values. To identify these variables we look at the set of branches B_{a_c} where a_c was used, and the set of branches $B_{\neg a_c}$ where a_c was not used.

We identify the set of literals P_{a_c} that occur in all branches in B_{a_c} prior to the execution of a_c in that branch, and the set of literals $P_{\neg a_c}$ that occur in all branches in $B_{\neg a_c}$ throughout the branch execution. Then, $P_{a_c} \setminus P_{\neg a_c}$ defines the difference between these branches. All agents that collaborate on a_c must be able to observe the value of these propositions. Otherwise, the collaborative action cannot be soundly executed.

To employ the constraint in the single agent compiled problem, we implement time into the contingent problems. Although it is inconvenient to implement time into the propositional, PDDL based, description that we use, in our case we implement time only for a limited horizon, equivalent to the depth d of the already computed tree. We set the successor of time t_d to be t_∞, and the successor of time t_∞ to be also t_∞, allowing other agents to plan beyond time d if need be. We add a time parameter to all actions.

We now set the preconditions of all actions except a_c at time t to contain the conditioned variables values. That is, all actions except a_c can be executed at time t only in branches that conform to the conditioned variable values, where a_c was not executed in the original tree τ_i. In the example above, we add to the precondition of all actions at time t_4 the literal $\neg BoxAt_{2,2}$. In order to execute any action other than $JointPushUp_{2,2}$ at time t_4 the agent must first observe the value of $BoxAt_{2,2}$. Although it is not always the case, in this example, the collaborative action $JointPushUp_{2,2}$ has a precondition $BoxAt_{2,2}$, forcing the agent to observe the value of $BoxAt_{2,2}$ before time t_4 in all branches.

3.4 Forward Progression and Backtracking

Given the constraints extracted from the adjusted policy tree for agent i, we now create a new single agent planning problem for the next agent $i+1$. The planning problem augments the definition in Section 3.1 with the collaborative action and missing precondition constraints above. Constraints irrelevant to agent $i+1$, such as missing preconditions that $i+1$ cannot achieve, or collaborative actions that do not apply to $i+1$ are not added onto the single agent problem.

Agent $i+1$ now runs the contingent solver over the constructed single agent planning problem. If a solution is found, then we again level the policy tree, extract additional constraints, and create a new planning problem for the next

Time	t_1	t_2	t_3	t_4
Agent	a_1	a_2	a_1	a_2
$g_1 = \neg BoxAt_{1,1}$	1	-	**1**	**-**
$g_2 = \neg BoxAt_{2,2}$	4	4	**4**	**4**
$g_3 = \neg BoxAt_{3,3}$	6	1	**-**	**1**

Table 1: Iterated improvements of goal completion time for the simple box-pushing problem. Agent a_1 plans alone, achieving goals g_1, g_2, g_3 at t_1, t_4, t_6, respectively, and assuming that a_2 will help pushing Box_2 at t_4. Agent a_2 manages helping pushing Box_2 and achieves g_3 faster at t_1, therefore improving the overall plan and forcing a_1 to replan. Agent a_1 plans without considering g_3, and a_2 replans ignoring g_1.

agent based on all constraints gathered thus far.

Although the original, multi-agent problem may have a solution, the single agent problem of $i + 1$ may not be solvable. For example, it may be that agent i added a constraint for agent $i + 1$ to be at position p at time t, but agent $i + 1$ is unable to reach p at the appropriate time. To understand why the solver fails, we now plan again for each constraint for agent $i + 1$, attempting to achieve the specified constraints by reversed order of appearance. That is, we start removing constraints that occur later in the plan, until a solution is found. This allows us to identify the first constraint c that agent $i + 1$ cannot achieve at the required time t_c.

We now replan for agent $i + 1$ given all constraints prior to c, where the goal is to achieve c at any time after t_c. If c cannot be achieved at any possible time after t_c, then we fail. Otherwise, the planner computes a plan that achieves all constraints prior to c, and achieves c at time t' at the latest.

We now identify the agent $j < i + 1$ that required the constraint c, which agent $i + 1$ cannot fulfill in the given time t_c, and report back that the constraint c can be achieved only at time $t' > t_c$. We now backtrack and replan for agent j, with a constraint that c can only be achieved at t' at the earliest. All plans between j and $i + 1$ are removed. That is, if j manages to solve the problem, we move to agent $j + 1$ and continue.

3.5 Sub-Goal Assignment

While the above process can be used to solve the multi-agent problem, it may produce inefficient plans. Consider, for example, the policy tree generated in Figure 2. The agent produced a solution for pushing all 3 boxes. Obviously, however, it is more efficient to leave the rightmost box at position 3 for agent 2 to handle, as done in the local policy trees in Figure 1. We now describe a mechanism that allows agent 1 to entrust the task of pushing the rightmost box to agent 2.

In addition to the constraints described in Section 3.3, we extract from the policy tree of agent i for each goal literal $g \in G$, the minimal time t_g when g is achieved. We allow other agents to acknowledge that g can be achieved by agent i at time t_g, by adding a conditional effect $(time_{t_g}, g)$ to all actions. That is, every action executed at time t_g achieves g.

An agent j may, however, achieve g at a time $t' < t_g$. In our running example, as shown in Table 1, agent 1 achieves the goal $\neg BoxAt_{3,3}$ at time 6. Agent 2 can help agent 1 in pushing the heavy box, and then wait for time 6, but a shorter plan for agent 2 achieves $\neg BoxAt_{3,3}$ at time 1, and only then assist agent 1 with the heavy box, at which point all goals have been achieved.

When progressing forward in the agent sequence we maintain for each such goal $g \in G$ only the earliest time of achievement t_g, and add the resulting conditional effects when planning for all agents $j > i$. Once we successfully finish planning for the last agent, we start again from the first agent, this time allowing agents to use all the goal achievement conditional effect. Each agent uses only goal achievement effects created by other agents, to avoid a case where agent i assigns goal g to agent j, while j assigns g to i, and thus no agent actually achieves g.

This process may be repeated several times, because an agent that earlier achieved g at time t_g, may now be able to achieve g at time $t' < t_g$, for example because it now ignores other goals achieved more rapidly by other agents. However, each time we start over it is because at least one goal was achieved at an earlier time than in the previous iteration. Hence, this process must terminate eventually, and we repeat this goal allocation replanning process until we converge.

3.6 Ensuring Soundness

There can be several reasons why the above procedure may not produce a sound, executable, plan. For example, it may be that one agent consumes a precondition that another agent relies on. One can simulate the joint policy for every possible initial state, effectively traversing the implicit joint policy tree, but this process is exponential.

We take instead an approximate approach. Each agent collects from all other agents the effects of their actions at each time step. Then, the agent checks whether the preconditions of its local policy tree are not invalidated by the actions of other agents. This requires us to iterate only twice over each local policy tree, once to collect the effects and once to check that preconditions are not invalidated by other agents.

Our soundness test is approximate because when an agent produces an effect p in two different branches, at different time steps t, t', where $t < t'$, we assume that p is achieved at time t in all branches. Thus, our soundness test is stronger than needed, and it may be that a valid solution would be discarded, but not vice versa.

If the test failed, then we report a failure. We leave a deeper discussion of resolving such problems by introducing additional constraints to future work.

4 Experimental Results

We now provide experimental results showing that our algorithm can scale up to large QDec-POMDP problems. We experiment with two domains — a variant of the well-known box pushing problem (Seuken and Zilberstein 2007), and an adaptation of the rovers domain (Stolba, Komenda, and Kovacs 2016). All transitions and observations in both domains are deterministic. We use CPOR (Komarnitsky and

Table 2: Comparing IMAP to Dec-POMDP solvers over small, 1D, box pushing problems. $|A_i| = 4$, $|\Omega_i| = 2$. W is the grid width, L and H denote the number of light and heavy boxes. $E[C]$ is computed given a uniform initial belief and unit costs. *Planning* is the number of single agent planning episodes.

Domain					Compilation		GMAA-ICE		DICEPS		JESP		IMAP		
W	L	H	$\|S\|$	$\|b_0\|$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	Planning
2	2	0	16	4	12.8	1.5	**0.22**	**1.5**	16.18	**1.5**	0.78	**1.5**	7.23	4.6	2
3	2	0	36	4	25.5	1.5	**0.58**	**1.5**	18.19	**1.5**	1.67	**1.5**	6.28	4.87	2
3	2	1	70	8	50.3	4.5	×	×	40.88	**3.85**	×	×	**37.07**	4.06	2
5	2	1	200	8	164.6	6.5	×	×	83.4	**4.38**	×	×	**41.84**	12.24	4
5	4	1	800	32	×	×	×	×	×	×	×	×	**123.17**	**14.17**	4

Shani 2016b) as the underlying planner, and Metric-FF as the classical planner of CPOR. The experiments were run on a Windows 10 64-bit machine, $i5, 2.2GHz$ CPU, and 8GB RAM. Our method is implemented in $C\#$, while Dec-POMDP solvers were run on an Ubuntu virtual box on the same machine.

Domains In the box pushing domain a set of boxes are spread in a grid, and the agents must push each box to a designated location at the edge of the grid (the end of the column it appears in). Each box may be either in a pre-specified location, or at its goal location to begin with. The agent must be in the same location as the box in order to observe where it is. Agents may move in the 4 primary directions, and can push boxes to an upper cell, if they occupy the same location as the box. Some boxes are heavy and must be pushed by two agents jointly. The number of agent actions is hence 9, and the number of observations is 2.

We also experiment with an adaptation of the multi-agent rovers domain, where multiple rovers must collect together measurements of soil, and rock. The agents navigate a map of waypoints, and successful measurements can only be taken at some waypoints, unknown initially to the agents. When an agent is at a waypoint it can attempt a measurement, which may be successful or not based on whether the waypoint is appropriate for that measurement. Images of rocks, and samples of soils can be collected by a single rover, while rock samples require two rovers working jointly. After collecting the measurements, the rovers must broadcast them back to the ground station.

Comparison to Dec-POMDP Solvers Table 2 shows a comparison between IMAP, exact (GMAA-ICE (Spaan, Oliehoek, and Amato 2011)) and approximate (JESP (Nair et al. 2003), DICEPS (Oliehoek, Kooij, and Vlassis 2008)) Dec-POMDP solvers on small box pushing domains. All solvers were executed on various horizons until convergence, and we report the result over the horizon with best $E[C]$ (computed assuming a uniform initial belief). In all domains we used a 1D grid, and hence only 4 actions per agent (observe-box, push, move-right, move-left). We also compare to the compilation-based approach (Brafman, Shani, and Zilberstein 2013). All solvers managed to solve only very small problems, with a short horizon.

We acknowledge that this comparison is not entirely fair, because Dec-POMDP solvers try to optimize solution quality, whereas we only seek a satisfying solution. Thus, Dec-POMDP solvers may need to explore many more branches of the search graph, at a much greater computational cost. Furthermore, many Dec-POMDP solvers are naturally anytime, and can possibly produce a good policy even when stopped before termination. It may well be that solvers may reach a satisfying policy, which is the goal in a QDec-POMDP, well before they terminate their execution.

Table 3: IMAP on large domains. *C* is the number of required collaborations, *T* is the runtime (secs), *M* is the final makespan, $E[C]$ is the expected cost under a uniform distribution, and *P* is the number of agent planning episodes.

$\|S\|$	$\|I\|$	$\|A_i\|$	$\|\Omega\|$	$\|b_0\|$	C	T	M	$E[C]$	P
Box pushing									
196	2	8	2	4	1	13.2	16	8	5
400	2	8	2	4	1	30	18	10.2	3
25000	5	8	3	8	1	56.2	13	6.82	12
1000	3	8	3	8	1	59	12	6.68	11
4000	3	8	5	32	2	247.1	19	11.6	9
Rovers									
308	2	15	2	4	1	13.1	18	7.81	2
462	2	16	3	8	1	14.4	16	8.7	2
300	2	16	7	128	1	33.7	35	19.3	2
20250	3	15	5	32	1	37.1	12	5.5	3
1500	3	15	7	128	1	156.3	26	11.2	6
600	2	16	9	512	1	205.01	39	17.6	4

IMAP on Larger Problems Table 3 shows results over large domains. We experimented with varying grid sizes, number of agents, and different compositions of light and heavy boxes . We report runtime (T, secs.), the expected cost under a uniform distribution ($E[C]$), and the plan makespan (M, maximal time to completion), as a measure of plan quality, and the total number of single agent planning episodes (P). The difficulties in this domain are mainly due to the number of collaborations, and the alternative goal assignments.

The problem difficulty is defined, as before, by the number of agents and the size (number of waypoints), but also by the uncertainty — the number of potential waypoints where a measurement can be taken. While the number of states is smaller, the number of actions, and the initial belief uncertainty, are larger in this domain compared to the box pushing problems. We can see here that the difficulty mostly stems from the size of the initial belief, requiring more complex single agent plan trees.

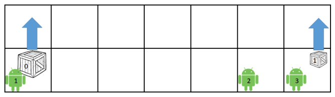


Figure 3: Box pushing example for Table 4

As can be seen, IMAP scales well to problem sizes well beyond the ability of current planners. The complexity of the problem grows both with the state space size, as well as the number of agents, the number of required collaborative actions, and the initial uncertainty. The number of planning episodes encapsulates both constraints that could not be met, or plan improvements by later agents. Where the number of planning episodes is identical to the number of agents, there was no need to backtrack. The largest number of planning episodes is 12, where 5 agents had to push 3 boxes, and several agents suggested improvements to the initial plan, requiring us to go back and forth. Still, this did not cause the planner to take an exceptionally long time. The main bottleneck of IMAP is the time required to compute a single agent plan in the larger domains, not the number of replanning episodes.

Figure 3 shows an example a box pushing domain with 3 agents and 2 boxes, box_0 which is heavy, and box_1 which is light. Table 4 shows the execution of IMAP on this domain. Agent 1 wants to push box_0 with agent 2 at time 4, and then push box_1 at time 14. Agent 2 reports that it can only push box_0 at time 7. Agent 1 now selects agent 3 for collaboration. Agent 2, relieved of pushing box_0 can push box_1 at time 5. Agent 1 confirms this plan improvement. When reaching agent 3 it reports that it can help with box_0 only at time 8. Agent 1 replans, and chooses collaboration with agent 2, who cannot push box_1 at time 5. Agent 3 now reports that it can push box_1 at time 4. All agents confirm the new plan.

5 Discussion and Related Work

Our approach is well rooted in the multi-agent literature. Iterating over the agents, focusing on one agent at a time is used for reducing the complexity of considering a joint policy. For example, JESP (Nair et al. 2003) modifies the policy of one agent, while keeping the policies of all other agents fixed. The exponential complexity of considering all possible joint observations is not reduced, though, making scaling up difficult, as can be seen in our experiments as well.

Similar approaches have been used in other multi agent problems such as DCOP (Chapman et al. 2011), privacy preserving planning (Borrajo 2013), and many more.

Focusing on interaction points between the agents is also a well known idea. In Dec-POMDPs, (Spaan and Melo 2008) reduce the problem complexity by considering states in which agents must interact, and states where they can act independently. Similar intuitions were pursued by (Witwicki and Durfee 2010) and (Oliehoek, Witwicki, and Kaelbling 2012), for decoupling the state space considering how agents influence one another.

To resolve the dependencies, the AI community has suggested that agents should enforce commitments (Jennings 1993): constraints on policies that must be adhered. In multi agent planning, these commitments take a very similar role and structure to what we do (Brafman and Domshlak 2008).

As such, the main contribution in this paper is in an efficient adaptation of these well known ideas into the new QDec-POMDP framework, scaling well beyond the ability of current exact and approximate Dec-POMDP solvers.

Table 4: IMAP execution example. P denotes the planning episode, BT denotes the reason for backtracking (F - constraint failure, I - goal improvement).

P	1	2	3	4	5	6	6	7	8	9	10	11	12
Agent	1	2	1	2	1	2	3	1	2	3	1	2	3
box_0	4	×	4	4	4	4	×	7	7	7	7	7	7
box_1	14	14	14	5	5	5	5	5	17	4	4	4	4
BT		F		I			F			I			

6 Conclusion

We presented IMAP — an iterative algorithm for multi-agent planning for QDec-POMDPs, which iteratively plans for a single agent. IMAP assumes that other agents will be available to help the agent with collaborative actions, and produces constraints to the other agents for the required collaborations. Later agents that cannot meet these constraints results in backtracking. In addition, our algorithm allows for later agents to improve the plans of the first agents, by taking responsibility for some tasks, causing again backtracking.

We experiment with two types of domains, the well-known box pushing domains and a new Dec-POMDP domain adapted from the multi-agent planning community. On both domains, we have shown a scaling up ability well beyond the limitation of current Dec-POMDP planners.

In the future, we will experiment with more domains, focusing on other types of collaborations. E.g., we would explore cases where each agent completes a part of a task, but there are no collaborative actions. We will also explore cases where agents execute actions that interfere with other agents, such as actions that consume a precondition required by another agent. We believe that simple extensions of our approach will be able to handle such cases.

An important extension of our algorithm is for non-deterministic actions, incurring loops. Our constraints must be significantly modified to cope with loops.

7 Acknowledgments

Supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center, and by the Lynn and William Frankel Center for Computer Science

References

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27:819–840.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res.* 50:923–970.

Borrajo, D. 2013. Multi-agent planning by plan reuse. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1141–1142. International Foundation for Autonomous Agents and Multiagent Systems.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Brafman, R. I., and Shani, G. 2016. Online belief tracking using regression for contingent planning. *Artif. Intell.* 241:131–152.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*

Chapman, A. C.; Rogers, A.; Jennings, N. R.; and Leslie, D. S. 2011. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems1. *The Knowledge Engineering Review* 26(4):411–444.

Jennings, N. R. 1993. Commitments and conventions: The foundation of coordination in multi-agent systems. *The knowledge engineering review* 8(3):223–250.

Komarnitsky, R., and Shani, G. 2016a. Computing contingent plans using online replanning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 3159–3165.

Komarnitsky, R., and Shani, G. 2016b. Computing contingent plans using online replanning. In *AAAI*, 3159–3165.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D.; and Marsella, S. 2003. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI*, volume 3, 705–711.

Oliehoek, F. A.; Kooij, J. F.; and Vlassis, N. 2008. The cross-entropy method for policy search in decentralized pomdps. *Informatica* 32(4).

Oliehoek, F. A.; Witwicki, S. J.; and Kaelbling, L. P. 2012. Influence-based abstraction for multiagent systems. In *AAAI*.

Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 344–351.

Spaan, M. T., and Melo, F. S. 2008. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, 525–532. International Foundation for Autonomous Agents and Multiagent Systems.

Spaan, M. T.; Oliehoek, F. A.; and Amato, C. 2011. Scaling up optimal heuristic search in dec-pomdps via incremental expansion. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, 2027.

Stolba, M.; Komenda, A.; and Kovacs, D. L. 2016. Competition of distributed and multiagent planners (codmap). In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 4343–4345.

Witwicki, S. J., and Durfee, E. H. 2010. Influence-based policy abstraction for weakly-coupled dec-pomdps. In *ICAPS*, 185–192.

Stubborn Sets Pruning for Privacy Preserving Planning

Tim Schulte
Albert-Ludwigs-Universität Freiburg
schultet@cs.uni-freiburg.de

Abstract

We adapt a partial order reduction technique based on *stubborn sets* to the setting of privacy-preserving multi-agent planning. We proof that the presented approach preserves optimality and show experimentally that it can significantly improve search performance on some domains.

Introduction

Recently, privacy preserving planning (Nissim and Brafman 2014) has become an increasingly popular multi-agent planning framework. It enables agents to engage in a cooperative planning process in order to compute joint plans that achieve mutual goals. Notably, the framework allows agents to keep certain information private. There are many settings in which this is of great importance. Consider, for instance, research departments of different companies that want to collaborate on a common project in order to mutually benefit from each others' competence. Exchanging proprietary data could diminish the benefits of this endeavor.

Heuristic search is a particularly successful approach to privacy-preserving planning. Specifically, multi-agent forward search (MAFS) (Nissim and Brafman 2012) has proven to be highly efficient, when coupled with good heuristic functions (Štolba and Komenda 2014; Štolba, Fišer, and Komenda 2015). However, when accurate heuristic estimates are unavailable, the search space is often searched exhaustively (e.g. when the search gets stuck on a plateau). Even with almost perfect heuristic estimates, search effort can scale exponentially (in the size of the planning task), when an optimal solution is sought (Helmert and Röger 2008). In these cases, additional pruning techniques that narrow down the number of state expansions, while preserving optimality, can substantially improve the search performance.

Partial order reduction (POR) techniques exploit that independent actions can be applied in an arbitrary order. Ideally, search algorithms would consider only one such order, thereby reducing the number of expanded states exponentially. Partial order reduction based on *stubborn sets* (Valmari 1989) strives to achieve just that and has successfully been applied to optimal (single agent) planning (Alkhazraji et al. 2012; Wehrle et al. 2013). In this paper we adapt and apply stubborn sets pruning to the privacy-preserving planning setting. The main challenge addressed is how to account for private information without losing completeness or optimality. We show experimentally that the revised algorithm can significantly improve search performance.

As a running example, we use a new domain, inspired by a production site. The goal is to produce a set of products with certain properties. The agents must process the products to establish their required properties. Each property has a corresponding processing action, all of which are private and independent of one another. A concrete example that embodies this type of domain has the agents building personal computers according to a given set of orders. Each order specifies an individual PC setup, i.e. the set of components the PC should consist of. Many components, like hard disc drives, physical drives, sound card, working memory, etc., can independently be installed onto the mainboard.

Background

We consider multi-agent planning in a notational variant of the privacy-preserving planning formalism (Nissim and Brafman 2014). The formalism extends *classical planning* with a notion of *agents*, their respective *action sets*, and a *privacy partition*.

Definition 1 (Multi-agent planning task)**.** *A* multi-agent planning task *is a tuple* $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i \in N} \rangle$*, where*

- $N = \{1, 2, \ldots, n\}$ *is a finite set of agents,*
- V *is a finite set of* state variables. *Each* $v \in V$ *is associated with a domain* D_v. *A* variable assignment *is a function* s *with domain* $D_s \subseteq V$*, such that* $s(v) \in D_v$ *for all* $v \in D_s$. *A* variable assignment *defined for all variables in* V *is called* state.
- s_0 *is the* initial state*,*
- $s_\star$ *is a variable assignment over* V *called the* goal*,*
- A_i *is a finite set of actions available to agent* i. *Each* action $a = \langle \mathrm{pre}(a), \mathrm{eff}(a), \mathrm{c}(a) \rangle \in A_i$ *consists of two variable assignments over* V *called* precondition $\mathrm{pre}(a)$ *and an* effect $\mathrm{eff}(a)$*, and a* cost $\mathrm{c}(a) \in \mathbb{R}_0^+$. *The set of all actions is* $A = \bigcup_{i \in N} A_i$.

An action a is *applicable* in state s if s agrees with *pre*(a) wherever *pre*(a) is defined. Application of action a in state s yields the *successor state* $a(s)$ which agrees with *eff*(a)

where *eff*(a) is defined, and agrees with s, elsewhere. The set of all applicable actions in state s is $app(s)$. The solution to a planning task is a sequence of actions $\pi = (a_1, \ldots, a_k)$ such that a_1 is applicable in s_0, every subsequent action is applicable in the state generated by its preceding action, and $a_k(\ldots(a_1(s_0))\ldots) \models s_\star$.

Multi-agent planning tasks can be conceived as "agent-decoupled" classical planning tasks, and are solvable by centralized classical planning systems like Fast Downward (Helmert 2006). Some settings require agents to preserve privacy during the planning process. By constraining the agents to keep certain information on the planning task private, the use of distributed planning techniques becomes sensible. We now introduce the required notation to then define the *privacy-preserving* extension to multi-agent planning.

Definition 2 (Projection). *Let s be a variable assignment over the set of variables V. The projection of s to $V' \subseteq V$ is a variable assignment $s|_{V'}$ that is defined on V' and agrees with s wherever it is defined, i.e. $s|_{V'}(v) = s(v)$, for all $v \in V'$.*

Definition 3 (Action projection). *The* projection *of an action a to the set of variables V' is $a|_{V'} = \langle \mathrm{pre}(a)|_{V'}, \mathrm{eff}(a)|_{V'}, \mathrm{c}(a)\rangle$.*

Consequentially, the projection of a set of actions A to the set of variables V' is defined as $A|_{V'} = \{a|_{V'} | a \in A\}$.

Definition 4 (Privacy partition). *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task. A* privacy partition *is an indexed family of sets*

$$\mathcal{P} = \{P_v\}_{v \in V}$$

that, for each variable $v \in V$, contains the set of agents $P_v \subseteq N$ that have access to v.

In this paper, we only consider privacy partitions where all sets $P_v, v \in V$ have a cardinality of either one or $|N|$. Furthermore, if $v \in D_{s_\star}$ then $P_v = N$. Thus, $\mathcal{P}$ partitions the set of variables V into a set of public *variables V^{pub}, known to all agents, and $|N|$ sets of* private *variables V_j^{pri}, each known to a single agent $j \in N$ only:*

- $V_j^{pri} = \{v \in V \mid P_v = \{j\}\}$, *for $j \in N$*
- $V^{pub} = \{v \in V \mid P_v = N\}$

Actions are partitioned into a set of public actions A^{pub} and sets of private actions A_j^{pri}, accordingly:

- $A_j^{pri} = \{a \in A_j \mid a = a|_{V_j^{pri}}\}$, *for $j \in N$*
- $A^{pub} = \bigcup_{j \in N} (A_j \setminus A_j^{pri})$

Definition 5 (Local view). *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task and $\mathcal{P}$ be a privacy partition for Π. The* local view *of agent j on Π is defined as*

$$\Pi^j = \langle N, V^j, s_0^j, s_\star, \{A_i^j\}_{i\in N}\rangle, \text{ where}$$

- $V^j = V^{pub} \cup V_j^{pri}$,
- $s_0^j = s_0|_{V^j}$, *and*
- $A_i^j = (A_i \setminus A_i^{pri})|_{V^j}$ *for $i \neq j$, and $A_j^j = A_j$.*

Definition 6 (Privacy preserving planning task). *A* privacy preserving planning task *is a tuple $(\Pi, \mathcal{P})$ consisting of a multi-agent planning task Π and a privacy partition $\mathcal{P}$.*

A multi-agent planning algorithm is *weakly private* if each agent can only access its own *local view* on the planning task and the agents never exchange private information with one another. A multi-agent planning algorithm is *strongly private* if no agent can deduce private information from the course of conversation (message history) between the agents. Private information includes knowledge about the existence or value of a variable private to another agent, or an action model (Brafman 2015).

Multi-Agent Forward Search

Because agents can only access a factor (their local view) of the original multi-agent planning task, cooperation with other agents becomes a necessity.

Multi-Agent Forward Search (MAFS) (Nissim and Brafman 2014) is a general search scheme for privacy preserving multi-agent planning. Each agent conducts a best-first search, maintaining its own *open* and *closed* list. Successors of expanded states are generated by using the agents' own actions only. Whenever a state is generated for which another agent has an applicable public action, a message is sent to that agent. The message contains the full state, heuristic score and g-value of the sending agent. Private fluents of the state are encrypted such that only the relevant agents can decrypt it. When agent i receives a message $m = \langle s, h_j(s), g_j(s)\rangle$ of some other agent j, it checks whether s is already in its open or closed list. If this is not the case, agent i puts s on its open list. If agent i generated state s previously with higher cost, then it puts s on its open list again and assigns new costs $g_j(s)$ to it. When an agent generates a goal state, it initiates a distributed plan extraction procedure by broadcasting the goal state in a message to all agents.

Strong Stubborn Sets

Strong stubborn sets can be used within forward search algorithms to potentially reduce the number of successor states generated in each *state expansion* step. Instead of expanding a state s by generating a successor state $a(s)$ for each applicable action $a \in app(s)$, only a subset of actions $T_{app(s)} \subseteq app(s)$ needs to be considered. Applicable actions that are not contained in $T_{app(s)}$ are said to be *pruned.*

In the following, we provide the definitions of *action dependencies*, *disjunctive action landmarks* (Helmert and Domshlak 2009), and *necessary enabling sets*, which are the three crucial components for the computation of strong stubborn sets.

Definition 7 (Action dependency). *Let $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i\in N}\rangle$ be a multi-agent planning task, and let $a_1, a_2 \in A$.*

- *a_1* disables *a_2 if there exists a variable $v \in V$ and facts $\langle v, d_1\rangle \in \mathrm{eff}(a_1)$ and $\langle v, d_2\rangle \in \mathrm{pre}(a_2)$ s.t. $d_1 \neq d_2$.*
- *a_1 and a_2* conflict *if there exists a variable $v \in V$ and facts $\langle v, d_1\rangle \in \mathrm{eff}(a_1)$ and $\langle v, d_2\rangle \in \mathrm{eff}(a_2)$ s.t. $d_1 \neq d_2$.*

Algorithm 1: Strong stubborn set computation of agent i for state s (incomplete)

```
Input: $\Pi = \langle N, V, s_0, s_\star, \{A_i\}_{i \in N}\rangle$, state $s$
Result: strong stubborn set $T_s \subseteq A$
1 $T_s \leftarrow L_s^{s_\star}$ for some DAL $L_s^{s_\star}$ for $s_\star$ in $s$
2 repeat
3     forall $a \in T_s$ do
4         if $a \in app(s)$ then
5             $T_s \leftarrow T_s \cup \mathrm{dep}(a)$
6         else
7             $T_s \leftarrow T_s \cup N_s^a$ for some NES $N_s^a$
8 until $T_s$ reaches a fixed-point
9 return $T_s$
```

- *a_1 and a_2 are dependent if a_1 disables a_2, or a_2 disables a_1, or a_1 and a_2 conflict. We write* $\mathrm{dep}(a)$ *for the set of actions with which a is dependent.*

Definition 8 (Disjunctive action landmark)**.** *A* disjunctive action landmark (DAL) *for a set of facts F in state s is a set of actions L such that every applicable action sequence that starts in s and ends in $s' \supseteq F$ contains at least one action $a \in L$.*

Definition 9 (Necessary enabling set)**.** *A* necessary enabling set (NES) *for action $a \notin app(s)$ in state s is a disjunctive action landmark for* $\mathrm{pre}(a)$ *in s.*

We can now give the definition of strong stubborn sets. It is identical to the one used in classical single-agent planning (Alkhazraji et al. 2012) except for the input data now being a multi-agent planning problem.

Definition 10 (Strong stubborn set)**.** *Let* Π *be a multi-agent planning task with actions A and goal $s_\star$, and let s be a state of* Π. *A* strong stubborn set (SSS) *in s is an action set $T_s \subseteq A$ such that:*

1. *For each $a \in T_s \cap app(s)$, we have* $\mathrm{dep}(a) \subseteq T_s$.
2. *For each $a \in T_s \setminus app(s)$, we have $N_s^a \subseteq T_s$ for some necessary enabling set N_s^a of a in s.*
3. *T_s contains a disjunctive action landmark for $s_\star$ in s.*

The above definition of strong stubborn sets ensures that for every plan π for the current state s, a permutation of π exists which is not pruned. An algorithm to compute a strong stubborn set for a state s is given in Algorithm 1. The state expansion step of forward search algorithms can be modified in the following way: before expanding state s, compute the respective strong stubborn set T_s using Algorithm 1, then expand s by applying the actions in $T_{app(s)} := T_s \cap app(s)$ only. As a consequence, states reached by actions in $app(s)$ but not in T_s are pruned.

Strong Stubborn Sets Revised

We now exemplify two ways in which the pruning of successor states based on strong stubborn sets, as defined above, violates completeness when used in combination with a privacy preserving distributed planning approach, like MAFS. We then propose two possible adaptations that make up for the identified shortcomings and argue that the revised stubborn set approach maintains the completeness property of MAFS.

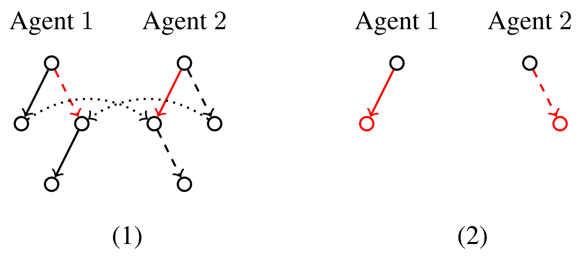
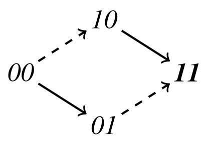


Figure 1: Two ways in which planning for the task of Example 1 goes wrong. Agent 1's action is represented by a solid arrow, agent 2's action by a dashed arrow. Dotted arrows represent state transmissions. Pruned actions are marked red.

The first example fails despite the absence of private information. The second example fails because required information is private to another agent.

Example 1. *Let* $(\Pi, \mathcal{P}) = (\langle N, V, s_0, s_\star, \{A_1, A_2\}\rangle, \mathcal{P})$ *be a privacy preserving planning task, with*

Search space



$$
\begin{aligned}
N &= \{1,2\}, V = \{v_0, v_1\} \\
\mathcal{P} &= \{N, N\} \\
s_0 &= \{v_0 \to 0, v_1 \to 0\} \\
s_\star &= \{v_0 \to 1, v_1 \to 1\} \\
A_1 &= \{a\} \text{ with } a = \langle v_0 \to 0, v_0 \to 1\rangle \\
A_2 &= \{b\} \text{ with } b = \langle v_1 \to 0, v_1 \to 1\rangle
\end{aligned}
$$

There is no private information (all variables are known to both agents), hence, the agents' public projections are identical, i.e. $\Pi^1 = \Pi^2 = \Pi$. *When expanding a state, each agent independently applies stubborn set pruning (Algorithm 1) to potentially reduce the number of generated successor states. Ideally, the agents would either prune state* 10 *or state* 01.

Because actions a and b are independent of one another and both are applicable in the initial state, the respective strong stubborn set contains either a or b (but not both), depending on the choice of the initial disjunctive action landmark. When both agents choose the same DAL, they would virtually agree on either pruning state 01 *or state* 10 *(which is the desired outcome). If the agents select different DALs, however, this leads to the following undesirable outcomes, depicted in Figure 1:*

(1) *The agents end up generating all states. This happens because both of them prune the other's initial action. Therefore, they generate different successor states, which they then transmit to the respective other agent.*

(2) *The agents end up in a* livelock, *waiting for one another to apply the first action (which they pruned) and to transmit the resulting state (which never happens).*

In this example, planning fails because the agents do not synchronize their pruning efforts, using different strong stubborn sets, and prune "public" successor states, relevant to the other agent. We can resolve this issue by enforcing that the stubborn sets computed by each agent only include the agent's own actions. That way, the agents would not prune their own public action in the initial state and therefore not end up in a livelock. We will explain this in more detail further below and continue by emphasizing another issue that occurs when dealing with private information.

Example 2. *Let* $(\Pi, \mathcal{P}) = (\langle N, V, s_0, s_\star, \{A_1, A_2\}\rangle, \mathcal{P})$ *be a privacy preserving planning task, with*

$$\begin{tikzcd}
& 100 \arrow[dr] & & & \\
000 \arrow[ur] \arrow[dr] & & 110 \arrow[r, dashed] & 111 \arrow[r, dashed] & \mathbf{211} \\
& 010 \arrow[ur] & & &
\end{tikzcd}$$

$$N = \{1, 2\}, V = \{v_0, v_1, v_2\}, \mathcal{P} = \{N, N, \{2\}\}$$
$$s_0 = \{v_0 \rightarrow 0, v_1 \rightarrow 0, v_2 \rightarrow 0\}$$
$$s_\star = \{v_0 \rightarrow 2\}$$
$$A_1 = \{a, b\}, A_2 = \{c, d\} \textit{ with}$$
$$a = \langle v_0 \rightarrow 0, v_0 \rightarrow 1\rangle, b = \langle v_1 \rightarrow 0, v_1 \rightarrow 1\rangle$$
$$c = \langle v_0 \rightarrow 1 \wedge v_1 \rightarrow 1 v_2 \rightarrow 1\rangle, d = \langle v_2 \rightarrow 1, v_0 \rightarrow 2\rangle$$

Here, agent 1 *can savely prune either action* a *or* b *in the initial state, thereby avoiding either state* 100 *or* 010*, respectively. This, however, does not work out, as both agents are planning with their local view. Consider the local view of agent* 1*:*

$$N = \{1, 2\}, V^1 = \{v_0, v_1\}$$
$$s_0^1 = \{v_0 \rightarrow 0, v_1 \rightarrow 0\}, s_\star = \{v_0 \rightarrow 2\}$$
$$A_1^1 = \{a, b\}, A_2^1 = \{c|_{V^1}, d|_{V^1}\} \textit{ with}$$
$$a = \langle v_0 \rightarrow 0, v_0 \rightarrow 1\rangle, b = \langle v_1 \rightarrow 0, v_1 \rightarrow 1\rangle$$
$$c|_{V^1} = \langle v_0 \rightarrow 1 \wedge v_1 \rightarrow 1, \emptyset\rangle, d|_{V^1} = \langle \emptyset, v_0 \rightarrow 2\rangle$$

To agent 1 *there appears to be no connection between agent* 2*'s actions, i.e.* c *and* d *appear to be independent. Furthermore, action* d *appears to be applicable in the initial state. Applying Algorithm 1 in* s_0 *therefore yields the following strong stubborn set* T_s*:*

$$T_s = \{d\}$$

Hence:

$$T_{app(s_0)} = app(s_0) \cap T_s = \{a, b\} \cap \{d\} = \emptyset$$

Since agent 1 *has no action to apply in its initial state, no goal can be reached and completeness is violated.*

Initially, agent 1 *computes a disjunctive action landmark for the goal (Algorithm 1, line 1). The only action satisfying a goal condition is action* d *of agent* 2*. The stubborn set therefore initially consists of action* d *only. According to Algorithm 1 either the set of dependent actions* $\mathrm{dep}(d)$ *has to be added to* T_s *(if* d *is applicable in* s_0*) or a necessary enabling set for* d *(if* d *is not applicable in* s_0*). Neither is possible for agent* 1*. Additionally, agent* 1 *cannot decide correctly, whether action* d *is applicable in* s_0 *or not. This would require knowledge of private information not available to agent* 1*. In the example, agent* 1 *adds all dependent actions instead of a necessary enabling set for* d*. Since action* d *is not dependent on any other action in agent* 1*'s local view, a fixed point is reached and the algorithm returns* $T_s = \{d\}$*. Even if agent* 1 *could decide that action* d *was not applicable in* s_0*, he could not add a valid necessary enabling set for* d*. The only enabling action is* c *and it enables a private precondition of* d *that is not visible to agent* 1*. Hence, the NES would be empty. (As we have already seen, computing the dependent actions is error prone for the same reasons.)*

Algorithm 2: Strong stubborn set computation of agent i for state s (revised, complete)

```
  Input: $\Pi = \langle N, V, s_0, s_\star, \{A_j\}_{j \in N}\rangle$, state $s$
  Result: strong stubborn set $T_s \subseteq A$
1 $T_s \leftarrow \{a \in A_i^{pub} \mid a|_{V^{pub}}$ appl. in $s\}$
2 repeat
3     forall $a \in T_s$ do
4         if $a \in app(s)$ then
5             $T_s \leftarrow T_s \cup (\mathrm{dep}(a) \cap A_i)$
6         else
7             $T_s \leftarrow T_s \cup (N_s^a \cap A_i)$ for some NES $N_s^a$
8 until $T_s$ reaches a fixed-point
9 return $T_s$
```

Again, we observe that the inclusion of other agents' (publicly projected) actions during the strong stubborn set computation is problematic. For these actions, the agent cannot compute disjunctive action landmarks or neccessary enabling sets correctly. Simply excluding other agents' actions from the stubborn set computation, as the solution to the first example suggests, does not resolve the entire issue. In Example 2, agent 1 has no action to satisfy a goal condition, hence, the initial DAL would be empty when simply omitting agent 2's actions.

Having other agents' actions in the stubborn sets is only a superficial cause of failure, while the real problem is deeper. The ultimate cause is that public actions, which are part of a plan, are pruned. States created by public actions resemble potential interaction points between the agents. An agent cannot decide, whether these states are part of a plan leading to a goal or not, because they have only partial knowledge of the other agents' actions. Consequently, we need to alter the definition of strong stubborn sets in the context of privacy preserving planning. Instead of requiring the stubborn sets for state s to contain a disjunctive action landmark for a goal condition (Definition 10, point 3), we constrain them to contain the set of all public actions that are reachable from s by a (potentially empty) sequence of private actions. This definition of strong stubborn sets ensures that for every sequence of actions π, starting in the current state s and ending with a public action, a permutation of π exists which is not pruned. All successor states created by public actions are therefore preserved. Furthermore, we can now savely restrict the stubborn set computation to include only the agents' own actions.

Consider the following, revised definition of strong stubborn sets:

Definition 11 (Strong stubborn set for privacy preserving planning)**.** *Let* $(\Pi, \mathcal{P})$ *be a privacy preserving planning task and let* s *be a state of* Π. *A* strong stubborn set *for agent* i *in* s *is an action set* $T_s \subseteq A_i$ *such that:*

1. *For each* $a \in T_s \cap app(s)$*, we have* $\mathrm{dep}(a) \cap A_i \subseteq T_s$.
2. *For each* $a \in T_s \setminus app(s)$*, we have* $N_s^a \cap A_i \subseteq T_s$ *for some necessary enabling set* N_s^a *of* a *in* s.
3. T_s *contains all actions* $a \in A_i^{pub}$*, such that* $a|_{V^{pub}}$ *is applicable in* s.

Note how the first two constraints are only subtly different from Definition 10 restricting the included actions to belong to agent i only, while the third constraint ensures that all interaction points are preserved. Algorithm 2 computes stubborn sets consistent with the above definition.

Consider Example 1 again. According to Algorithm 2 each agent initially adds its own public action (line 1). Since they are both applicable in the initial state, all dependent actions are added in the next step (line 4, 5). There are none, hence, the computation reaches a fixed point and the stubborn sets are returned: $T_{00} = \{a\}$ for agent 1 and $T_{00} = \{b\}$ for agent 2. We end up with case (1) depicted in Figure 1. Similarly, in Example 2 agent 1 adds actions a and b to the initial strong stubborn set. Since these two actions are both applicable in the initial state and there are no dependent actions, the computation finishes returning $T_{000} = \{a, b\}$.

In both examples completeness is retained at the expense of pruning capacity. Since an agent cannot savely prune public actions, the pruning potential is restricted to permutations of private action sequences. We now discuss some theoretical properties of the presented stubborn set pruning technique.

Privacy

SSS for privacy preserving planning strives to reduce each agents individual search space without introducing any additional communication. It never transmits a state that is not transmitted by the respective planning algorithm without SSS pruning. We therefore believe that the presented technique is strongly privacy preserving.

Optimality

First, we define the terminology used in the proof.

Definition 12 (Public step)**.** *A* public step *in state* s *is a sequence of actions* πa*, where*

- a *is a public action of agent* i *and*
- π *is a* minimal plan *from* s *to* $\mathrm{pre}(a)$*, i.e.* $\pi[s] \models \mathrm{pre}(a)$*, that consists of private actions of agent* i *only.*

A plan π *from* s *to* $pre(a)$ *is* minimal*, if there is no subsequence* π'' *of* π *that can be moved behind action* a*, such that* $\pi a[s] = \pi' a \pi''[s]$*, where* π' *is the sequence* π *without* π''.

A public step can be thought of as a sort of "macro action" that encapsulates the execution of private actions followed by a single public action.

Definition 13 (Public state)**.** *A state* s *is called* public state *if it is reachable from the initial state by a sequence of public steps.*

Lemma 1. *Let* $(\Pi, \mathcal{P})$ *be a privacy preserving planning problem and* $\pi = (a_1, a_2, \ldots, a_k)$ *be a solution to* Π. *Then, there exists a permutation* $\pi' = (a'_1, a'_2, \ldots, a'_k)$ *of* π*, such that for all pairs of consecutive public actions*[1] a'_i, a'_j *in* π'*,* $(a'_{i+1}, a'_{i+2}, \ldots a'_j)$ *is a public step.*

Proof. Let $\pi = (a_1, a_2, \ldots, a_k)$ be a solution to Π, such that every private action in π is followed by another action (public or private) of the same agent. Only considering solutions of this type preserves optimality and completeness (Nissim and Brafman 2014). Assume that, between two consecutive public actions a_i and a_j we have a sequence of actions (of the same agent) $\pi_{i..j} = (a_{i+1}, a_{i+2}, \ldots, a_j)$ that is not a public step. Then, there must be a subsequence in $\pi_{i..j}$ that can be moved behind a_j. By moving this subsequence behind a_j, just before the next sequence of actions of the same agent, we create a permutation π'' that is a legal plan. Repeating this process until all inconsistencies have been removed yields a plan π' that is a permutation of π and that consists of public steps only. $\square$

Lemma 2. *Restricting the successor generation to a SSS (according to Def. 11) in every state is optimality preserving for privacy preserving planning.*

Proof. Let $(\Pi, \mathcal{P})$ be a privacy preserving planning task. The proof is by induction over $k \in \mathbb{N}$, where S_k is the set of public states that are reachable in at most k public steps from the initial state and S'_k is the set of public states that are reachable in at most k public steps when stubborn set pruning is applied. We show that $S_k = S'_k$ for all k. (It suffices to consider public states instead of all possible states because of Lemma 1.)

The initial state s_0 is reachable by an empty sequence of actions (zero public steps), therefore, $S_0 = \{s_0\} = S'_0$.

Let the set of reachable states expand from S_{k-1} to $S_k \supset S_{k-1}$. For each new state $s^* \in S_k \setminus S_{k-1}$, a state $s \in S_{k-1}$ must exist from which s^* is reachable, in a single public step. Therefore, there must be a public state $s \in S_{k-1}$ and a public step πa, such that $\pi a[s] = s^*$. Let i be the agent, such that $a \in A_i^{pub}$.

According to the induction hypothesis $S_{k-1} = S'_{k-1}$, it holds that $s \in S'_{k-1}$. We argue that SSS preserves a public step (of agent i) σa, such that $\sigma a[s] = s^*$. Observe that a is included in T_s for agent i since $a \in A_i^{pub}$ and its public projection $a|_{V^{pub}}$ is applicable in s (Definition 11, point 3).

If a is applicable in s, i.e. $a(s) = s^*$, then $s^* \in S'_k$. If a is not applicable in s, then a necessary enabling set for a must be contained in T_s (Definition 11, point 2). That is, a

[1]By *consecutive public actions* we mean that there are no other public actions between a'_i and a'_j. There might be private actions in between, however.

	blind		goalcount		FF	
Domain	def	sss	def	sss	def	sss
blocksworld	0	0	1	1	0	**1**
depot	2	2	**6**	4	0	0
driverlog	7	7	**17**	16	16	16
elevators	**3**	2	20	20	12	**14**
logistics	3	3	**18**	14	**17**	15
rovers	20	20	19	**20**	**20**	18
satellites	3	3	20	20	**20**	19
sokoban	**2**	0	2	**4**	7	7
taxi	6	**8**	11	**13**	2	2
wireless	0	0	0	0	**2**	1
woodworking	**2**	1	**2**	1	**2**	1
zenotravel	5	5	**20**	16	**16**	14
prod. site	0	**20**	11	**20**	8	**20**
Total	53	**71**	147	**149**	122	**128**

Table 1: Benchmark results.

disjunctive action landmark for $pre(a)$ in s. The stubborn sets generated for s according to Definition 11 correspond to the stubborn sets generated for s according to Definition 10 when planning towards the goal $s_\star = pre(a)$ with the set of actions $A = A_i$. Since strong stubborn sets consistent with Definition 10 are optimality and completeness preserving (Alkhazraji et al. 2012), a permutation σ of π must be preserved, such that $\pi a[s] = s^* = \sigma a[s]$. Hence $s^* \in S'_k$. □

Evaluation

The presented algorithms were implemented in a distributed multi-agent planning system written in Go. Experiments were run on a 2.6 Ghz Intel Xeon 8-core CPU. Each problem instance used a single core and 8 GB of RAM, shared by all agents.

We experimented with the benchmarks from the CoDMAP competition (Štolba, Komenda, and Kovacs 2015) consisting of 12 domains with 20 problems each. Of the new *production site* domain 20 problem instances of varying difficulty were included in the benchmarks. Planning time was limited to 30 minutes per problem instance. Table 1 shows coverage results for the tested configurations, while Figure 2 shows the running time for the production site instances.

Production site domain. While plain MAFS solves 0, 11 and 8 instances of the production site domain when using the blind, goalcount and FF heuristic (Hoffmann and Nebel 2001) respectively, MAFS with stubborn set pruning solves all 20 instances, independent of the heuristic used.

Blind MAFS resembles depth-first search and chains together random sequences of actions, most of which do not lead to a goal. Due to the expansive search space, even the easiest instances cannot be solved.

The goalcount and FF heuristics, on the other hand, both guide MAFS towards states with as many subgoals satisfied as possible. That way, the search focuses on one subgoal, or product, after the other and the number of generated states is reduced decisively. Although this behaviour seems to be favourable, it has its own shortcomings. The heuristics cannot differentiate between two states in which the same number of subgoals are satisfied, even if one state is significantly closer to satisfying another subgoal than the other. The reason for this is that the heuristics are computed based on each agent's local view. The public actions of other agents establish a subgoal (finish a product) with a cost of one and appear to always be applicable, because their public projections do not include their private preconditions. Because of this heuristic inaccuracy, states that satisfy a larger number of subgoals but which do not lead to a goal are preferred to states that lead to a goal but satisfy fewer subgoals. Processing actions, for instance, cannot be undone. Therefore, if a product is processed in a way not consistent with its goal requirements, the agent cannot finish that product. The respective subgoal can then only be supplied by another agent. If no agent can supply the subgoal, the search has to backtrack to a state in which the faulty processing action has not been applied yet.

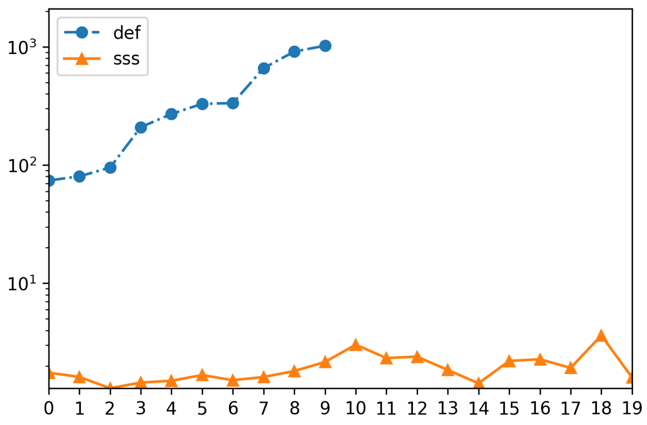


Figure 2: Runtime in seconds for increasingly difficult instances of the production site domain. Both configurations used the goalcount heuristic.

This problem does not occur in the stubborn set pruning variant. Counter-productive processing actions that prevent a product from being finished are always pruned. These actions are independent of the other processing actions and therefore may only be included in the stubborn set if they establish a precondition of the public action that finishes the product. Stubborn set pruning therefore effectively restricts the search to consider only such states that can be extended into a goal state. When FF or goalcount heuristic is used, the stubborn set approach also focuses on one subgoal after the other. The generated plans encourage the division of labor between the agents, each creating a subset of the products, rather than one agent creating them all. Furthermore, plans are found very fast, as all parts of the search space that do not progress towards a goal are pruned. Figure 2 highlights this fact.

CoDMAP domains. Regarding the CoDMAP domains, the results are mixed. There are no major differences in coverage between the strong stubborn set approach and regular MAFS, although overall the latter configuration solves a few problems more. We believe that this is due to the additional computations required for computing the stubborn sets. Interestingly, these benchmarks seem not to benefit from the stubborn sets based partial order reduction at all.

A possible explanation is that these domains already internalize a form of POR by decoupling the planning task in such a way that each agent has their own individual responsibilities. If in the production site domain each agent had a single processing action only, there would be as good as no pruning potential. This is exactly what we find in some of the CoDMAP domains. The *woodworking* domain is a good example of such an agent decoupling. Here, most of the agents can only perform a single action.

Another explanation is that the pruning potential cannot be exploited, because the agents compute their stubborn sets independent of one another. Therefore, they might end up generating more states than necessary, similar to the first case of Figure 1. Investigating how to get the agents' pruning efforts more in sync seems to be worthwhile.

Conclusion

This paper provides a theoretical basis for stubborn sets pruning in the context of privacy preserving planning. The empirical results show that some domains significantly benefit from partial order reduction. Although the production site domain was created with partial order reduction in mind, we believe that it models a specific situation that can also occur within the search space of other domains. In this situation, the heuristics are blind or misleading and, in consequence, the search exhaustively explores the affected parts of the search space. When these parts consist of many independent actions, then stubborn sets pruning can significantly reduce the search effort.

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In *Proc. ECAI*, 891–892.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *IJCAI*, 1530–1536.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proc. AAAI*, volume 8, 944–949.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proc. AAMAS*, 1265–1266.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *JAIR* 51:293–332.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proc. ICAPS*.

Štolba, M.; Fišer, D.; and Komenda, A. 2015. Admissible landmark heuristic for multi-agent planning. In *Proc. ICAPS*.

Štolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners (CoDMAP). In *Proc. WIPC*, 24–28.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proc. Petri Nets*, 491–515. Springer.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In *ICAPS*.