Planning-Based Approach for Silent Proactive Assistance

José G-Barroso, Raquel Fuentetaja, Susana Fernández

Department of Computer Science and Engineering
Universidad Carlos III de Madrid,
Avda. de la Universidad 30, 28911,
Leganés, Madrid, Spain
jgbarros@pa.uc3m.es, {rfuentet, sfarregu}@inf.uc3m.es

Abstract

This paper focuses on multi-agent systems where the goal of some agents is to provide assistance to some other agents in a silent and proactive way. We assume contexts in which explicit inter-agent communication can not take place, so that the kind of assistance provided is non-cooperative and, therefore, non-intrusive. Agents are not aware that there are assistants. Assistant agents perform changes in the environment that are then perceived by the other agents, which can modify their original plan accordingly. We propose a decision algorithm for assistant agents based on classical planning, where the assisted agent behaviour is simulated to determine how the assistant can open an opportunity for it to improve its plan. To evaluate the approach we have defined several new planning domains where silent proactive assistance can be useful. Results show that the approach is effective with some limitations that depend on domain characteristics.

Introduction

A recurrent framework in the area of autonomous systems consists on two or more agents acting in a shared environment. We are interested in tackling the problem of **proactive assistance**, where an agent **S** (say, a supporter or assistant agent) tries to help an agent **P** (say, a prime agent), while **P** is unaware of **S**'s purpose. In this area, communication is an important tool for the different agents to coordinate with each other (Geib et al. 2016). However, it may sometimes not be possible. For example, when we need to help elderly and the communication is not precise, or when we need to assist busy people, who are performing tasks that require their full attention, without being too intrusive. In contrast to those approaches, in our case there is no communication between agents. Then, the assistance would be based on changes made by **S** in the shared environment, so that **P** can observe these changes and act consequently to improve its plan. On the other hand, other works that carry on proactive assistance where communication is not possible also apply techniques to recognise the objective/goal of the assisted agent (goal recognition). However, these works focus more on recognising goals correctly rather than creating a generic assistant agent able of helping to achieve such goals properly (Freedman and Zilberstein 2017). For this reason, we focus on reducing **P**'s total cost instead of recognising its goals and action model. We assume that they are known by **S**. Inferring them would be complementary to our approach and can be done in a preliminary phase.

Moreover, **S**'s cost function is important. Among all possible ways to help, we are interested in those that require also the lowest possible cost for the assisting agent (but prioritizing always the assisted agent's cost) as this cost could represent energy cost, amount of resources, etc.

In order to carry out a deliberative and long-term reasoning we apply Automated Planning (AP). But, the success of the assistance depends not only on the way in which **S**'s actions are chosen, but also on characteristics of the domain. Thus, we aim to study different domains and problems in order to empirically evaluate our approach. For this purpose, we have: (1) built an assisting agent (**S**) based on AP, (2) defined several new planning domains where a silent proactive assistance is viable, and (3) evaluated the assistance provided by that assisting agent in those domains.

Background

Classical Planning

A classical planning task can be defined as a tuple $\Pi = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{C} \rangle$; where $\mathcal{F}$ is a set of fluents; $\mathcal{A}$ is a set of actions with precondition, add and delete lists $pre(a) \subseteq \mathcal{F}$, $add(a) \subseteq \mathcal{F}$, and $del(a) \subseteq \mathcal{F}$, respectively; $\mathcal{I} \subseteq \mathcal{F}$ is the initial situation; $\mathcal{G} \subseteq \mathcal{F}$ is a set of goals; and $\mathcal{C}$ is a non-negative cost function.

A planning task Π defines a state model which states s are subsets of $\mathcal{F}$. In this model, the initial state is $\mathcal{I}$ and the goal states are those that include the goals $\mathcal{G}$. The actions a applicable in a state s, denoted as $A(s)$, are those for which $pre(a) \subseteq s$. The transition function γ, defined for applicable actions, specifies the resulting state of applying an action $a \in A(s)$ in a state s as $s' = \gamma(s, a) = (s \backslash del(a)) \cup add(a)$.

A solution or plan for Π is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces a state sequence $\langle s_0, \ldots, s_n \rangle$ such that $s_0 = \mathcal{I}$ and, for each i such that $1 \leq i \leq n$, a_i is applicable in s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$. A plan π solves Π if $\mathcal{G} \subseteq s_n$. Each action a has non-negative cost $\mathcal{C}(a)$, so that the cost of a plan is $\mathcal{C}(\pi) = \sum_{i=1}^{n} \mathcal{C}(a_i)$. A plan π is optimal if it has minimum cost. An optimal plan is represented by π^*.

Problem Framework

We consider two agents, **S** (Supporter) and **P** (Prime), acting in a shared environment. While the objective of **P** is to achieve its goals with minimum cost, the purpose of **S** is just to proactively assist **P** on reaching its objectives at a lower cost than acting alone.

In our setting, **P** is a rational agent who knows only its own planning task. However, it is not aware that another agent wants to help it. On the other hand, **S** knows its own actions model, **P**'s planning task and how **P** plans, i.e. the algorithm that it uses to plan. We define both agents using the classical planning notation.

Definition 1 (Agents) **S** *has an action model* $M_{\mathbf{S}} = \langle \mathcal{F}, \mathcal{A}_{\mathbf{S}}, \mathcal{I}, \mathcal{C}_{\mathbf{S}} \rangle$. **P** *has an associated planning task* $\Pi_{\mathbf{P}} = \langle \mathcal{F}, \mathcal{A}_{\mathbf{P}}, \mathcal{I}, \mathcal{G}_{\mathbf{P}}, \mathcal{C}_{\mathbf{P}} \rangle$. *Both agents share a common environment with states s, such that* $s \subseteq \mathcal{F}$.[1]

At every time step, both agents execute an action. We consider an interleaved execution, where the first action from the initial state $\mathcal{I}$ is executed by **S**. Agents interleave their actions until $\mathcal{G}_{\mathbf{P}}$ is achieved (infinite loops may occur). We assume both agents can execute a *noop* action with a cost of zero, which represents doing nothing ($pre(noop) = add(noop) = del(noop) = \emptyset$). [2]

Definition 2 (Interleaved execution) *Given an environment state s, the execution of an applicable joint action* $(a_{\mathbf{S}}, a_{\mathbf{P}})$*, where* $a_{\mathbf{S}} \in A_{\mathbf{S}}(s)$ *(i.e.* $a_{\mathbf{S}}$ *is applicable in s) and* $a_{\mathbf{P}} \in A_{\mathbf{P}}(a_{\mathbf{S}}(s))$ *(i.e.* $a_{\mathbf{P}}$ *is applicable in the state generated from the execution of* $a_{\mathbf{S}}$*), generates a new state* s''*, such that* $s' = a_{\mathbf{S}}(s)$ *is the intermediate state and* $s'' = a_{\mathbf{P}}(s')$.

Every agent has full observability. Although **P** ignores the existence of the other agent, it can observe the changes that **S** causes in the environment. The agents decide what action to execute reasoning from the state reached by the previous action performed by the other agent. The joint execution of actions is deterministic, but from the point of every agent, the execution of its actions is non deterministic, since there is another agent acting in the same environment.

We assume that **P** is a rational agent that always executes the first action of an optimal plan that solves $\Pi_{\mathbf{P}}$ from the current state s at each time step. **P** only executes *noop* when it does not find a plan from the current state.

Definition 3 (Silent Proactive Assistance (SPA) problem) *A Silent Proactive Assistance problem is defined by a tuple* $\langle M_{\mathbf{S}}, \Pi_{\mathbf{P}} \rangle$*, where* $M_{\mathbf{S}}$ *is the action model of* **S** *and* $\Pi_{\mathbf{P}}$ *is the planning task of* **P**.

Definition 4 (SPA problem solution) *A solution to a given SPA problem* $\langle M_{\mathbf{S}}, \Pi_{\mathbf{P}} \rangle$ *defines the sequence of actions* $a_{\mathbf{S}_i}$ *that* **S** *executes at every time step i, such that:*

a) the consecutive execution of the joint actions $(a_{\mathbf{S}_i}, a_{\mathbf{P}_i})_{i=1}^{n}$ *from the initial environment state, where the sequence of actions* $\boldsymbol{\alpha}_{\mathbf{P}} = \langle a_{\mathbf{P}_i} \rangle_{i=1}^{n}$ *constitutes the rational response of* **P** *and* $\boldsymbol{\alpha}_{\mathbf{S}} = \langle a_{\mathbf{S}_i} \rangle_{i=1}^{n}$ *is the sequence of actions of* **S***, generates a state* s_{2n} *containing* **P***'s goals,* $\mathcal{G}_{\mathbf{P}} \subseteq s_{2n}$ *; and*

b) the cost for **P** *is reduced w.r.t. the cost of its optimal plan if it was acting alone in the environment (i.e.* $\forall i, a_{\mathbf{S}_i} =$ *noop):*

$$\mathcal{C}_{\mathbf{P}}(\alpha_{\mathbf{P}}) < \mathcal{C}_{\mathbf{P}}(\pi^*_{\mathbf{P}})$$

In case there is no $\pi^*_{\mathbf{P}}$, the assistance problem will only consist on achieving $\mathcal{G}_{\mathbf{P}}$, and $\mathcal{C}_{\mathbf{P}}(\pi^*_{\mathbf{P}}) = \infty$. The optimal solution to a SPA problem, ${\alpha_{\mathbf{S}}}^*$, is a solution with minimum cost for **S** among those tha minimize **P**'s cost.

Centralized Planning

The assumption that **P** is not aware that there is an assistant agent trying to help it, makes it impossible to apply centralized planning-based approaches that decide the behaviour of both agents. However, solutions provided by these approaches can serve as an indicator of what is the best behaviour that could be achieved, which is interesting from an experimental point of view.

A joint planning task can be posed in two ways, either defining an action model that considers directly joint actions or defining an action model considering the individual actions of both agents and alternate turns. In both cases, the planning task to solve can be defined using some ideas of Multiple Cost Function (MCF) planning tasks (Katz et al. 2019), but with the peculiarities that there is a secondary cost function to be also minimized and that there is a unique bound, which is established over the primary cost function.

Thus, the joint planning task would be $\Pi_{joint} = \langle \mathcal{F}, \mathcal{A}_{joint}, \mathcal{I}, \mathcal{G}_{\mathbf{P}}, \mathcal{C}_o, \mathcal{C}_1, \mathscr{C} \rangle$, where:

- $\mathcal{A}_{joint} = \{a \in \mathcal{A}_{\mathbf{S}} \times \mathcal{A}_{\mathbf{P}} \mid del(a_{\mathbf{S}}) \cap pre(a_{\mathbf{P}}) = \emptyset\}$, and its actions are defined assuming that **S** executes first: [3]
  - $pre(a) = pre(a_{\mathbf{S}}) \cup (pre(a_{\mathbf{P}}) \setminus add(a_{\mathbf{S}}))$
  - $add(a) = add(a_{\mathbf{P}}) \cup (add(a_{\mathbf{S}}) \setminus del(a_{\mathbf{P}}))$
  - $del(a) = del(a_{\mathbf{P}}) \cup (del(a_{\mathbf{S}}) \setminus add(a_{\mathbf{P}}))$
- $\mathcal{C}_0 = \mathcal{C}_{\mathbf{P}}$ is a primary cost function,
- $\mathcal{C}_1 = \mathcal{C}_{\mathbf{S}}$ is a secondary cost function, and
- $\mathscr{C} = \{(\mathcal{C}_0, \mathcal{C}_{\mathbf{P}}(\pi^*_{\mathbf{P}}))\}$, imposes an upper bound on the primary cost of plans and the cost of **P**'s optimal plan when acting alone.

The primary cost of a joint action $(a_{\mathbf{S}}, a_{\mathbf{P}})$ is $C_{\mathbf{P}}(a_{\mathbf{P}})$ and its secondary cost is $C_{\mathbf{S}}(a_{\mathbf{S}})$. The secondary cost function would be eliminated if the cost of **S** is not relevant. In that case, this model would still be useful from an assistance point of view.

An operator sequence π^*_{joint} is an optimal plan for Π_{joint} if its application generates $\mathcal{G}_{\mathbf{P}}$, minimizes $\mathcal{C}_0(\pi^*_{joint})$, minimizes $\mathcal{C}_1(\pi^*_{joint})$ over all solutions with minimum $\mathcal{C}_0(\pi^*_{joint})$ and it is consistent with $\mathcal{C}_0(\pi^*_{joint}) < \mathcal{C}_{\mathbf{P}}(\pi^*_{\mathbf{P}})$.

The joint planning task considers all compatible actions resulting from the cartesian product. An equivalent alternative with a smaller number of actions, which also reduces the

[1] An action model here is like a planning task but with no goals.

[2] This assumption can be relaxed, but in that case it would be necessary to define what the agents do when they have no plan.

[3] When $del(a_{\mathbf{S}}) \cap pre(a_{\mathbf{P}}) \neq \emptyset$ the actions are not compatible.

branching factor when planning, is to include turns. Thus, $\Pi_{turn} = \langle \mathcal{F} \cup \{t_{\mathbf{S}}, t_{\mathbf{P}}\}, \mathcal{A}_{turn}, \mathcal{I} \cup \{t_{\mathbf{S}}\}, \mathcal{G}_{\mathbf{P}}, \mathcal{C}_o, \mathcal{C}_1, \mathscr{C} \rangle$, where:

- The fluents t_X, $X \in \{\mathbf{S}, \mathbf{P}\}$, indicate that it is the turn of the X agent;
- $\mathcal{A}_{turn} = \mathcal{A}'_{\mathbf{S}} \cup \mathcal{A}'_{\mathbf{P}}$, where $\mathcal{A}'_X$, $X \in \{\mathbf{S}, \mathbf{P}\}$, are just the actions of the agents modified to make a turn change, i.e. $pre(a'_X) = pre(a_X) \cup \{t_X\}$, $add(a'_X) = add(a_X) \cup \{t_Y\}$, Y being the other agent, and $del(a'_X) = del(a_X) \cup \{t_X\}$; and
- $\mathcal{C}_o$, $\mathcal{C}_1$ and $\mathscr{C}$ are analogous to those defined before for Π_{joint}, but the primary cost of **S**'s actions and the secondary cost of **P**'s actions are both zero.

We have chosen to solve the problem using classical planners. This can be done whenever both the output plans and the ranking between them established by the cost functions coincide with those of the defined problem. The former is achieved using the bound $\mathscr{C}$ to prune the paths where **P**'s cost exceeds that value. To achieve the latter we define a unique linear cost function, $\mathcal{C} = \omega \times \mathcal{C}_{\mathbf{P}} + \mathcal{C}_{\mathbf{S}}$, with $\omega \geq 1$. $\mathcal{C}$ maintains the ranking between plans if for any two plans, π and π':

a) It establishes the same ranking as the **P**'s cost function : $\mathcal{C}_{\mathbf{P}}(\pi) < \mathcal{C}_{\mathbf{P}}(\pi') \implies \mathcal{C}(\pi) < \mathcal{C}(\pi')$; and

b) It solves ties in the **P**'s cost function by considering the **S**'s cost function: $(\mathcal{C}_{\mathbf{P}}(\pi) = \mathcal{C}_{\mathbf{P}}(\pi') \wedge \mathcal{C}_{\mathbf{S}}(\pi) \leq \mathcal{C}_{\mathbf{S}}(\pi')) \implies \mathcal{C}(\pi) \leq \mathcal{C}(\pi')$.

Condition b) always holds with the defined linear function. The only problematic case for condition a) is when $C_{\mathbf{P}}(\pi) < C_{\mathbf{P}}(\pi')$, $C_{\mathbf{S}}(\pi) > C_{\mathbf{S}}(\pi')$ and $C_{\mathbf{S}}(\pi) - C_{\mathbf{S}}(\pi') \geq C_{\mathbf{P}}(\pi') - C_{\mathbf{P}}(\pi)$, because in that case the suppporter costs invert the correct ranking. Then, w should be high enough to guarantee that $C_{\mathbf{S}}(\pi) - C_{\mathbf{S}}(\pi') < \omega(C_{\mathbf{P}}(\pi') - C_{\mathbf{P}}(\pi))$. Theoretically:

$$w > \frac{\max\limits_{\pi,\pi'}(C_{\mathbf{S}}(\pi) - C_{\mathbf{S}}(\pi'))}{\min\limits_{\pi,\pi'}(C_{\mathbf{P}}(\pi') - C_{\mathbf{P}}(\pi))} \quad (1)$$

Obviously, the value of ω depends on the specific problem to solve and its possible plans. In our experiments we have selected it empirically considering the specific problems in our benchmarks.

Solving the SPA Problem

We adopt a planning-replanning strategy at execution time. Other approaches considering policies would also be possible. At every time step every agent decides what action to execute. We assume **S** decides first, and then **P** takes its decision from the resulting state. As defined before, the behaviour of **P** is rational: it takes the first action of an optimal plan from the current state to the goals. If no plan exists it will execute *noop*. In a first approach to the problem, we assume **S** is able of reproducing the decision algorithm of **P**, so that it can compute its exact optimal plan for a given state.

Under the defined conditions, the decision algorithm for **S** can be seen as a search process similar to the one for solving Π_{turn}, with the difference that in **P**'s turn there is only one successor generated by the first action of **P**'s optimal plan, instead of branching considering all **P**'s applicable actions. This would guarantee the optimal solution to the SPA problem, but involves to compute the optimal plan of **P** for every node generated by **S**'s actions (in the worst case, because it could sometimes reuse previous optimal plans), which is too expensive computationally.

In the following sections we propose different and more feasible alternatives based on planning to approximate the solution of the problem. We focus on the reasoning process of **S** to decide its next actions. The idea is to compile **P**'s planning task and **S**'s action model to a planning task which solution provides those next actions.

Based on Turns Approach (*BTA*)

One simple approximation for **S** to decide its actions consists on assuming that **P** will act optimally collaborating with it, as if **P** knew about **S**'s purpose. With this assumption, the reasoning process of **S** is not taking in consideration the real **P** behaviour (i.e. executing its current optimal plan without being aware that **S** wants to help it). But it can still allow to provide assistance in a reasonable time. In fact, SPAM (Single-Plan Action Matching) (Freedman and Zilberstein 2017) incorporates the *joint* compilation to determine the behaviour of the assisted agent. However, this compilation does not prevent getting into a loop of *noops* in execution time. These loops occurs if **S** executes *noop* when the last action of **P** was a *noop* too. To develop this idea, the compilation Π_{BTA} is proposed. Π_{BTA} is similar to Π_{turn}, but with the following changes to avoid infinite loops of *noops* in execution time:

- there is a new fluent $canNoop_{\mathbf{S}}$, which indicates that **S** can do *noop*,
- the current state contains $canNoop_{\mathbf{S}}$ when the previous action of **P** was different from *noop*,
- all **P**'s actions except *noop* add $canNoop_{\mathbf{S}}$, and
- $canNoop_{\mathbf{S}}$ is a precondition of **S**'s *noop*.

Optimal plans π^*_{BTA} from Π_{BTA} are centralised plans in which both agents act optimally to achieve **P**'s goals at minimum cost (preventing infinite loops of *noops* in execution time), even if that implies collaboration.

As we said before, in execution time **P** selects *noop* iff there is no a plan to achieve its goals. In order to avoid **P**'s *noops* in π_{BTA} and get closer to the real **P**'s behaviour we propose *BTA2*, which is as Π_{BTA} but the *noop* action for **P** has a very high cost ψ. However, if it is necessary for **S** to disable **P** at a certain state in order to assist it in the future (so that **P** can only execute *noops* from that state), *BTA2* will not be able to find such a assistance.

The main issue with these approaches is that they incorrectly approximate the behaviour of **P**. Also, *BTA* and *BTA2* are very similar to the centralized compilation in SPAM. For this reason we consider *BTA* as a baseline approach for comparison instead of a contribution of our work.

Compilation Based on Opportunities (*CBO*)

We propose the Compilation Based on Opportunities (*CBO*) with the purpose that **S** can reason according to a more accurate behaviour of **P**. The main idea of *CBO* is to compute **P**'s optimal plan only once, just for the current state, and then simulate its execution, interleaving its actions with **S**'s actions until an action of **S** opens an *opportunity* for **P** to improve its plan. After that, **S**'s actions can be inhibited at any time, so that only **P**'s actions can be applied to the end of the process.

Definition 5 (Opportunity) *A* $\mathbf{S}$*'s action* $a_{\mathbf{S}} \in \mathcal{A}_{\mathbf{S}}$ *opens an opportunity for* $\mathbf{P}$ *if there exists a* $\mathbf{P}$*'s action* $a_{\mathbf{P}} \in \mathcal{A}_{\mathbf{P}}$ *such that* $pre(a_{\mathbf{P}}) \cap (add(a_{\mathbf{S}}) \cup del(a_{\mathbf{S}})) \neq \emptyset$*; i.e if* $\mathbf{S}$*'s action adds/deletes a precondition of some* $\mathbf{P}$*'s action.*

We opted for a weak definition of *opportunity* that defines a potential opportunity without guarantees that it is in fact an opportunity for **P** to improve its cost.

Definition 5 allows to divide the set of **S**'s actions into two subsets, those openning opportunities, $\mathcal{A}_{\mathbf{S}}^{opp}$, and those not openning opportunities $\mathcal{A}_{\mathbf{S}}^{\neg opp}$, thus $\mathcal{A}_{\mathbf{S}} = \mathcal{A}_{\mathbf{S}}^{opp} \cup \mathcal{A}_{\mathbf{S}}^{\neg opp}$.

In *CBO*, the decision algorithm of **S** consists of the following steps: (1) compute **P**'s (optimal) plan $\pi_{\mathbf{P}}^{*} = a_1, \ldots, a_n$ from the current state; (2) solve optimally a planning task, Π_{CBO}, to carry out the reasoning described above; and (3) get $\alpha_{\mathbf{S}}$ from the solution of Π_{CBO} to determine the next **S**'s actions. In the following, **P**'s plan will be denoted simply as $\pi_{\mathbf{P}}$, since the fact that it is optimal could be relaxed to make the approach more efficient.

We define the planning task $\Pi_{CBO} = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}_{\mathbf{P}}, \mathcal{C} \rangle$ as follows:

- $\mathcal{F}' = \mathcal{F} \cup \{t_{\mathbf{S}}, t_{\mathbf{P}}, ph_1, en\_ph_2\} \cup \mathcal{F}_{next}^{\pi}$, where $t_X, X \in \{\mathbf{S}, \mathbf{P}\}$, represents the turn; ph_1 refers to phase 1 where **P**'s initial plan is being simulated, interleaving its actions with **S**'s actions; en_ph_2 indicates that phase 2 is enabled to start (or continue in case it has already started), i.e. the phase that only considers **P**'s actions; and $\mathcal{F}_{next}^{\pi} = \{next_{a_i} \mid 1 \leq i \leq |\pi_{\mathbf{P}}|\}$represents the next action of **P** to be simulated in phase 1.
- $\mathcal{I}' = \mathcal{I} \cup \{t_{\mathbf{S}}, ph_1, next_{a_1}\}$; and
- $\mathcal{A}' = \mathcal{A}_{\mathbf{S}}^{'\neg opp} \cup \mathcal{A}_{\mathbf{S}}^{'opp} \cup \mathcal{A}_{\mathbf{P}}^{ph_1} \cup \mathcal{A}_{\mathbf{P}}^{ph_2}$;

  Actions $a \in \mathcal{A}_{\mathbf{S}}^{'X}$, $X \in \{\neg opp, opp\}$, represent **S**'s actions and they are defined in terms of $a_{\mathbf{S}} \in \mathcal{A}_{\mathbf{S}}$:
  - $pre(a) = pre(a_{\mathbf{S}}) \cup \{t_{\mathbf{S}}\}$
  - $add(a) = \begin{cases} add(a_{\mathbf{S}}) \cup \{t_{\mathbf{P}}, en\_ph_2\} & \text{if } X = opp \\ add(a_{\mathbf{S}}) \cup \{t_{\mathbf{P}}\} & \text{otherwise} \end{cases}$
  - $del(a) = del(a_{\mathbf{S}}) \cup \{t_{\mathbf{S}}\}$
  - $\mathcal{C}(a) = \mathcal{C}(a_{\mathbf{S}})$

  Actions $a \in \mathcal{A}_{\mathbf{P}}^{ph_1}$ represent the (instantiated) actions a_i in **P**'s current plan:
  - $pre(a) = pre(a_i) \cup \{t_{\mathbf{P}}, ph_1, next_{a_i}\}$
  - $add(a) = \begin{cases} add(a_i) \cup \{t_{\mathbf{S}}\} & \text{if } i = |\pi_{\mathbf{P}}| \\ add(a_i) \cup \{t_{\mathbf{S}}, next_{a_{i+1}}\} & \text{otherwise} \end{cases}$
  - $del(a) = del(a_i) \cup \{t_{\mathbf{P}}, next_{a_i}\}$
  - $\mathcal{C}(a) = \omega \times \mathcal{C}(a_i)$

  Actions $a \in \mathcal{A}_{\mathbf{P}}^{ph_2}$ represent **P**'s actions $a_{\mathbf{P}} \in \mathcal{A}_{\mathbf{P}}$:
  - $pre(a) = pre(a_{\mathbf{P}}) \cup \{t_{\mathbf{P}}, en\_ph_2\}$
  - $add(a) = add(a_{\mathbf{P}})$
  - $del(a) = del(a_{\mathbf{P}}) \cup \{ph_1\}$
  - $\mathcal{C}(a) = \omega \times \mathcal{C}(a_{\mathbf{P}})$

When $\pi_{\mathbf{P}}$ is the empty plan, $\mathcal{F}_{next}^{\pi} = \emptyset$, i.e. the planning task Π_{CBO} does not contain fluents of type $next_{a_i}$.

In general, plans for Π_{CBO} are composed of a sequence of actions of **S** interleaved with a sequence of actions of **P** (phase 1) followed by a sequence of actions of **P** (phase 2). We denote the complete sequence of **P**'s actions (including both phases) extracted from a solution plan as $\hat{\alpha_{\mathbf{P}}}$.

When an optimal plan, π_{CBO}^{*}, for Π_{CBO} is computed, three different scenarios can occur:

1. No assistance has been found. This occurs when π_{CBO}^{*} consists of **P**'s actions of the computed initial plan interleaved with *noop* actions of **S**. Then, $\mathcal{C}(\pi_{CBO}^{*}) = \mathcal{C}(\hat{\alpha_{\mathbf{P}}}) = \omega \times \mathcal{C}(\pi_{\mathbf{P}})$. If $\pi_{\mathbf{P}}$ is also the empty plan, π_{CBO}^{*} is also the empty plan.
2. π_{CBO}^{*} contains a possible assistance, i.e. $\mathcal{C}(\hat{\alpha_{\mathbf{P}}}) < \omega \times \mathcal{C}(\pi_{\mathbf{P}})$, and π_{CBO}^{*} is composed of **S**'s actions belonging to $\mathcal{A}_{\mathbf{S}}^{'opp}$ and to $\mathcal{A}_{\mathbf{S}}^{'\neg opp}$ and some **P**'s actions of its initial plan $\mathcal{A}_{\mathbf{P}}^{ph_1}$. This means that the assistance is provided by achieving directly some of the goals of **P**.
3. π_{CBO}^{*} contains a possible assistance, i.e. $\mathcal{C}(\hat{\alpha_{\mathbf{P}}}) < \omega \times \mathcal{C}(\pi_{\mathbf{P}})$, and the last action in π_{CBO}^{*} belongs to $\mathcal{A}_{\mathbf{P}}^{ph_2}$. This means that **S** has found a possible way to open an opportunity for **P** to improve its plan.

The sequence of actions for **S**, $\alpha_{\mathbf{S}}$, is updated directly from the solution plan π_{CBO}^{*}. When the action executed by **P** does not match with $\hat{\alpha_{\mathbf{P}}}$ or when all actions in $\alpha_{\mathbf{S}}$ have been already executed, **S** replans from the current state and update $\alpha_{\mathbf{S}}$. In this way, **S** will try to provide additional assistances instead of executing *noop* when $\alpha_{\mathbf{S}}$ has been already executed. This could only occur in the third scenario presented above.

The **P**'s behaviour considered by the presented approach tends to be more similar to the real one in comparison with *BTA*, since we assume **P** does not cooperate with **S**. **P**'s actions in the phase 1 of π_{CBO}^{*} belong to its original plan, and those of phase 2 only contains **P**'s actions. Therefore, the assistance is expected to be more effective than with *BTA*. However, the decision algorithm for **S** is focused on detecting the first opportunity for **P** to improve its plan instead of simulating completely **P**'s behaviour. It could be also the case that the obtained behaviour for **S** generates the opposite effect to the desired one. There is a trade-off between efficiency and effectiveness. The proposed approach improves efficiency at the expense of compromising effectiveness.

Domains

To test the approach, we generated 6 domains in PDDL in which Silent Proactive Assistance makes sense. For every domain there are two planning action models, one for **S** and

one for **P**. 4 domains were generated from scratch (teleport-assistance 1 and 2, and car-assistance 1 and 2), while the other 2 (visitall-assistance and termes-assistance) are based on IPC [4] domains. All of them are summarised below:

- **Teleport-assistance 1 (TEA1).** In this domain **P** has to reach a target cell in a grid with walls and teleports. A teleport is a device for changing the position of anything above it. **P** can move between adjacent positions. **S** can activate just one teleport as origin and another as destination. **S** can activate a teleport at any time with a single action. When both are activated, **S** can teleport **P** above the origin teleport to the destination teleport with another action. The assistance consists of teleporting **P** to a position closer to the goal.
- **Teleport-assistance 2 (TEA2).** This domain is a variant of TEA1 in which there is no destination teleport to be activated because the actual destination of the teleportation is **P**'s target cell. Also, when **S** activates the origin teleport the target cell is blocked and can only be accessed using teleportation. The assistance consists of teleporting **P** to the goal.
- **Car-assistance 1 (CA1).** In this domain **P** has to reach a target cell in a grid with walls. It can move either on foot (slow) or by car (fast). **S** has an action model analogous to **P**. However, the car can only be occupied by one of the two agents. The assistance consists of bringing the car closer to **P**, so that it can take it to get its goal faster.
- **Car-assistance 2 (CA2).** This domain is variant of CA1 in which both agents can be inside the car at the same time, but in that case only **P** can drive.
- **Visitall-assistance (VA).** This domain is a variant of the Visitall domain. The original Visitall domain consists of an agent who wants to visit all the nodes of a connected graph. **P**'s action model and its goals are equivalent to the original IPC domain. **S** is able to create unidirectional connections between nodes of the graph. There are problems where **S**'s actions are necessary for **P** to achieve its goals, since not all nodes are reachable from each node. The assistance consists of making it easier for **P** to traverse all the nodes of the network, preventing **P** from passing through the same node twice.
- **Termes-assistance (TA).** This domain is a variant of the Termes domain. The original Termes domain consists of a robot who wants to build a structure of identical blocks placing them at different positions and heights on a grid. To do that it can move between adjacent positions, create or destroy blocks in the depot, pick up or put down blocks at the same height as the robot, or climb/descend to a single block. In this way, the robot has to build a block ladder to place the blocks at different heights. **P**'s action model and goals are equivalent to the original IPC domain. **S** is a crane that can lift and deposit **P** (changing its position) or any block located in any cell. **S** is more efficient than **P**, because it does not need to move between cells in order to place blocks. The assistance consists of reducing **P**'s workload.

[4]The International Planning Competition (IPC) is a global competition where state-of-the-art planning systems are empirically evaluates on a number of benchmark problems.

Experiments

We execute ***CBO***, ***BTA***, ***BTA2*** **(with $\psi = 1000000$)** and **no assistance**, i.e. **S** just performs *noop*. The parameter ω used to prioritise the **P**'s cost over **S**'s cost is set to 1000 which is an upper bound for the minimum ω as defined in Equation 1. A lower bound for the minimum difference between **P**'s plans is 1, and an upper bound for the maximum difference between **S**'s plans is 1000 in all problems tested.

In **S**'s plans, consecutive *noop* (one from each agent) are eliminated because they are useless and to avoid identifying infinite loops incorrectly.

In our setting, **P** computes a new optimal plan only in case there exists $a_{\mathbf{P}} \in \mathcal{A}_{\mathbf{P}} : pre(a_{\mathbf{P}}) \cap (add(a_{\mathbf{S}}) \cup del(a_{\mathbf{S}})) \neq \emptyset$, where $a_{\mathbf{S}}$ is the last executed action by **S**. Otherwise, it continues with its previous optimal plan. On the other hand, **S** executes *noop* when it is not able to compute its plan.

The optimal planner used is *Fast Downward* (FD) (Helmert 2006) version 19.06 with A^* search and the *iPDB* heuristic (Haslum et al. 2007). Each approach was executed for every problem in a machine that consists of a 12-core Intel® Core™ i7-9750H, with 2.6 GHz clock speed, with 16 GBytes of RAM installed. We collected the following data:

- The sequence of **S**'s actions ($\alpha_{\mathbf{S}}$) and **P**'s actions ($\alpha_{\mathbf{P}}$) executed in the common environment.
- The total cost of each agent ($\mathcal{C}(\alpha_{\mathbf{S}})$ and $\mathcal{C}(\alpha_{\mathbf{P}})$).
- Total time to compute all **P** and **S**'s plans.

From these data, we extracted:

- The percentage of solved problems (i.e. **P**'s goal is reached).
- The percentage of problems where the assistance occurs.
- The mean and standard deviation of the percentage of **P**'s cost improvement in those problems where the assistance occurs (and when the cost without assistance is not infinity, i.e. when **P** can get its goals by itself).
- The mean and standard deviation of **S**'s cost in those problems where the assistance occurs.

Results

The obtained results are shown in Table 1 for each domain and algorithm. In this seciton we analyze them in more detail.

TEA1 is a domain where **S** provides the optimal assistance using *CBO*. That is because **S** knows which route **P** will take to the goal, and because **P** does not modify its original plan as a consequence of the changes made in the environment by **S** until the assistance has been successfully carried out, i.e. when **P** has been teleported by **S**. Figure 1 shows an example, the first image shows **P**'s original plan. **P**'s total cost of that plan is 7 considering unit costs. But **P**'s total cost using *CBO* is 5, since **S** knows which teleport **P** will step on (fourth image in Figure 1, where green represents the activated teleporters by **S** and red the ones that are

Table 1: Results about percentage of solved problems, percentage of problems where the assistance occurs, mean and standard deviation of the percentage of **P**'s cost improvement (when there is an assistance) and mean and standard deviation of **S**'s cost (when there is an assistance).

Domain	Approach	% Solved	% Assistance	Improvement ratio	**S**'s cost
TEA1	*CBO*	100%	100%	35.4% ± 17.8%	3.0 ± 0.0
(200)	*BTA*	100%	47%	40.2% ± 18.3%	3.0 ± 0.0
	BTA2	100%	69%	37.5% ± 18.1%	3.0 ± 0.0
TEA2	*CBO*	10%	0%	-	-
(200)	*BTA*	23%	23%	97.6% ± 11.6%	2.0 ± 0.0
	BTA2	15%	13%	100.0% ± 0.0%	2.0 ± 0.0
CA1	*CBO*	100%	100%	16.2% ± 5.1%	5.5 ± 2.2
(20)	*BTA*	100%	90%	16.0% ± 5.4%	5.9 ± 2.4
	BTA2	100%	100%	16.7% ± 5.0%	5.5 ± 2.2
CA2	*CBO*	100%	100%	19.9% ± 7.2%	4.7 ± 1.5
(20)	*BTA*	100%	100%	19.9% ± 7.2%	4.9 ± 1.6
	BTA2	100%	100%	19.9% ± 7.2%	4.7 ± 1.5
VA	*CBO*	100%	100%	13.1% ± 6.9%	1.3 ± 0.5
(50)	*BTA*	100%	94%	11.6% ± 6.7%	1.4 ± 0.5
	BTA2	100%	94%	11.6% ± 6.7%	1.4 ± 0.5
TA	*CBO*	100%	100%	53.8% ± 15.4%	3.8 ± 1.9
(50)	*BTA*	96%	96%	65.8% ± 10.5%	15.8 ± 7.0
	BTA2	100%	100%	60.8% ± 12.5%	6.4 ± 2.7

not). On the other hand, using the centralized approaches *BTA* and *BTA2*, **S** does not always activate the correct origin teleport. Whereas with *BTA* **S** always activates the closest one to the position of **P** (second image in Figure 1), *BTA2* tends to discard those located in adjacent cells to the initial **P**'s position (third image in Figure 1), as it does not consider that **P** will execute *noop* waiting for **S**'s actions. This implies that if the assistance occurs using *BTA*, it will also occurs using *BTA2*. For that reasons *BTA2* shows a better performance than *BTA* in this domain. In any of the three proposals, all successful assistances are optimal, because the chosen origin teleport is the closest to **P**'s position that **P** reaches and the chosen destination teleport is the closest to the goals.

In **TEA2** the only way to assist is waiting for **P** to get on a teleport, activating it and in the next turn teleporting **P** to the target cell. With *CBO*, **S** is never able to assist **P**. If **S** waits for **P** to act according to its initial plan, **S** would always activate the teleport just before **P** steps on it. However, **P** would stop just before reaching such a teleport (when the teleport is activated, the goal is blocked and **P** is unable to compute any plan). In the best scenario, **P** would be initially on a teleport and would not pass over any other teleport on its way to the goal. In that case, **S** would do nothing, mistakenly thinking that it is impossible to assist. On the other hand, using *BTA*, **S** will always activate the closest teleport to the position of **P** (in most cases in the first timestep), thinking that **P** will go there, which succeeds just in case **P** is on top of such a teleport when it is activated. Using *BTA2*, **S** supposes that **P** will not execute *noop*, even when the teleport is activated. For that reason, **S** does not always activate a teleport, as it considers **P**'s cost after **S**'s actions could end up being the same as without **S**'s actions. Also, if **P** is initially on a teleport, **S** could also activate such a teleport when **P** moves, mistakenly thinking that it is going to return to the same position, rather than from the first moment, making the assistance impossible. For these two reasons *BTA* shows a better performance than *BTA2* in this domain. As in TEA1, in any of the three proposals all successful assistances are optimal.

In **CA1**, if **S** got into the car, the car would become occupied and **P** would have to walk to the goal. With *CBO*, **S** reasons from a plan of **P** that fits better with the real **P**'s behaviour: before getting into the car, **S** knows where **P** is going (directly to the goals or to the car), because **S** can compute **P**'s plan. If **P** changes its plan after **S** occupies the car, **S** also replans, computing **P**'s plan to know where **P** is going (directly to the goals). With *BTA*, **S** supposes that **P** will stay still in its current position, waiting for **S** to place the car there. However **P** does not do that. Also, in some problems **S** is not able to assist. Specifically, in case moving between adjacent positions by car is necessary to reach **P**'s position, if those moves are placed at the beginning of **S**'s plan and **P** moves away from **S**'s position, **S** would be unable to reach **P** because it would drive too slow. In *BTA2*, since the use of *noop* is limited, **S** supposes that **P** goes towards an intermediate cell between the car and the current **P**'s position. This fact does not prevent the assistance even if it does not correspond to the actual plan of **P**, because **S** reduces the distance between **P** and the car. In case of assistance, the cost improvement is practically the same in all approaches. That is because the most important thing in this domain is that **P** reaches the car as soon as possible, and all algorithms achieve this in most cases.

In **CA2**, **P** is able to get into the car even if it is occupied. With any of the 3 approaches **S** goes towards the car and subsequently drives to reduce the distance between **P** and such a car. Eventually, **P** considers that walking towards the car is better than walking directly towards the goal. The assistance will occur, and the improvement will be very similar.

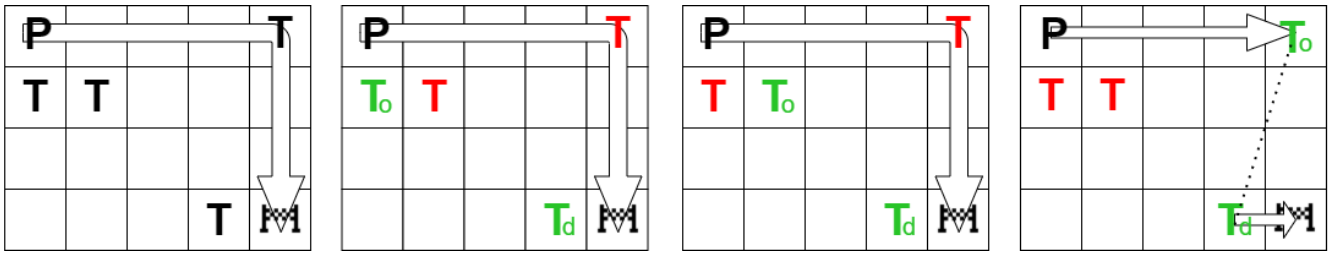

Figure 1: Executions in the environment of a **TEA1** problem using *no assistance*, *BTA*, *BTA2* and *CBO* repectively.

In **VA**, there are problems where **S**'s actions are necessary for **P** to achieve its goals, since not all nodes are reachable from each node. This happens in 39/50 problems. The three approaches solved all problems, since the only way to not achieve the goals is falling into an infinite loop of *noops*, but we avoid that in all our compilations. *CBO* has a good performance in comparison to the other two alternatives, since **S** is able to know the order of the nodes **P** will pass through after computing the plan of **P** and, therefore, which connections would have to create so that **P** does not pass through the same node twice. In all generated problems (where it is not necessary to create so many connections to achieve an optimal assistance), the cost of **P** is reduced as much as possible (**P**'s cost = number of nodes - 1), i.e. the optimal assistance is achieved. This may not occur in other problems. In centralized compilations (*BTA* and *BTA2*) the minimum cost is achieved in most of these problems, but in others assistance does not even occur. In these cases **S** executes *noop* waiting for **P** to move as **S** estimates, for the purpose of creating the required connections between nodes later. If **P** visits a node it has already visited the assistance may not occur.

In **TA**, agents can fall into loops, repeating actions infinitely. Despite this, the assistance occurs in most cases, since **S** can build the structure of blocks faster than **P**. This domain is the only one with problems that take several hours to execute them, considering all plans computed of both agents. In some problems the total execution time reached 10 hours, while the executions of **P** without assistance do not exceed 30 minutes in any problem. In an environment where both agents replan, the execution time increases significantly. Using *CBO* the cost improvement percentage is lower than the other two alternatives. This is mainly due to two reasons: (1) The number of actions that **S** can perform is very limited, since some of them force **P** to execute a different action from its original plan, which implies moving to phase 2. Therefore, it is not possible to decrease **P**'s cost much if to do so it is necessary to compute plans that consist of some of these actions. For example, the action of lifting **P** to change its position would never be executed, because that means disabling **P**, forcing it to execute *noop* and thus passing to phase 2. (2) It may be desirable to obtain plans that consist of a shorter phase 1, even if the cost is not reduced too much, if later, when **S** replans, it is possible to continue reducing **P**'s cost and thus perform a better assistance, instead of obtaining an unique plan that consists on reducing the cost as much as possible regardless of the duration of phase 1. For this reason, in *CBO*, **S** executes *noop* many times, looking forward to assisting later, hence the average of **S**'s cost is so small. In fact, in most cases, the assistance consists on lifting the last remaining block, so that **P** does not have to perform any action to remove it. Centralized plans are better, as they allow **S** to achieve **P**'s goals by itself, requiring **P** only for creating blocks. *BTA* obtains better results than *BTA2* because the second one never executes the action of lifting **P**, since it would force **P** to execute *noop*. Using *BTA*, **S** has a larger repertoire of actions in comparison with *BTA2*, which are used to perform a better assistance, that is why the average of **S**'s cost is so high in *BTA*.

As expected, the performance of each approach depends on the domain. ***CBO*** could work well if **S** can open *opportunities* for **P** to improve its plan while **P** is computing its initial plan. In that case, the assistance will be effective if **P** does not replan prematurely (**TEA1**) or if the deviation from **P**'s original plan continues allowing assistance (**CA1**, **CA2** and **VA**). However, it will not be a good choice in those domains where small consecutive cost reductions in **P**'s total cost are necessary for a better assistance (***TA***). This is because *CBO* does not consider that **P** will replan more than once when **S** is computing its plan, it so tends to prioritise higher cost reductions regardless of the duration of the plan and future replannings. **S** does not replan for further cost reductions until it has successfully completed the previous assistance, or in case it has failed. On the other hand, ***BTA*** and ***BTA2*** are good options if **P** acts as if it knows that another agent wants to help it (***CA1*** and specially ***CA2***), i.e. if **P**'s real actions match with the actions of **P** estimated by **S**. Also, they will work well in case **S** is able to achieve some of **P**'s goals more easily than **P** by itself (***TA***). Finally, ***BTA2*** is usually preferable to ***BTA*** except when it is necessary to disable **P** (making it impossible for **P** to compute a plan) to perform a good assistance (***TEA2*** and ***TA***). By limiting the use of *noop*, the actions of **P** in the centralized plans will not be based on waiting for **S** to facilitate its goals, which would be something **P** would not really do.

Related Work

Many previous works require communication between agents in order to assist and avoid risks and conflicts (Geib et al. 2016; Kulkarni, Srivastava, and Kambhampati 2021), or to capture the human attention (Sengupta et al. 2017). Some of them provide proactive assistance in specific domains where communication is necessary such as a com-

puter assistant (Yorke-Smith et al. 2012). In some cases of communication, all agents could coordinate with each other to achieve an optimal assistance. However, in some domains communication may not be possible. For example, when we need to help elderly and the communication is not precise, or when we need to assist busy people, who are performing tasks that require their full attention, without being too intrusive. In our work we pretend to assist **P** by inducing its behaviour and changing the environment accordingly.

Additionally to the compilation of Freedman et al. in (2017), which corresponds to *BTA*, another close work is done by Chakraborti, Tathagata et al. in (2015). They redefine the concept of serendipity as the emergence of a change in the environment that achieves a reduction in the total human cost (**P**). They incorporate a robot (**S**) acting in parallel whose purpose is to modify the environment to achieve such serendipity. Thus, the human deviates from his original plan, reducing his cost. Eventually, **P** resumes the execution of the original plan's actions. To develop the idea, they define some constraints (avoiding concurrence of actions, penalising the time window where serendipity happens, etc.) and apply planning. This implementation covers a small fraction of what the assistance is, as the robot tries to ensure that the human does not deviate from its original plan prematurely, interfering as little as possible and includes communication between agents. In some other works, agents may decide not to act if doing so could lead the assisted agent to a worse situation (Kamar, Gal, and Grosz 2009; Unhelkar and Shah 2016). Other works defined a helpfulness measure to determine how useful would be the assistant agent's actions (Freedman and Zilberstein 2017). In our case, **S** will act even if **P** computes more expensive plans in its next turns, looking for a future assistance in the long term. Taking this risk implies **P** could finally worsen its plan in a planning-execution environment. However, this would not be a problem as **S** could simulate the entire execution before carrying it out, since it knows **P**'s model and how **P** compute their plans.

Since there is no communication, **S** might not know **P**'s goal either. For that reason, a recurrent line of research consists on applying goal recognition (Fern et al. 2007; Freedman and Zilberstein 2017; Molineaux et al. 2018). To facilitate the goal recognition between the agents, they may decide to execute those actions that provide the most information about the goal that they are pursuing (Dragan and Srinivasa 2013; MacNally et al. 2018; Miura and Zilberstein 2020). Others also apply plan and intent recognition (Levine and Williams 2018). Some of those approaches related to plan, intent or activity recognition were unified by Freedman's work in (2019). In some cases, we do not know **P**'s action model either. In these cases, we could apply also model recognition (Zhuo and Kambhampati 2013; Zhuo and Yang 2014). These approaches could be used before using our compilations.

There are also works where several agents cooperate with each other, especially in multi-agent environments (Chakraborti et al. 2015), and even deal with goal, beliefs and actions recognition (Albrecht and Stone 2018). Also, agents could act in adversarial environments (Keren, Gal, and Karpas 2016; Masters and Sardina 2017; Freedman and Zilberstein 2017; Kulkarni, Srivastava, and Kambhampati 2019). Some proposals that solve the problem of assistance can be changed slightly to define agents acting in adversarial environments, despite being opposites, and vice versa.

Conclusions and Future Work

We have proposed a assisting agent (**S**) whose purpose is to proactively assist another agent (**P**) using **P**'s initial plan through a compilation to classical planning in a non-communicating environment. We have defined several domains to test our idea in different situations. Subsequently, we compared our main approach with simple centralised compilations. Finally, we have identified which compilation is most suitable for each of those domains.

In view of the promising results, we want to relax some of our strong assumptions to design solutions based on *CBO* that open up opportunities for **P** to improve its plan and estimating the future actions of **P**. One of these relaxations can consist on replacing **P**'s optimal plans by policies, since they are more reactive and efficient than automated planning. In addition, **S**'s knowledge about the environment could be reduced: assuming that the assistant is not aware of **P**'s algorithm for computing plans, or even that it does not know the **P**'s goals and action model. This would require solutions based on goal and model recognition. On the other hand, it could be interesting to assist without calculating the goal of **P**, since performing goal recognition in grid-based problems (as in CA1, to find out what is the goal cell) could be complex. For example, in CA1 it would be enough to bring the car close to **P**, regardless of where **P** wants to go. Once these problems have been adequately solved, we would like to develop approaches for non-deterministic environments with partial observability, where both agents also act in parallel, and finally in a real environment.

Acknowledgements

This work has been partially funded by FEDER/Ministerio de Ciencia, Innovación y Universidades - Agencia Estatal de Investigación/TIN2017-88476-C2-2-R and by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17), and in the context of the VPRICIT (Regional Programme of Research and Technological Innovation).

References

Albrecht, S. V.; and Stone, P. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258: 66–95.

Chakraborti, T.; Briggs, G.; Talamadupula, K.; Zhang, Y.; Scheutz, M.; Smith, D.; and Kambhampati, S. 2015. Planning for serendipity. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5300–5306. IEEE.

Dragan, A. D.; and Srinivasa, S. S. 2013. Generating legible motion. In *Robotics: Science and Systems*.

Fern, A.; Natarajan, S.; Judah, K.; and Tadepalli, P. 2007. A decision-theoretic model of assistance. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1879–1884.

Freedman, R.; and Zilberstein, S. 2017. Integration of planning with recognition for responsive interaction using classical planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4581–4588.

Freedman, R. G.; and Zilberstein, S. 2019. A unifying perspective of plan, activity, and intent recognition. In *Proceedings of the AAAI Workshops: Plan, Activity, Intent Recognition (Honolulu, Hawaii, USA, 2019)*, 1–8.

Geib, C.; Weerasinghe, J.; Matskevich, S.; Kantharaju, P.; Craenen, B.; and Petrick, R. 2016. Building helpful virtual agents using plan recognition and planning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 162–168.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; Koenig, S.; et al. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, volume 7, 1007–1012.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Kamar, E.; Gal, Y.; and Grosz, B. J. 2009. Incorporating helpful behavior into collaborative planning. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 875–882. Springer Verlag.

Katz, M.; Keyder, E.; Winterer, D.; and Pommerening, F. 2019. Oversubscription planning as classical planning with multiple cost functions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 237–245.

Keren, S.; Gal, A.; and Karpas, E. 2016. Privacy Preserving Plans in Partially Observable Environments. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 3170–3176.

Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2019. A unified framework for planning in adversarial and cooperative environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2479–2487.

Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2021. Planning for proactive assistance in environments with partial observability. In *Workshop on Explainable AI Planning (XAIP). International Conference on Automated Planning and Scheduling (ICAPS).*

Levine, S. J.; and Williams, B. C. 2018. Watching and acting together: concurrent plan recognition and adaptation for human-robot teams. *Journal of Artificial Intelligence Research*, 63: 281–359.

MacNally, A. M.; Lipovetzky, N.; Ramirez, M.; and Pearce, A. R. 2018. Action selection for transparent planning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 1327–1335.

Masters, P.; and Sardina, S. 2017. Deceptive path-planning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, 4368–4375.

Miura, S.; and Zilberstein, S. 2020. Maximizing plan legibility in stochastic environments. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems*, 1931–1933.

Molineaux, M.; Floyd, M. W.; Dannenhauer, D.; and Aha, D. W. 2018. Human-agent teaming as a common problem for goal reasoning. In *Proceedings of AAAI Spring Symposium Series*.

Sengupta, S.; Chakraborti, T.; Sreedharan, S.; Vadlamudi, S. G.; and Kambhampati, S. 2017. RADAR — a proactive decision support system for human-in-the-loop planning. In *Proceedings of AAAI Fall Symposium Series*, 269–276.

Unhelkar, V. V.; and Shah, J. A. 2016. ConTaCT : deciding to communicate during time-critical collaborative tasks in unknown, deterministic domains. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2544–2550.

Yorke-Smith, N.; Saadati, S.; Myers, K. L.; and Morley, D. N. 2012. The design of a proactive personal agent for task management. *International Journal on Artificial Intelligence Tools*, 21(01): 1250004.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proceedings of Twenty-Third International Joint Conference on Artificial Intelligence*, 2444—-2450.

Zhuo, H. H.; and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artificial Intelligence*, 212: 80–103.