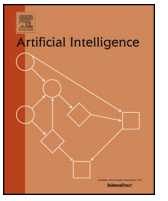
Artificial Intelligence 318 (2023) 103883

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Width-based search for multi agent privacy-preserving planning ☆

Alfonso E. Gerevini [a,*], Nir Lipovetzky [b], Francesco Percassi [c,a], Alessandro Saetti [a,*], Ivan Serina [a]

[a] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
[b] School of Computing and Information Systems, The University of Melbourne, Australia
[c] School of Computing and Engineering, University of Huddersfield, United Kingdom

ARTICLE INFO

Article history:
Received 29 December 2021
Received in revised form 15 January 2023
Accepted 12 February 2023
Available online 21 February 2023

Keywords:
Planning
Multi-agent planning
Privacy-preserving planning
Distributed planning

ABSTRACT

In multi-agent planning, preserving the agents' privacy has become an increasingly popular research topic. For preserving the agents' privacy, agents jointly compute a plan that achieves mutual goals by keeping certain information private to the individual agents. Unfortunately, this can severely restrict the accuracy of the heuristic functions used while searching for solutions. It has been recently shown that, for centralized planning, blind search algorithms such as width-based search can solve instances of many existing domains in low polynomial time when they feature atomic goals. Moreover, the performance of goal-oriented search can be improved by combining it with width-based search. In this paper, we investigate the usage of width-based search in the context of (decentralised) collaborative multi-agent privacy-preserving planning, addressing the challenges related to the agents' privacy and performance. In particular, we show that width-based search is a very effective approach over several benchmark domains, even when the search is driven by heuristics that roughly estimate the distance from goal states, computed without using the private information of other involved agents. Moreover, we show that the use of width-based techniques can significantly reduce the number of messages transmitted among the agents, better preserving their privacy and improving their performance. An experimental study presented in the paper analyses the effectiveness of our techniques, and compares them with the state-of-the-art of collaborative multi-agent planning.

1. Introduction

A number of real-world applications require that different agents collaborate without sharing all the knowledge they have about the task to solve. For instance, in a logistic domain a customer wants to buy goods from a retailer without the courier knowing the content of the packages that he manages. The problem that raises this issue takes the name of collaborative multi-agent privacy preserving planning (CMAPP) [3,9]. Indeed, in CMAPP planning agents may have private knowledge that they do not want to share with others during the planning process and plan execution, and an important issue is related to how the search of a solution handles the agents' privacy. This issue prevents the straightforward usage of

☆ This paper is a revised and extended version of [2,13,14].

* Corresponding authors.
E-mail addresses: alfonso.gerevini@unibs.it (A.E. Gerevini), alessandro.saetti@unibs.it (A. Saetti).

https://doi.org/10.1016/j.artint.2023.103883

most of the modern techniques developed for centralized (classical) planning, which are based on heuristic functions that use the knowledge of all the involved agents.

Recent work in classical planning has shown that width-based search algorithms can solve planning problem instances of many existing domains in low polynomial time when they feature atomic goals. Width-based search relies on the notion of "novelty". The novelty of a state s has been originally defined as the size of the smallest tuple of facts that hold in s for the first time in the search, considering all previously generated states [19]. For instance, a state search with novelty equal to 1 is such that it achieves for the first time in the search some single literal; a state search with novelty equal to 2 is such that it achieves for the first time in the search some pair of literals and all the single literals in the state have been previously achieved during the search. The novelty measure has been used to both prune search states and guide the search, by defining heuristics that combine it with different estimates of the goal distance [20,21]. Search states with smaller novelty correspond to states that have higher priority to be expanded. For instance, the achievement of a new literal is considered more important than the achievement of a new pair of literals formed by two single literals already achieved in the search.

In this paper, we investigate the usage of width-based search algorithms for CMAPP planning. The effectiveness of width-based search for CMAPP planning can be affected by the need of treating the privacy of the involved agents. In order to preserve the privacy, the private knowledge shared among agents can be encrypted: an agent α_i shares with the other agents a description of each reached search state in which all the private facts of α_i that are true are substituted with a single identifier. This encryption has an impact on the measure of novelty, and hence for CMAPP planning it can also affect the effectiveness of the width-based search algorithms.

The first algorithms that we consider for CMAPP planning are Iterative Width search (IW) and Serialized Iterative Width search (SIW) [19]. Essentially, IW consists of a sequence of breadth-first searches with an increasing integer value k for which all states with novelty greater than k are pruned. The success of this simple pruning technique derives from the fact that usually actions in optimal plans achieve tuples of facts of size smaller than or equal to k for the first time on the solution path [19]. For planning problems featuring multiple goal propositions, IW is less effective, but there are some variants of this search algorithm that perform well for a number of planning benchmarks [19]. One of these variants is SIW, which restarts the Iterative Width search from each reached state that achieves at least one more goal since the last restart. The usage of IW and SIW for CMAPP-planning problems is not straightforward, as it raises some synchronization issues to overcome. For instance, a state generated by an agent after the restart of the search may be received by another agent before that agent restarts its own SIW search.

Algorithms IW and SIW are pure exploration methods that are not goal oriented. With the aim of computing plans that are not necessarily optimal, the performance of goal oriented search can be improved by combining it with width-based search. The combination yields a search algorithm, called best-first width search (BFWS), that for classical planning domains outperforms the state-of-the-art planners even when the estimate of the distance to the problem goals is inaccurate [21]. For this reason, the third width-based search algorithm that we consider for CMAPP planning is BFWS. While in classical planning the heuristic used to guide BFWS uses *all* the knowledge of the problem specification, in the setting of CMAPP planning, computing search heuristics using the knowledge of all the involved agents should be avoided because it might compromise the agents' privacy. In order to preserve the privacy of the involved agents, in each agent's search the distance to the problem goals is estimated using the knowledge of a *single* agent. On the other hand, this estimate is much more inaccurate than the estimation obtainable using the knowledge of all the agents. Since for classical planning best-first width search performs very well even when the estimate of the goal distance is inaccurate, such an algorithm is a good candidate to effectively solve CMAPP-planning problems without compromising the agents' privacy.

This paper has the following contributions. First, we propose a search algorithm, CMAPP-SIW, which is a variant of SIW for CMAPP planning, and a new definition of novelty that overcomes the synchronization issue of IW and SIW. Then, we propose a new search algorithm CMAPP-BFWS, which uses width-based exploration in the form of novelty-based preferences to provide a complement to goal-directed heuristic search for CMAPP planning. We adapt the definition of classical width [19] to CMAPP planning, and propose a definition of state novelty for which CMAPP-BFWS can be complete when states are pruned only if their novelty is higher than the width of the problem. Then, we define a number of heuristics for which the preferred states in the open list are the ones with the smallest novelty and, among those, the ones with the lowest goal distance. For this purpose, we define the novelty in a different way, taking the heuristics used to estimate the goal distance into account [21]. Finally, we investigate the use of novelty to filter the messages sent by each agent, propose different methods to exploit such filtering within forward search CMAPP planning, and discuss its properties in terms of privacy.

Moreover, we theoretically examine and classify a number of existing CMAPP-planning domains according to the agents' capability of distinguishing among instances of a planning problem which differ only for the private parts of an agent in the problem. Finally, we experimentally evaluate the effectiveness of the proposed search algorithms and heuristics, and compare the proposed techniques with state-of-the-art planners. Such planners exchange more and different information with respect to our approaches. On one hand, this may imply that these planners preserve the agents' privacy in a form weaker than ours, but on the other hand, it may also make their search heuristics and techniques more effective. However, the results of our experimental study show that best-first width search is competitive with the state-of-the-art also for CMAPP planning, even if less information is exchanged. Moreover, we evaluate the robustness of our approach, considering

different delays in the transmission of messages as they would occur in overloaded networks, due for example to massive attacks or critical situations.

The remainder of the paper is structured as follows. Section 2 gives the necessary background on the MA-STRIPS planning problem, on the notion of privacy in MA-STRIPS planning, and on the width-based search algorithms IW, SIW, and BFWS (originally proposed for classical planning). Section 3 explores the usage of width-based search for CMAPP planning and, specifically, it formalizes new definitions of novelty for CMAPP planning, and investigates the usage of these definitions for pruning the search, guiding the search, and filtering the messages exchanged among the agents. Section 4 gives a taxonomy of a number of exiting CMAPP-planning domains which is based on the admitted privacy. Section 5 reports on a set of experiments that demonstrate the effectiveness of width-based search for CMAPP planning in practice. Finally, in Sections 6 and 7, we discuss related work and draw our conclusions.

2. Background

In this section first, we present the MA-STRIPS planning problem, then we discuss a number of issues concerning the agents' privacy, and finally we describe some prominent width-based search algorithms developed for classical planning.

2.1. MA-STRIPS planning

Our work relies on MA-STRIPS, a "minimalistic" extension of the STRIPS language for multi-agent (MA) planning [10], that is the basis of the most popular definition of CMAPP-planning problem, see, e.g., [9,24,27,29,31].

Definition 1. A MA-STRIPS planning problem Π for a set of agents $\Sigma = \{\alpha_i\}_{i=1}^{m}$ is a 4-tuple $\langle \{A_i\}_{i=1}^{m}, P, I, G \rangle$ where:

- A_i is the set of actions that agent α_i can execute, and such that for every pair of agents α_i and α_j $A_i \cap A_j = \emptyset$;
- P is a finite set of propositions;
- $I \subseteq P$ is the initial state;
- $G \subseteq P$ is the set of goals.

Each action a consists of a name, a set of preconditions, $Prec(a)$, representing facts required to be true for the execution of the action, a set of additive effects, $Add(a)$, representing facts that the action makes true, a set of deleting effects, $Del(a)$, representing facts that the action makes false, and a real number, $Cost(a)$, representing the cost of the action. A fact is *private* for an agent if other agents can neither achieve, destroy nor require it [10]; otherwise the fact is *public*. An action is *private* if all its preconditions and effects are private; otherwise the action is *public*. A state obtained by executing a public action is said to be public; the initial state is also said to be public; any other state is said to be private. We use $Public(s)$ to denote the public part of a state s. Moreover, in the remainder of the paper the planning problems encoded by the MA-STRIPS language are called CMAPP planning problems.

We assume that the set of goals is public, so that any agent can distinguish goal states from other states. A solution plan is a sequence of actions whose application over the initial state I leads to a state that satisfies the problem goals G. Each action in the plan is labeled with the step at which it is executed. A *solution single-agent plan* is the part of the solution plan including the actions (with their steps) of a given agent. The cost of a plan is the sum of all the costs of the actions in the plan. A solution plan is said to be optimal if there exists no other solution with lower cost.

A popular algorithm for solving CMAPP planning problem is MAFS [31], the distributed variant of forward best-first search. Essentially, in MAFS each agent considers a separate search space, and maintains its own open list of states that are candidates for expansion as well as its own closed list of already visited states. Each agent expands the state among those in its open list which is estimated to be most promising for reaching the problem goals. When an agent expands a state, it uses only its own actions. If the action used for the expansion is public, the agent sends a message containing the expanded public state to other agents. When an agent receives a state via a message, it checks whether this state appears in its open or closed lists. If it is not contained in these lists, the agent inserts the state into its open list.

2.2. Agents' privacy

Privacy in CMAPP planning is concerned with guaranteeing that the private information of an agent α_i remains known only by agent α_i. Such private information consists of the agent's private propositions, as well as the existence, structure and cost of its private actions. To maintain the agents' privacy, the private information shared among agents can be encrypted. An agent can share with the other agents a description of a search state in which each private fact that is true in the state is substituted with a (different) identifier [7,8]. While this encryption does not reveal the names of the private facts of each agent α_i to other agents, an agent can realize the existence of a private fact of agent α_i and monitor its truth value during search. This in turn allows the other agents to infer the existence of private actions of α_i, as well as to infer their causal effects. Another way of sharing a state containing private information during the search is to substitute, for each agent α_i, all the private facts of α_i that are true in the state with a single identifier [31]. Such an identifier denotes a new dummy private fact of α_i, which is treated by other agents as a regular fact. The work presented in this paper uses this latter

method for the state encryption. With this method, the other agents can only infer the existence of a group of private facts of α_i, since the dummy identifier contained in the state exchanged by α_i substitutes an arbitrary number of private facts of α_i.

Brafman (2015) [9] defines an algorithm as *weakly privacy preserving* if no private propositions is shared with other agents, and the information shared among agents consists of *public projections* of public actions, i.e., private preconditions and effects are dropped from public actions. The existing algorithms for CMAPP planning typically achieve weak privacy by encrypting the private propositions of a state together before sending it. Weakly privacy-preserving algorithms allow agents to track changes in states sent by one agent and infer the existence of a group of private propositions. In contrast, an algorithm is *strongly private* if no agent α_i can deduce the existence of a variable private to another agent α_j, a possible value of a private variable, and the model of a private action of α_j, beyond (1) what the actions A_i of α_i reveal, (2) the public projection of the actions A_j of α_j, and (3) the public projection of the actions in the solution plan.

Brafman (2015) [9] also proposes *secure-MAFS*, a complete and sound forward search algorithm that achieves *strong privacy* when the heuristic used to guide the search is independent from the private part of the problem and all actions have unary cost. The key insight relies on making sure that an agent sends a state s to other agents iff the public projection of state s has never been sent before. This ensures that agents never receive two states with the same public projection from the same source agent, which is sufficient to guarantee that agents cannot distinguish between the executions of *secure-MAFS* with problems $\Pi, \Pi' \in C_{\Pi}^{Pub}$, where C_{Π}^{Pub} is an equivalence class containing all problems that share the same public solution space as the original problem Π being solved [9,50]. Another notion of privacy proposed by Maliah et al. (2016b) [26] is the *cardinality privacy*, which prevents an agent from inferring the number of private objects of the same type managed by other agents.

The amount of private information leaked by agents during the planning process can be measured by appropriate metrics. The leakage metric proposed by van der Krogt (2009) [18] is based on the number of plans of an agent which are compatible with the exchanged information. More recently, Stolba et al. (2019) [34] use a leakage metric based on the difference between the number of transition systems of the agent compatible with the information available from an adversary agent before and after the planning process.

2.3. Width-based search for classical planning

Lipovetzky and Geffner (2012) [19] introduced the notion of *width* for a classical planning problem. Specifically, given a natural number i, they define a set S^i of tuples such that each tuple t' in S^i is formed by at most i atoms and the presence of t' in S^i indicates that either t' is true in the problem initial state or there is another tuple t in S^i such that all the optimal plans π achieving t yield optimal plans achieving t', once a suitable action a is appended to π. They define the notion of width of a formula ϕ as the minimum natural number w such that S^w contains a tuple that implies ϕ. Finally, they define the notion of width of a planning problem as the minimum natural number w such that the conjunction of the problem goals is implied by a tuple in S^w.

Pure width-based search algorithms explore the search space without any guidance to the problem goals, i.e. they are blind search algorithms. The simplest of such algorithm is known as bounded iterative width, IW(1), which is just a breadth-first search extended with the following pruning rule: a generated state s is kept in the queue only if it contains a proposition $p \in s$ that has not been seen so far in any other state in the search, i.e., $p \notin s'$ for any previously generated state s'. Algorithm IW(2) is similar except that a state s is kept in the queue if there is either 1) a single proposition $p \in s$ that has not been seen so far, as in IW(1), or 2) there is a pair of propositions $p, q \in s$ that has not been seen before, i.e., $p, q \notin s'$ for any state s' generated before s. More generally, bounded iterative width, IW(k), is a standard breadth-first search except that each newly generated state s is pruned when its novelty is greater than k. The next definition states the notion of novelty for a state of the search.

Definition 2 *(Novelty).* Given a search procedure, the set T of states generated by the procedure so far, and a new generated state s, the novelty of s, $w(s)$, is the size of the smallest tuple t of propositions in s such that t is achieved for the first time during the search procedure [19], i.e., $w(s) = \min_{t \subseteq s,\, t \not\subseteq s',\, s' \in T} |t|$.

While IW(k) is relatively simple, it has been shown that IW(k) manages to solve arbitrary instances of many of the standard benchmark domains in low polynomial time provided that the goal states of the planning problem are defined with a single proposition, i.e., $|G| = 1$. Such domains can be shown to have a small and bounded width $w = 1$ or $w = 2$ that does not depend on the instance size, which implies that they can be solved (optimally) by running IW(w). Interestingly, IW(k) runs in time and space that are exponential in k and not in the number of problem propositions P [19].

Bounded IW(k) with $k < w$ can terminate without returning a solution as it may prune all the states leading to a solution. Therefore, Lipovetzky and Geffner (2012) [19] proposed a complete Iterative Width (IW) procedure that sequentially calls the bounded IW(k) with increasing $k = 1, \ldots, |P|$ until the problem is solved. This is especially useful when the width w of a problem is not known in advance but it is likely to be low, e.g., $w = 1$ or $w = 2$, as it renders IW a quadratic algorithm.

The width of problems where goal states are defined with a goal formula containing more than a single proposition ($|G| > 1$) tend to be larger than the width of problems with a single atomic goal formula ($|G| = 1$). Serialized Iterative

Width (SIW) was proposed to solve problems with $|G| > 1$ while keeping a low maximum bound k needed in IW(k) [19]. To that end, SIW calls IW to achieve one atomic goal at a time. More precisely, SIW is a sequence of $|G| - 1$ calls to IW, where the j-th call of IW stops when IW generates a state s_j achieving $G_j \subseteq G$ such that $G_{j-1} \subset G_j$, i.e., at least one extra goal proposition is achieved with respect to the goals G_{j-1} achieved in the last call of IW. The new state s_j then becomes the initial state for the next $(j+1)$ IW search. The well-known heuristic h_{max} introduced by Bonet and Geffner (2001) [5] is used to avoid committing to states $G_j \in s$ where G_j cannot lead to states achieving the remainder goals in $G \setminus G_j$. This last condition is checked by testing whether $h_{max}(s_j) = \infty$ is true once the actions that delete propositions from G_j are excluded. While SIW is an incomplete blind search algorithm (if dead-ends exist), it turns out that is performs better than a greedy best-first search guided by standard delete relaxation heuristics [19].

Width-based exploration in the form of novelty-based soft preferences, changing the expansion order of search algorithms instead of pruning generated states, can provide an effective complement to goal-directed heuristic search without losing completeness [21]. Most standard goal-directed heuristic search algorithms are based on a best-first search strategy. Best-first search uses an evaluation function f to rank the nodes in the *open list*, defining which is the node to expand next in the search. A widely used evaluation function is $f = h$, where h can be any suitably defined goal-directed heuristic, giving preference to expand nodes closer to the goal. The combination of width-based search and goal-directed heuristic search is called best-first width search (BFWS). Differently from classical best-first search procedures, BFWS(f) adopts an evaluation function $f = \langle w, h_1, ..., h_n \rangle$ that uses w to rank the nodes in the *open list*, preferring nodes with best novelty (smaller value of w), and breaking ties lexicographically with n goal directed heuristic functions $h_1, \ldots, h_n$. The primary evaluation function w is given by the novelty measure of the node. To integrate novelty with the goal directed heuristics, the definition of novelty used by BFWS is different from that used for the breadth-first search IW.

The next definition of novelty integrates both the structure of the states in terms of their propositions, and the goal directed heuristics used to guide the search [21].

Definition 3 *(Goal Directed Novelty).* Given a search procedure, the set T of the states generated by the procedure so far, a new generated state s, and a series of heuristic functions $H = \langle h_1, \ldots, h_n \rangle$, let $C(s, T, H) = \{\ s' \mid \forall h_i \in H,\ h_i(s') = h_i(s),\ \ s' \in T\ \}$ be the set of states $s' \in T$ with the same heuristic value of state s. The novelty $w(s)$ of a state s is the size of the smallest tuple t of propositions in s such that t is achieved for the first time with respect to the partition of states $C(s, T, H)$ induced by the heuristic functions $h_i \in H$, i.e., $w(s) = \min_{t \subseteq s,\, t \not\subseteq s',\, s' \in C(s,T,H)} |t|$.

In the remainer of the paper, novelty measure w is sometimes denoted as $w_{(h_1,...,h_n)}$ in order to make the functions $h_1, \ldots, h_n$ used in the definition and computation of w explicit.

3. Width-based search for CMAPP planning

In CMAPP planning, the private information of an agent remains known only to the agent. Most of the search algorithms for CMAPP planning preserve the privacy of the involved agents by encrypting the private information shared among agents. Indeed, an agent α_i shares with the other agents a description of every reached search state in which all the private facts of α_i that are true in a state are substituted with a single identifier.

The encryption of the private knowledge has an impact on the novelty measure, and hence it can also affect the effectiveness of the width-based search algorithms developed for classical planning. E.g., consider states $s_1 = \{p\}$, $s_2 = \{q\}$, $s_3 = \{p, q\}$, where p and q are private facts of an agent different from α_i. Let $[x]$ denote the dummy identifier representing one or more private facts x of an agent different from α_i. The descriptions of s_1, s_2, and s_3 received by α_i are $\{[p]\}$, $\{[q]\}$, and $\{[p, q]\}$, respectively. Assume that the order with which these states are processed by α_i is s_1, s_2, s_3. For the determination of the novelty of a state, we consider a dummy identifier substituting one or more private facts as a regular proposition. Then, without the encryption, for α_i the novelty of s_1 and s_2 is 1, and the novelty of s_3 is 2, while with the encryption the novelty of each of these states is 1, because in s_3 the dummy identifier representing encrypted facts $[p, q]$ is true for the first time in the search. Assume that, as for classical planning, states with novelty greater than a certain threshold k are pruned from the search tree, and let k be equal to 1. Then, the search without the encryption prunes s_3 from the search, while the search encrypting private facts does not.

In this section we present a collection of width-based search techniques for CMAPP planning. First, we adapt the definition of classical width [19] to CMAPP planning, we study the usage of iterative width search for CMAPP planning, and we point out some synchronization issues to address for CMAPP planning. Then, we propose a new search algorithm CMAPP-BFWS, which uses width-based exploration in the form of novelty-based preferences to provide a complement to goal-directed heuristic search, and we give a definition of state novelty for which CMAPP-BFWS can be complete when states are pruned only if their novelty is greater than the width of the problem. Then, we define several heuristics for which the preferred states in the open list are the ones with the smallest novelty and, among those, the ones with the lowest goal distance. For this purpose, we define the novelty in a different way, taking the heuristics used to estimate the goal distance into account [21]. Finally, we investigate the use of novelty to filter the messages sent by each agent, we propose different methods to exploit such filtering within forward search CMAPP planning, and we discuss its properties in terms of privacy.

3.1. Serialized IW for CMAPP planning

The search procedure of SIW for solving a CMAPP problem, as well as for classical planning problems, can be understood as casting the problem into a sequence of planning problems, each of which requires reaching one more goal with respect to the previous one. Each of these sub-problems is solved by running an IW search. Each run is called an *episode* of the search.

Algorithm 1 shows the multi-agent version of SIW search for an agent α_i of the CMAPP-planning problem. Each agent considers a separate search space, since each agent maintains its own list of open states, *open*, and, when an agent expands an open state, it generates a set of states using only its own actions.

During the search, each agent α_i sends two types of messages to other agents, *restart messages* and *state messages*. A restart message contains the encrypted description of a state s to start a new episode, an identifier of the episode, and the number of episodes required to achieve s from the problem initial state. Similarly, a state message contains the encrypted description of a search state s for the current episode, the depth of s in the episode search tree of α_i, and the identifier of the current episode.

Each agent α_i maintains its own list of received messages to process, *open_msgs*. Given a message m, $State(m)$, $Episode(m)$, and $Depth(m)$ respectively denote the search state in m, the episode identifier in m, and the depth of $State(m)$ in the episode search tree of α_i. Algorithm 1 assumes the presence of a separated thread listening for incoming messages sent from other agents. Each time a message is received, it is added to the *open_msgs* list. More precisely, the restart messages are added at the beginning of *received_msgs*, while the state messages are added at its end. In Algorithm 1, both *open* and *open_msgs* are global variables. An agent iteratively processes the messages received in the *open_msgs* list (steps 4–20) and expands the states in the *open* list (step 21–40). Loop 4–40 is repeated until the lists *open* and *open_msgs* are not empty.[1]

Each time a state s is extracted from *open*, first agent α_i checks if the state satisfies the goal of the planning problem. If it does, α_i, together with the other agents, reconstructs the plan achieving s and the solution plan is returned (steps 22–24). Once an agent expands a goal state s, algorithm ReconstructPlan(s) performs the trace-back of the solution plan. Agent α_i begins the trace-back, and when it reaches a state received via a message m, it sends a trace-back message to the agent that sent m. This continues until the initial state is reached. The MA-plan derived from the trace-back is a solution of the CMAPP planning problem. Finally, at step 23 Algorithm 1 returns the part of the solution plan regarding a (single) agent computed by ReconstructPlan(s).

If s is not a goal state, function $IsStateBetter(s, s_I)$ evaluates whether s is a final state of the current episode (steps 25–29). This is the case when s contains more goals than the initial state s_I of the current episode, and the number of goals reachable from s by α_i using actions A_i is the same as those reachable from s using set A_i minus the set of actions deleting the goals in s that are not in s_I. The set of goals reachable from a given state is estimated by constructing a relaxed planning graph from the state using the set A_i of actions. A relaxed planning graph is a planning graph [4] constructed by ignoring deleting effects of actions. Intuitively, the second part of the condition checked by *IsStateBetter* estimates if the remaining unachieved goals are reachable without destroying the new goals achieved by s. The condition used in the version of SIW for classical planning considers whether any goal among the remaining ones becomes unreachable, not only the goals in s that are not in s_I. The reason why our restart condition is different is that in CMAPP planning the h_{max} value that an agent computes for a state s using its own actions gives no valuable information about the solvability of the planning problem from s, since in CMAPP planning each agent is capable of executing only a subset of the problem actions. Essentially, $h_{max}(s)$ can be infinite each time reaching the problem goals from s requires the (joint) work of more than one agent.

Then, agent α_i checks if state s is the result of the application of a public action, and in this case it sends a state message containing state s, its depth, and the identifier of the current episode e to other agents (steps 30–32). Finally, α_i expands state s by applying the actions executable in s and, for each successor state s' of s, α_i decides whether to include s' in its IW search according to the novelty of the state. CMAPP-SIW uses the following definition of novelty.

Definition 4 *(Episode Cost Novelty).* The novelty $w_{(g,e)}(s)$ of a state s is the size of the smallest tuple t of propositions in s such that: (1) t is achieved for the first time during the search of episode e, or (2) for every other state s' containing t, s' was previously generated in the search tree of episode e through paths with greater accumulated cost, i.e., $g(s') > g(s)$.

If $w_{(g,e)}(s')$ is lower than the novelty bound k, α_i adds s' to its *open* list (steps 33–39).

In classical planning, a state s produces a new tuple of propositions t iff s is the first state generated in the episode search that makes t true, i.e., only condition (1) is used for the definition of the novelty of a state. In the context of CMAPP planning, the computation of the novelty is more difficult because, at a given time, the depth in episode search trees constructed by each agent can be quite different. This happens because in CMAPP planning each agent is capable of executing a different set of actions, and hence the searches conducted by two different agents can have a different branching

[1] More precisely, when the *open* and *open_msg* lists of an agent become empty, the agent sends a special message to the other agents representing the fact that its own lists are empty. Similarly, the agent sends another special message to the others when its own open list is not empty anymore. The algorithm terminates when the lists of all the agents are empty.

Algorithm 1: SIW search run by agent α_i from an initial episode state s_I to achieve goals G using only set of actions A_i of α_i. The output is a single-agent solution plan π_i for α_i, or failure.

```
1  Algorithm CMAPP-SIW(s_I, G, A_i, k, E, e)
       Input: An initial episode search state s_I, the set G of goals, the set A_i of actions that agent α_i can execute, a novelty bound k, a (initially
              empty) set E of episode identifiers, an (initially null) episode identifier e;
       Output: A single-agent plan π_i for agent α_i, or failure.
2      open ← s_I
3      while open is not empty or open_msg is not empty do
4          received_msgs ← Dequeue(open_msgs)                                   /* Receive messages */
5          foreach m ∈ received_msgs do
6              if m is a restart message then                                   /* Process restart msg */
7                  E ← E ∪ Episode(m)
8                  if IsEpisodeBetter(State(m), s_I) then
9                      return CMAPP-SIW(State(m), G, A_i, k, E, Episode(m))
10                 end
11             else                                                             /* Process state msg */
12                 if Episode(m) = e then
13                     Enqueue(State(m), open)
14                 else
15                     if Episode(m) ∉ E then
16                         Enqueue(m, open_msgs)                                /* Process later */
17                     end
18                 end
19             end
20         end
21         s ← Dequeue(open)
22         if G ∈ s then                                                        /* Plan found */
23             return ReconstructPlan(s)
24         end
25         if IsStateBetter(s, s_I) then                                        /* Restart */
26             e ← GenerateNewEpisodeIdentifier()
27             SendRestartMessage(⟨s, e⟩)
28             return CMAPP-SIW(s, G, A_i, k, E ∪ {e}, e)
29         end
30         if s is public then                                                  /* Send state */
31             SendStateMessage(⟨s, Depth(s), e⟩)
32         end
33         foreach a ∈ A_i s.t. Prec(a) ⊆ s do                                  /* Expand */
34             s' ← s \ Del(a) ∪ Add(a)
35             g ← Depth(s')
36             if w_(g,e)(s') ≤ k then
37                 Enqueue(s', open)
38             end
39         end
40     end
41     return failure
```

factor. Therefore, in CMAPP planning state s' with search depth d produces a new tuple of propositions t for agent α_i iff s' is the first state with depth lower than or equal to d in the episode search tree of α_i that makes t true. A similar novelty definition was exploited recently in the context of online planning with rollout-IW [1], as well as in alternative formulations of the novelty for classical planning [16].

In Algorithm 1, the episode of each agent α_i terminates if it achieves a state with a new goal, as well as if it receives a restart message from another agent. Consider now this latter case. When a restart message is received, α_i has to assess whether the new restart state is better than the starting state of the current episode. Therefore, each time a restart message m is received, function $isEpisodeBetter(State(m), s_I)$ evaluates whether a new episode starts using $State(m)$ as the initial state (steps 6–11). This happens if the restart state $State(m)$ contains more goals than the initial state s_I of the current episode. If both achieve the same number of goals, we break ties choosing $State(m)$ if it has been achieved with less episodes than s_I, and finally we break ties if needed by choosing the state generated by the process running on a network node whose IP is alphanumerically smaller.

The usage of SIW for CMAPP planning raises some issues about the synchronization of the IW search: when an agent α_i reaches a state achieving a new problem goal, all the agents should restart a new IW search from that state, but accidentally a state s generated by α_j after the restart may arrive to another agent α_i before that agent restarts its own IW search. In

Algorithm 2: k-CMAPP-BFWS run by agent α_i from the initial state I to achieve goals G using only the action set A_i of α_i. The output is a single-agent solution plan π_i for α_i, or failure. Parameter $k \in \mathbb{N}$ is an upper bound for the novelty of expanded states. Function $g(s)$ is the accumulated cost from I to s, and function f is the heuristic function used to sort the *open* list.

```
1  Algorithm k-CMAPP-BFWS(I, G, A_i, f)
       Input: The initial state I, the set G of goals, the set A_i of actions that agent α_i can execute, a heuristic function f;
       Output: A single-agent plan π_i for agent α_i, or failure.
2      open ← I
3      g_I ← 0
4      while open is not empty or open_msg is not empty do
5          foreach s ∈ open_msg do
6              open ← open ∪ s
7              open_msg ← open_msg \ s
8          end
9          s ← SelectBest(f, open)
10         open ← open \ s
11         if G ∈ s then                                              /* Plan found */
12             return ReconstructPlan(s)
13         end
14         if s was generated by agent α_i and s is public then
15             SendStateMessage(⟨s, g(s)⟩)
16         end
17         foreach a ∈ A_i s.t. Prec(a) ⊆ s do                        /* Expand */
18             s' ← s \ Del(a) ∪ Add(a)
19             g(s') ← g(s) + Cost(a)
20             if w_(g)(s') ≤ k then
21                 open ← open ∪ s'
22             end
23         end
24     end
25     return failure
```

such a case, α_i needs to postpone the expansion of s until it restarts its own search from the same episode during which s was expanded. This kind of synchronization issues happens when messages are received in a different order than they were transmitted, and it can often occur if the agents reside on interconnected machines through a networked infrastructure.

We treat this synchronization issue as follows. When a state message m is received, agent α_i checks if $State(m)$ belongs to the current episode (steps 12–14). If it does, α_i includes $State(m)$ in its IW search by adding the state to its *open* list. Otherwise, if $State(m)$ belongs to another episode, α_i checks if the episode it belongs to has already been marked as worst, and simply ignores the message. If the episode identifier associated to m has not been evaluated yet, i.e., $Episode(m) \notin E$, then α_i re-inserts the new state into the *open_msgs* list in order to process it later, once the restart message with the same episode identifier arrives (steps 15–17).

The other synchronization issue arising in CMAPP planning is that at a given time the depth reached by agent α_i in its own episode search tree can be different from the depth reached by other agents because, as mentioned before, the searches conducted by two different agents can have a different branching factor (each agent has its own set of actions). If α_i extracted the first state in the *open* list, it could happen that a state is expanded before another state achieved with fewer actions. To overcome this issue, given an incoming message m, Algorithm 1 adds $State(m)$ in its own *open* list using $Depth(m)$. I.e., it manages the *open* list as a priority list in which the priority of a state is the depth of the state in the episode search tree of the agent that expanded the state; if the state is expanded by an agent different from α_i, the depth of the state comes from the same message with which the state is communicated to α_i. Thereby, agent α_i iteratively extracts a state s from *open* selecting the first state in it that has the lowest depth.

3.2. Best-first width-based search for CMAPP planning

A problem of algorithms SIW and CMAPP-SIW is that they are incomplete, i.e., they do not guarantee to find a solution even if the problem is solvable. This is despite the fact that the search procedure of SIW and CMAPP-SIW consist of a succession of breadth-first state expansions and breadth-first search is a complete algorithm. Indeed, the pruning of SIW and CMAPP-SIW may compromise the completeness. For instance, assume that all the paths from the initial state to any goal state include the sequence of states $\langle s_1, s_2, s_3 \rangle$, where $s_1 = \{p\}$, $s_2 = \{q\}$, and $s_3 = \{p, q\}$. Then the novelty of s_1 and s_2 is (at least) 1, while the novelty of s_3 is (at least) 2. Therefore, if the novelty bound used by SIW and CMAPP-SIW is equal to 1, s_3 is pruned, and no solution can be found. In this section, we study another approach to using width-based search for CMAPP planning that is complete.

Algorithm 2 shows a search algorithm for an agent of the CMAPP-planning problem combining width-based search and goal-directed search, that we call k-CMAPP-BFWS. Parameter $k \in \mathbb{N}$ is an upper bound for the novelty of states that can be expanded, i.e., states with novelty greater than k are pruned from the search space.

Like in CMAPP-SIW, in k-CMAPP-BFWS each agent α_i maintains its own list of open states (*open*) and its own list of received messages to process (*open_msgs*). Agent α_i iteratively expands the states in the open list and those contained in the received messages. Loop 4–23 of k-CMAPP-BFWS(I, G, A_i, f) is repeated until *open* and *open_msg* are empty. Agent α_i extracts all the states in *open_msg*, computes the novelty according to the states generated or received by α_i, computes the given heuristic function f, and adds the states to the open list. Then, α_i extracts the best state s from *open* according to f (steps 5–9). Function f is defined as a sequence of n arbitrary heuristics $\langle h_1, \ldots, h_n \rangle$ that are applied consecutively to break ties. Each time a state s is extracted from *open*, first α_i checks if the state satisfies the goals of the planning problem. If it does, agent α_i, together with the other agents, reconstructs the plan achieving s, and returns its solution single-agent plan (steps 11–13). Otherwise, agent α_i checks if state s is the result of its own public action, and in this case it sends a message to all other agents containing state s together with its accumulated cost $g(s)$ from the initial state up to s (steps 14–16). Finally, α_i expands state s by applying the executable actions and, for each successor state s' of s, α_i evaluates the novelty and function f, and decides whether to add s' in its *open* list according to the novelty of state s' (steps 17–23).

k-CMAPP-BFWS prunes a state s according to a novelty measure akin to the novelty heuristics introduced by Katz et al. (2017) [16], but defined instead on the basis of the cost $g(s)$ accumulated through the trajectory from the problem initial state to s (steps 20-21).

Definition 5 *(Accumulated Cost Novelty).* The novelty $w_{(g)}(s)$ of a state s is the size of the smallest tuple of propositions t in s such that: (1) t is achieved for the first time during search, or (2) every other previously generated state s' where t holds has higher accumulated cost, i.e., $g(s') > g(s)$.

The accumulated cost g of the states that are at the same time in the *open* list of agent α_i can be very different, because k-CMAPP-BFWS does not necessarily extract the state from *open* with the lowest accumulated cost, and *open* may contain states incoming from other agents, which search in different search spaces and may visit states with much greater g-values.

To guarantee the agents' privacy, the private information contained in the visited states is encrypted. As stated before, the encryption affects the measure of novelty, and this consequently also affects the pruning of the search space.

Lemma 1. *The novelty $w_{(g)}(s)$ of a state s computed over states with encrypted information is lower than or equal to the novelty of s computed over states without encrypted information.*

Proof. For simplicity, we consider a CMAPP-planning problem with only two agents α_i and α_j, and we focus on the computation of the novelty for α_i. The argumentation for agent α_j and for problems with more than two agents is similar. Consider a state s and a tuple $t \subseteq s$ such that, without the encryption, $w_{(g)}(s) = |t|$. With the encryption, we distinguish three cases.

(1) Tuple t is formed by public or private facts of α_i (no other fact of other agents). In this case, since only the private facts of α_j are encrypted for α_i, the facts forming tuple t are the same as without the encryption, and hence even with the encryption $w_{(g)}(s) = |t|$.
(2) Tuple t includes at least $n \geq 1$ private facts of agent α_j, and the tuple t' of private facts of α_j that are true in s is different from those of previously generated states such that their accumulated cost is lower than or equal to $g(s)$. With the encryption, the tuple t' is substituted by a new dummy identifier. Such an identifier denotes a dummy fact that is false in all the previously generated states. Hence, by definition, with the encryption we have $w_{(g)}(s) = 1$, which is lower than or equal to $|t|$.
(3) Tuple t includes at least $n \geq 1$ private facts of agent α_j, the tuple t' of private facts of α_j that are true in s is the same as in a state s', s' has been previously expanded, and $g(s') \leq g(s)$. With the encryption, the tuple t' is substituted by an identifier u, which denotes a dummy fact that, in this case, is true in both s and s'. Therefore, the smallest tuple in s that is true for the first time in the search is formed by public or private facts of α_i in t plus u. By definition, with the encryption $w_{(g)}(s) = |t| - n + 1$ and, since $n \geq 1$, the value of $w_{(g)}(s)$ is lower than or equal to $|t|$. □

The definition of width given by Lipovetzky and Geffner (2012) [19] for the state model induced by STRIPS applies directly to the state model induced by MA-STRIPS.

Definition 6 *(CMAPP Width).* The width $w(\Pi)$ of a CMAPP-planning problem $\Pi = \langle \{A_i\}_{i=1}^{|\Sigma|}, P, I, G \rangle$ is the lowest $w \in \mathbb{N}$ for which there exists a sequence of tuples $\langle t_0, \ldots, t_n \rangle$ such that: (1) $t_i \subseteq P$ and $|t_i| \leq w$ for $i = 0, \ldots, n$, (2) $t_0 \subseteq I$, (3) all optimal plans achieving t_i can become optimal plans achieving t_{i+1} by adding an action $a \in \{A_i\}_{i=1}^{|\Sigma|}$, and (4) all optimal plans achieving t_n are also optimal plans achieving the set of propositions in G.

In the previous definition, some actions that extend optimal plans achieving a tuple t_i into optimal plans achieving tuple t_{i+1} can be private. Note that k-CMAPP-BFWS does not send states generated by private actions to other agents. In the following theorem, the novelty $w(s)$ of a state s is computed with respect to the search space of the agent α that generated s. The visited search space of α includes states that α generates as well as all the states received from other agents.

Theorem 1. *Given $k \in \mathbb{N}$, k-CMAPP-BFWS using $f = \langle w, h_1, \ldots, h_m \rangle$ is complete for every problem Π with width $w(\Pi) \leq k$ when $w = w_{(g)}$ and every action cost is non-negative.*

Proof. By Definition 6, if a CMAPP planning problem Π has width $w(\Pi) = k$, it implies that there is an optimal plan trajectory $\pi^{opt} = \langle s_0, \ldots, s_n \rangle$ where every state s_i along the plan trajectory has novelty $w(s_i) \leq k$, inducing a sequence of tuples $\langle t_0 \subseteq s_0, \ldots, t_n \subseteq s_n \rangle$ that complies with the conditions in Definition 6, i.e., $t_0 \subseteq s_0$, $|t_i| \leq w(\Pi)$, all optimal plans achieving t_i can become optimal plans achieving t_{i+1} by adding a single action, and all optimal plans achieving t_n are also optimal plans achieving G.

By Definition 5, if $w = w_{(g)}$, then it follows that there is at least one tuple $t \in s_i$, where $|t| \leq k$, such that no other state s' can be generated with $t \in s'$ and $g(s') < g(s_i)$. We need to show that k-CMAPP-BFWS is complete, when $k = w(\Pi)$, novelty is computed with $w_{(g)}$, and negative action costs are not allowed. Therefore, we show by induction that k-CMAPP-BFWS with $k = w(\Pi)$, $w = w_{(g)}$ is guaranteed to generate each state s_i in π^{opt}, where all $w(s_i) \leq w(\Pi)$, no matter the order in which states are generated, assuming negative action costs are not allowed.

The base case for $i = 0$ is trivially true, as s_0 is the initial state, and a tuple $t_0 \subseteq s_0$ trivially satisfies $|t_0| \leq k$, and no other state in the search tree generated by k-CMAPP-BFWS can have a cheaper plan than the empty plan, given that $g(s_0) = 0$ and negative cost actions are not allowed. Let us show that for $i = 0, \ldots, n-1$, if it holds true for step i, then this must also be true for step $i+1$. From the inductive hypothesis, it follows that k-CMAPP-BFWS generates a state s_i containing a tuple t_i, and no other state s' exists with $g(s') < g(s_i)$ containing t_i, which means that s_i achieves t_i optimally with cost $g(s_i)$, and must be part of an optimal plan trajectory π^{opt}. It also follows that the optimal plan trajectory $\pi_i = \langle s_0, \ldots, s_i \rangle$ for t_i must be extendable with a single action into an optimal plan trajectory $\pi_{i+1} = \langle s_0, \ldots, s_i, s_{i+1} \rangle$ for t_{i+1}, where $|t_{i+1}| \leq k$, resulting in the state s_{i+1} containing t_{i+1}. Since s_{i+1} is generated through an optimal plan, no other state s' containing t_{i+1} can be generated through a cheaper plan. By Definition 5, k-CMAPP-BFWS does not prune s_{i+1}, as the novelty $w_{(g)}(s_{i+1}) \leq k$. It is possible that k-CMAPP-BFWS generates state s' containing t_{i+1} before s_{i+1} through a sub-optimal plan, but once s_{i+1} is generated, s_{i+1} will be marked as novel and added to the open list because it achieves t_{i+1} optimally through a cheaper plan.

State expansion order is determined breaking ties by a sequence of search heuristics h_j, but the heuristics do not make k-CMAPP-BFWS algorithm prune any state, even if $h_j = \infty$, since h_j cannot be proved to be *safe*, i.e., it may give an infinite value even when the goal is reachable, as h_j does not have access to the private actions of other agents.

When a state in an optimal plan has been generated by another agent and sent to other agents, the private facts are encoded as a new fluent. Given Lemma 1, the novelty of such states is guaranteed to be lower than or equal to k, hence they are not going to be pruned by k-CMAPP-BFWS. From the arguments above, it follows that k-CMAPP-BFWS generates the sequence of states $\langle s_0, \ldots, s_n \rangle$ of an optimal plan trajectory π^{opt}, when the bound is $k = w(\Pi)$, the novelty function is $w = w_{(g)}$, and costs are positive real numbers, as $w_{(g)}(s_i) \leq k$, $i = 0, \ldots, n$. □

The next theorem gives an upper bound of the number of states expanded by k-CMAPP-BFWS. Such an upper bound is quantified by using the combination of a number of facts taken k at a time with repetitions, where k is the given novelty threshold. We consider repetitions because we allow combinations to include a tuple of size smaller than k, when a fact is repeated in the combination. The number of n elements taken at a time with k repetitions, also called k-combinations, is $\binom{n+k-1}{k}$.

Theorem 2. *Let k be the maximum novelty allowed by k-CMAPP-BFWS, P^{pub} be the set of public facts, and P_i^{pr} be the set of private facts of agent $\alpha_i \in \Sigma$ such that the total number of facts P is $P^{pub} \cup_{\alpha_i \in \Sigma} P_i^{pr}$. Let $n_i = \sum_{\alpha_i \in \Sigma, j \neq i} \binom{|P_j^{pr}|+k-1}{k}$ be the number of possible k-combinations with repetition of private facts that result in new identifiers, i.e., dummy fluents, that can be sent to agent α_i. k-CMAPP-BFWS using heuristic f terminates after expanding at most*

1. $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}^2$ *states if all action costs are 1,*
2. $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}$ *states if all action costs are 0,*
3. $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}^2 \times |A|$ *states when no action has negative cost,*

where $|\Sigma|$ is the number of agents, and $|A|$ is the total number of available actions for all agents.

Proof. Let $f_i = |P^{pub}| + |P_i^{pr}| + n_i$ be the number of possible state facts for an agent α_i of the CMAPP planning problem Π. We distinguish three cases.

Table 1
Number of instances, and coverages of 1-CMAPP-BFWS and 2-CMAPP-BFWS guided by $f = \langle w_{(g)} \rangle$ w.r.t. CMAPP-BFWS guided by $f = \langle h_{\mathsf{FF}} \rangle$ computed by each agent using its own actions for problem instances with a single goal.

Domain	#Instances	1-CMAPP-BFWS	2-CMAPP-BFWS	h_{FF}
Blocksworld	214	100.0%	100.0%	100.0%
Depot	155	85.81%	91.61%	100.0%
DriverLog	185	95.68%	100.0%	100.0%
Elevators	255	82.75%	66.27%	99.22%
Logistics	172	0.0%	93.6%	100.0%
Rovers	277	98.92%	100.0%	99.28%
Satellites	488	20.9%	98.36%	100.0%
Sokoban	61	54.1%	93.44%	98.36%
Taxi	95	90.53%	98.95%	98.95%
Wireless	160	51.88%	36.88%	58.75%
Woodworking	1084	98.89%	99.17%	97.05%
Zenotravel	258	99.22%	79.84%	100.0%
Overall	3404	77.59%	91.63%	96.94%

(1) When all action costs are 1, the longest path π an agent α_i can expand has length $\pi^{max} = \binom{f_i+k-1}{k}$. For π to be expanded, every state $s_1, \ldots, s_{|\pi|}$ along the path needs to have novelty $w_{(g)}(s_i) \leq k$. Therefore, each state s_i either makes a tuple t of size k true for the first time, or it achieves a tuple t with a lower $g(s_i) < g(s')$ than other previously generated states s' with $t \in s'$. A path $|\pi| > \pi^{max}$ cannot be expanded as the state $s_{\pi^{max}+1}$ in the path has novelty $w_{(g)}(s_{\pi^{max}+1}) > k$. For a path to reach length π^{max}, each state $s_1, \ldots, s_{\pi^{max}}$ must have added at most *one* new tuple or improved the g-value of at most *one* tuple to pass the novelty pruning criteria $w_{(g)}(s_i) \leq k$. Since g grows monotonically, the g-value of a tuple t cannot be improved more than once along the same path π. Once state $s_{\pi^{max}}$ is expanded in the path, all tuples have been generated with smaller g-values. Given that the longest possible plan has length π^{max}, the g-value of a tuple can be improved at most π^{max} times across different paths, therefore each tuple t can let π^{max} states to be expanded with novelty $w_{(g)}(s_i) \leq k$. As the number of tuples is at most $\binom{f_i+k-1}{k}$, in total we can expand $\binom{f_i+k-1}{k} \times \pi^{max} = \binom{f_i+k-1}{k}^2$ states. In the worst case, each agent $\alpha_i \in \Sigma$ can expand the state space independently, yielding the overall $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}^2$.

(2) In case all action costs are zero, the g-value can never be improved once a tuple has been made true by a state. Therefore each tuple t can let just one state to be expanded with novelty $w_{(g)}(s_i) \leq k$, and the total number of expanded states is at most $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}$.

(3) If the cost function maps actions A to positive real numbers including zero, then each tuple t can be improved π^{max} times with $|A_t|$ actions, the number of actions with different cost that add tuple t, which in the worst case is $|A|$. Therefore the total number of states that can be expanded is $\sum_{\alpha_i \in \Sigma} \binom{|P^{pub}|+|P_i^{pr}|+n_i+k-1}{k}^2 \times |A|$. □

Consider now the version of algorithm k-CMAPP-BFWS without the state pruning (i.e., without line 20 in Algorithm 2). We call such a version of the algorithm CMAPP-BFWS. Basically, in this version the notion of novelty is used only in the definition of the search heuristic f to extract the best state from the *open* list.

Theorem 3. CMAPP-BFWS *is complete.*

Proof. CMAPP-BFWS does not prune the search space according to the novelty of search states. In CMAPP-BFWS, each agent α_i expands all the search states reachable from the problem initial state except the private states of agents different from α_i. This is the same set of search states expanded by MAFS. Since MAFS is a complete search algorithm [31], even CMAPP-BFWS is complete. □

Theorem 4. *Let m be the maximum novelty of a state expanded by* CMAPP-BFWS *when* CMAPP-BFWS *finds a plan and terminates. The number of states expanded by* CMAPP-BFWS *is bounded by the worst-case time complexity of k-*CMAPP-BFWS *with* $k = m$.

Proof. k-CMAPP-BFWS with $k = m$ does not prune every state expanded by CMAPP-BFWS when m is the maximum novelty of a state in the search tree of CMAPP-BFWS. Therefore, the search tree of CMAPP-BFWS has the same size as the search tree of k-CMAPP-BFWS with $k = m$ given by Theorem 2. □

Note that, if a problem is solved by 1-CMAPP-BFWS, this does not imply that the problem has width 1. Indeed, sub-optimal solutions may be discovered with a novelty bound lower than the true width of the problem. Still, it provides an estimate on how hard it is to solve a CMAPP planning problem. As shown in Table 1 also for the CMAPP setting, for atomic goals all domains but Wireless generally have width lower than or equal to 2, since for all these domains the coverage of 2-CMAPP-BFWS is quite high. For this analysis, we considered the domains from the *distributed* track of the first

international competition on distributed and multi-agent planning. For each instance with m goal propositions, we created m instances with a single goal. The total number of instances is 3404. Then, we ran 1-CMAPP-BFWS and 2-CMAPP-BFWS over each generated instance. The search heuristic used for 1-CMAPP-BFWS and 2-CMAPP-BFWS is very simple: the best state in the open list is selected among those with the lowest novelty measure $w_{(g)}$, breaking the ties with the accumulated cost g. For each considered domain, Table 1 shows that the total number of generated instances, and the percentage of instances solved with width equal to 1 and lower than or equal to 2. We considered action costs unitary. Therefore, by Theorem 2, this table shows that 77.59% of the considered instances can be solved with a quadratic time $O(n^2)$ where $n = |P|$ is the number of propositions in the problem. These blind and bounded planners perform well with respect to a baseline goal-directed heuristic search planner, CMAPP-BFWS guided by h_{FF} [15] computed by each agent using only its own actions. However, problems with multiple goals have in general a higher width. In the next section we explore how to deal with multiple goals.

3.2.1. On the use of Novelty for heuristic estimates

In the context of CMAPP planning, the search heuristic computed using only the knowledge of one single agent can be quite inaccurate. In this section we propose some new search heuristics for CMAPP planning that combine the measure of the novelty of search states with the estimated distance to reach the problem goals. The goal distance is estimated using the knowledge of a single agent. Our conjecture is that, in the CMAPP setting, width-based exploration in the form of novelty-based preferences can provide a complement to goal-directed heuristic search, so that the search can be effectively guided towards a goal state even if the goal-directed heuristics are inaccurate.

The computation and memory cost of determining that the novelty w of a state s is k is exponential in k, since all the tuples of size up to k but one may need to be stored and processed. For efficiency, we restrict the computation of novelty $w(s)$ to only 3 levels, i.e., $w(s)$ is limited to be equal to 1, 2, or greater than 2.

For our heuristic functions, we used the measure of novelty introduced by Lipovetzky and Geffner (2017) [21].

Definition 7 *(Heuristic Cost Novelty).* Given an arbitrary sequence of heuristic functions $h_1, \ldots, h_n$, the novelty $w(s')$ of a newly generated state s' is k iff there is a tuple of k propositions and no tuple of smaller size, that is true in s but false in every previously generated state s' such that $h_1(s') = h_1(s), \ldots,$ and $h_n(s') = h_n(s)$.

In the remainder of the paper, $w_{(h_1,\ldots,h_n)}$ denotes the novelty measure w using functions $h_1, \ldots, h_n$.

The first heuristic that we study is $f_1 = \langle w_{(h_{\mathsf{FF}})}, h_{\mathsf{FF}} \rangle$, where h_{FF} denotes the goal-directed heuristic of the well-known planner FF. The goal distance of an agent α_i from a search state s is estimated as the number of actions of α_i in a relaxed plan constructed from s to achieve the problem goals. The plan is relaxed because it is a solution of a relaxed problem in which the negative action effects are removed. Substantially, the best state s in *open* according to f_1 is selected not among those with the lowest estimated goal distance, but among those with the lowest novelty measure $w(s) = w_{(h_{\mathsf{FF}})}$, and heuristic h_{FF} is only used to break the ties. The same heuristic was proposed for classical planning surprisingly obtaining good results [21]. A main difference with respect to classical planning is that in the context of CMAPP planning the distance estimated to reach the problem goals can be much more inaccurate, because for an agent α_i the relaxed plan is extracted using only the actions of α_i. When an agent evaluates the search states using only its own set of actions, it is possible that at least one of the problem goals is evaluated as unreachable. In this case, the extraction of the relaxed plan fails, and the estimated distance is evaluated as infinite. This is due to the agent not being able to solve the problem alone, needing to cooperate with other agents.

We consider other types of information for the definition of the search heuristic, in order to overcome the problem of the inaccuracy of the goal-directed heuristics computed using only the knowledge of a single agent. In the following, given a search state s, $G_\perp$ and G_u denote the number of goals that are false in s and the number of goals that are unreachable from s, respectively. For an agent α_i, the number of goals unreachable from s is estimated by constructing a relaxed planning-graph (RPG) from s using only the actions of α_i; specifically, the goals that are not contained in the last level of this RPG are considered unreachable.

Planner CMAPP-BFWS with heuristic function $f_2 = \langle w_{(G_\perp, h_{\mathsf{FF}})}, G_\perp, h_{\mathsf{FF}} \rangle$, denoted as CMAPP-BFWS($f_2$), selects the next state s to expand among those in *open* with the lowest novelty measure $w(s) = w_{(G_\perp, h_{\mathsf{FF}})}$, breaking ties according to the number of goals that are false in s. Heuristic h_{FF} is then used to break ties when there is more than one state in *open* with the same lowest measure of novelty and the same lowest number of false goals.

Similarly, CMAPP-BFWS with heuristic function $f_3 = \langle w_{(G_u, G_\perp, h_{\mathsf{FF}})}, G_u, G_\perp, h_{\mathsf{FF}} \rangle$ selects the next state s to expand among those in *open* with the lowest novelty measure $w(s) = w_{(G_u, G_\perp, h_{\mathsf{FF}})}$, breaking ties according to the number of goals that are unreachable from s. If there is more than one state in *open* with the same lowest measure of novelty and the same lowest number of unreachable goals, ties are broken according to the number of goals false in s. Finally, heuristic h_{FF} is used only if there are still ties to break.

The drawback of h_{FF} for CMAPP planning is that often the estimated goal distance from a search state s is infinite, even though s is not a dead-end. As stated before, the reason for this is that from s the planning problem cannot be solved by an agent alone. With the next proposed search heuristic, we study a method to overcome this problem, for which an agent α_i extracts a relaxed plan from s to the (sub)set of problem goals that are *reachable* from s. The estimated distance from s to all the problem goals is the number of actions in the relaxed plan plus the number of problem goals unreachable from

s multiplied by a constant. In our experiments, such a constant is equal to the maximum number of levels in the RPGs constructed so far. This variant of h_{FF} is denoted as h^P_{FF}. Essentially, the information about the unreachable goals is used to refine the estimated goal distance. We report experiments with CMAPP-BFWS(f_4), where function f_4 is obtained from f_3 using h^P_{FF} in place of h_{FF} as goal-directed component of the evaluation function.

Components G_u and h^P_{FF} of heuristic f_4 are computationally expensive, since for each expanded state G_u requires the construction of a RPG, and h^P_{FF} additionally requires the extraction of a relaxed plan from the RPG. The last two heuristics that we study consider the tradeoff between the accuracy of the estimated goal-distance and its computational cost. For this, the construction of the RPG and the extraction of the relaxed plan are not performed for each expanded state, but only for the initial state of the planning problem and the search states incoming from other agents. The facts that are preconditions of the actions in the relaxed plan are called *relevant facts*. Let s' be the last incoming state in the way to state s for which the relaxed plan was extracted. For evaluating the goal distance of state s, we consider the number $\#r$ of relevant facts that have *not* been made true in the way from s' to s. This measure is similar to that proposed by Lipovetzky and Geffner (2017) [21] for classical planning. The difference with respect to classical planning is that a relaxed plan is extracted for each incoming state, rather than for the states that decrement the number of achieved problem goals in relation to their parent. This is needed as the relevant fluents are not sent among agents in order to avoid compromising privacy. Planner CMAPP-BFWS(f_5) with $f_5 = \langle w_{(G_\perp, \#r)}, G_\perp, \#r \rangle$ considers counter $\#r$ in place of the more computationally expensive components G_u and h^P_{FF}.

The drawback of heuristic f_5 is that, when the number of exchanged messages is high, it still requires the construction of the RPG many times. The construction of the RPG is computationally much more expensive than extracting the relaxed plan and, when such a construction is performed many times, it can become the bottleneck of the search algorithm. Thereby, we propose another heuristic f_6 which, for each agent α_i, requires the construction of the RPG from only the initial state of the problem. With the aim of maintaining the agents' privacy, the RPG is still constructed using only the actions of a single agent. Nevertheless, when the CMAPP-planning problem cannot be solved by an agent alone, the last level of the RPG does not contain the problem goals. For this, the construction of the RPG from the initial state is special, and it is done in two steps. The first step is the construction of the RPG from I. Then, in the second step, the facts $p \in pre(a)$ of actions $a \in A_i$ of agent α_i that are not additive effects of any action of α_i and are not true in the last level of the RPG are made true in the last level of the RPG. Finally, the construction of the RPG continues from the last level of the RPG constructed so far.

Consider a state s to be expanded. For heuristic f_6, counter $\#r$ is defined as the number of relevant facts in the RPG constructed from the problem initial state I that have *not* been made true in the state trajectory from I to s. In the following, counter $\#r$ for f_6 is denoted by $\#r_I$. The computation of $\#r_I$ for f_6 differs from that of $\#r$ for f_5, because an agent α_i alone can reconstruct only the portion of trajectory from the last incoming state s' to s; it cannot reconstruct the entire trajectory from I to s'. The trajectory from I to s' can contain other actions of agent α_i that should be taken into account in the definition of the set of relevant facts that have *not* been made true by α_i in the way from I to s. For this, the presence of these actions of α_i is estimated by solving a *super relaxed* planning problem, i.e., a planning problem with the same initial state of the planning problem, the set of facts that are true in s' as goals, and a set of actions obtained from the set of actions of α_i by ignoring the action preconditions that are unreachable from I, as well as negative action effects. The algorithm for the extraction of the super-relaxed plan is similar to the one used by FF. The positive effects of the actions in such a super-relaxed plan are facts that we estimate have been made true in the way from I to s'. Therefore, for f_6 we define counter $\#r_I$ as the number of relevant facts that have *not* been made true in the super-relaxed plan from I to s' and in the state trajectory from s' to s.

3.2.2. On the use of Novelty for message transmission

A large amount of search states exchanged among agents may slow down the search for a solution of the CMAPP planning problem, specially when the agents' processes run on different network nodes. The notion of novelty can also be used to reduce the number of search states exchanged among agents during the search phase. Therefore, each agent retains, i.e., does not send to other agents, the search states whose novelty value, called *outgoing novelty*, exceeds a given threshold. The outgoing novelty is computed considering the public part of the search states previously transmitted to the other agents.

Definition 8. The **outgoing novelty** of a state s given m functions $h_1, \ldots, h_m$, denoted as $w^{out}_{(h_1,\ldots,h_m)}(s)$, is k iff there is a tuple (conjunction) of k propositions and no smaller tuple that is true in the *public part* of s and false in the *public part* of all states s' previously transmitted with the same function values, i.e., with $h_i(public(s')) = h_i(public(s))$ for $1 \leq i \leq m$. If no such tuple exists, then $w^{out}_{(h_1,\ldots,h_m)}(s) = \infty$.

Based on the outgoing novelty value, we define the notion of withheld state as follows.

Definition 9. The **withheld states** of an agent are the states that have not been sent to the other agents because their outgoing novelty exceeds a given threshold.

In order to preserve the completeness of the algorithm, withheld states can be transmitted to the other agents under specific conditions. To describe these conditions, it is necessary to introduce the notion of *locally empty search*.

Definition 10. The search of an agent α is **locally empty** when the following conditions hold:

1. the *open list* of α is empty;
2. α has no other entry message to process;
3. α has at least one withheld state.

If condition 1 and 2 hold and condition 3 does not hold, then the search of α is **empty**.

An agent whose search phase is empty or locally empty is in a status called *waiting*. When an agent α is *waiting*, it can send part of, or even all, its withheld states (if any) to the other agents in order to "enlarge" the search process of the other agents, or it can ask the other agents to send their withheld states to it in order to resume its search process. In contrast, α could wait that at least a given number of agents, indicated with *num_waiting*, are in the same condition before sending its withheld states or asking the withheld states of the other agents. With this purpose, the agents communicate if their own search is empty or locally empty to the others; when the number of waiting agents exceeds value *num_waiting*, the agents transmit their withheld states.[2] In particular, we distinguish three situations:

- *num_waiting* = 1, agents transmit/request their withheld states when at least one agent is waiting;
- *num_waiting* = *half*, agents transmit/request their withheld states when at least half the agents in the problem are waiting;
- *num_waiting* = *all*, agents transmit/request their withheld states when all the agents in the problem are waiting.

When a condition for the transmission is met, one or more agents can decide to transmit their withheld states. We considered three configurations, indicated with *who_send*, in order to define which agents send the withheld states:

- *who_send* = *waiting*, the *waiting* agents send their withheld states;
- *who_send* = *not waiting*, the *not waiting* agents send their withheld states;
- *who_send* = *all*, all the agents send their withheld states.

The rationale behind configuration *who_send* = *not waiting* is that the waiting agents are idle and need states from the other agents to resume their search. On the other hand, with configuration *who_send* = *waiting*, the waiting agents can take advantage of the period of inactivity to send their withheld states.

When the previous conditions are verified, the agents can send all their withheld states or only a portion of them. Specifically, we considered four different configurations, indicated with *num_withheld_states*, that specify which states, among the withheld ones, have to be sent when requested:

- *num_withheld_states* = *none*, no state is sent (in this case completeness is not guaranteed);
- *num_withheld_states* = 1, one state at the time is sent (the one with lowest heuristic value);
- *num_withheld_states* = *group*, all withheld states with the lowest heuristic value are sent;
- *num_withheld_states* = *all*, all withheld states are sent.

Finally, we study the theoretical properties of novelty-based message filtering for sound and complete CMAPP forward search planners. Without loss of generality, we focus on CMAPP-BFWS, which is weakly privacy preserving: it does not require sharing the public projection of public actions, and it sends messages containing only descriptions of states with the private facts encrypted.

Theorem 5. CMAPP-BFWS *with novelty messages filtering is sound and complete iff* *num_withheld_states* $\neq$ *none*.

Proof. By Theorem 2 we know that CMAPP-BFWS is complete. Novelty filtering can make CMAPP-BFWS incomplete if upon termination it removed sent messages.
If *num_withheld_states* $\neq$ *none*, then eventually all public states are going to be sent. *num_waiting* and *who_send* only change the order of the exchanged messages, and do not preclude that messages will be sent later in the search. Therefore, CMAPP-BFWS is complete as long as *num_withheld_states* $\neq$ *none*. □

Following previous definitions of strong privacy [9], given a problem Π, C_{Π}^{Pub} is the set of problems that share the same public solution space as Π, and hence it is the set of problems such that their public projection is equivalent, i.e., Π and Π' differ only in their private parts for every $\pi' \in C_{\Pi}^{Pub}$. Tožička et al. (2017b) [50] show that it is not possible for a MAP

[2] The transmission of the message "empty search space" among the agents is necessary in order to allow the agents to terminate their search before the timeout.

planner to be strong privacy preserving, complete and efficient at the same time. The next theorem gives certain conditions under which CMAPP-BFWS is incomplete but strong privacy preserving for the problems in set C_{Π}^{Pub}, which share the same public solution space as Π.

Theorem 6. *Let P^{pub} be the set of public facts of a CMAPP problem Π. When the following conditions hold,* CMAPP-BFWS *is incomplete but strong privacy preserving for the problems in C_{Π}^{Pub}, for any threshold $k \in \{1, .., |P^{pub}|\}$ used to withhold the messages:*

1. *$num\_withheld\_states = none$, namely pruning messages with outgoing novelty greater than threshold k;*
2. *no function h_i is used for the definition of the outgoing novelty;*
3. *the heuristic functions used to guide the search are agnostic of the cost and private propositions of the agents.*

Proof. We assume CMAPP-BFWS encrypts the private part of s and no information is leaked from the communication. By Definition 8, for any value of outgoing novelty, a state s is sent iff no other state s' with the same public projection has been sent before. The number of sent messages depends only on the public part of the problem, as the private part is not taken into account in the computation of outgoing novelty. Since any problem $\Pi' \in C_{\Pi}^{Pub}$ shares the same public part of the problem and the reachable public state space of Π, the number of messages sent is equivalent for Π and Π'. Therefore, the search algorithm is strong privacy preserving. □

The state expansion order of CMAPP-BFWS is independent from the private part of the problem if the search is guided, for example, by $f = \langle w_{(\#g)}, d \rangle$, where $w_{(\#g)}$ is the novelty over the public projection of the states using a goal counting heuristic $\#g$, breaking ties with the depth d of the public actions leading to the current state. CMAPP-BFWS with $num\_withheld\_states = none$ is incomplete if a state whose public projection has been sent before needs to be sent again in order to find a solution. This version of CMAPP-BFWS can be made complete by using a strategy similar to that proposed for secure-MAFS [9]. We call the strong-privacy preserving version of CMAPP-BFWS with $f = \langle w_{(\#g)}, d \rangle$ and $num\_withheld\_states = none$ *secure*-CMAPP-BFWS.

When strong privacy cannot be preserved, reducing the number of exchanged messages may be a good strategy to decrease the possibility that other agents can infer private information. This can be accomplished by filtering messages according to the notion of novelty. Indeed, this filtering makes sure that messages with different state variable values are sent first, before sending states with repeating values that can lead to information leakage.

4. A planning domain taxonomy based on the admitted form of confidentiality

In this section we classify CMAPP planning problems according to agents' capability of distinguishing between different instances of a problem, which differ only for the private parts of an agent, and we summarize the results in the form of a planning domain taxonomy. For our analysis, we assume that all problem goals are public. We consider two different classes of planning problems.

Definition 11. A class C of CMAPP planning problems is problem equivalent w.r.t. problem Π for every agent but α_i iff for each $\Pi' \in C$ the projection of Π' over the public and private facts of any agent other than α_i is the same as for Π, i.e., $\Pi = \Pi'$ or the difference between Π and Π' consists of only private actions, private preconditions/effects, and private initial facts of agent α_i.

In the rest of the section, the largest class of CMAPP planning problems that is problem equivalent w.r.t. problem Π for any agent other than α_i is denoted by C_{Π,α_i}^{Prob}.

For instance, consider the CoDMAP version of the Blocksworld domain. For this domain agents are robot arms. The public information in this domain consists in (a) which blocks are clear, (b) which blocks are on the table, and (c) whether a block is on another block. The private information is whether an arm is empty or is holding a certain block. Consider two problems Π and Π', and assume that, for both these problems, there are two agents α_1 and α_2, two blocks `A` and `B`, agent α_1 controls two arms, say `Arm1` and `Arm2`, and Π differ from Π' in the private initial fact of α_1 that either `Arm1` or `Arm2` is holding a block. Let's say that `Arm1` is holding `A` in Π, while `Arm2` is holding `A` in Π'. Then, problem Π' is problem equivalent w.r.t. Π for agent α_2, because the problem definition of Π' available to α_2 is the same as for Π.

For the next class of problems, we consider the forward search tree associated with an agent α_i of a CMAPP planning problem Π as the tree containing all and only the following states: (a) the initial state, (b) the public states sent by agents other than α_i, (c) the goal and dead-end states as leaf nodes, and (d) for every state s but the goal and dead-end states in the tree, the successor of s that are obtained by applying all actions of α_i applicable to s. Basically, such a tree includes the search states that can be *possibly* expanded by a forward-search state-based CMAPP algorithm. These states can be expanded in any topological order; the usage of a specific resolution algorithm determines such an order.

Definition 12. A class C of CMAPP planning problems is forward-search-tree equivalent w.r.t. problem Π for every agent but α_i iff for each $\Pi' \in C$ the forward search tree of Π' associated with any agent other than α_i is the same as for Π.

Table 2
Guarantee of privacy for different types of algorithms and problems.

Algorithm \ Problem	Limited confidentiality admitted	Partial confidentiality admitted	Complete confidentiality admitted
Weak privacy admitted	Privacy leaked from problems and the algorithm	Privacy leaked from problems and the algorithm	Privacy leaked from the algorithm
Strong privacy admitted	Privacy leaked from problems	Privacy leaked from problems	Privacy is kept

In the rest of the section, the largest class of CMAPP planning problems that is forward-search-tree equivalent w.r.t. problem Π for any agent other than α_i is denoted by C^{Tree}_{Π,α_i}.

Consider the previous example of the CoDMAP Blocksworld domain. The forward search trees of α_1 for the problems Π and Π' defined above are in Fig. 1. The search states sent by agent α_2 to α_1 are the same for both the problems; they are omitted in Fig. 1 for simplicity. All the states in the trees are public and the public part of all these states is the same. Since the initial states of Π and Π' is the same for α_2, and the set of public messages sent by agent α_1 to α_2 is the same for Π and Π', the forward search tree of α_2 is the same for Π and Π'. Thereby, problem Π' is forward-search-tree equivalent w.r.t. Π for α_2. Consider now two other problems Π'' and Π''' which differ from Π in the number of the arms controlled by agent α_1. Let's say that for Π'' and Π''' α_1 controls 1 and 3 arms, respectively. Like for Π', both these problems differ from Π in their private initial facts (about the initial availability of the controlled arms), but in addition they contain a set of private actions different from Π. Π''' is again forward-search-tree equivalent w.r.t. Π for agent α_2. This is not the case for Π'', because the forward search tree of agent α_2 in Π'' does not contain the (public) state for which both blocks `A` and `B` are not on the table (and α_2 is not in possession of any block), while such a state is part of the forward search trees of α_2 for Π.

Theorem 7. $C^{Tree}_{\Pi,\alpha_i} \subseteq C^{Prob}_{\Pi,\alpha_i}$.

Proof. Since the initial state is public and goal states are also public (assuming that goals are public), it holds that the initial state and the goal states are shared among agents; hence they are part of the forward search tree of each agent, and hence $C^{Tree}_{\Pi,\alpha_i} \subseteq C^{Prob}_{\Pi,\alpha_i}$. □

The following definition distinguishes three types of problems on the basis of further relations between C^{Prob}_{Π,α_i} and C^{Tree}_{Π,α_i}.

Definition 13. A CMAPP planning problem Π admits

- **limited confidentiality** if, for each agent α_i, $|C^{Tree}_{\Pi,\alpha_i}| = 1$;
- **partial confidentiality** if, for each agent α_i, $|C^{Tree}_{\Pi,\alpha_i}| > 1$ and $C^{Tree}_{\Pi,\alpha_i} \subset C^{Prob}_{\Pi,\alpha_i}$;
- **complete confidentiality** if, for each agent α_i, $|C^{Tree}_{\Pi,\alpha_i}| > 1$ and $C^{Tree}_{\Pi,\alpha_i} \equiv C^{Prob}_{\Pi,\alpha_i}$.

Given a problem Π that admits limited confidentiality, during the planning phase any agent other than α_i can distinguish between Π and *any other* problem Π' that differs from Π in private parts of agent α_i. Indeed, with $|C^{Tree}_{\Pi,\alpha_i}| = 1$, any change in the private part of agent α_i determines a change in the forward search tree of any agent other than α_i. Thereby, the resolution algorithm and/or the solution itself may explore the parts of the tree that have been changed. In this case, the change in the private part of agent α_i would reveal to other agents information about its own private knowledge. When Π admits partial confidentiality, an agent other than α_i may have the chance to distinguish between Π and *at least another* problem Π' in the set of problems that differ from Π in only private parts of agent α_i, but there is at least another problem Π'' in this set such that any agent other than α_i cannot distinguish between Π and Π''. On the contrary, when Π admits complete confidentiality, any agent other than α_i *cannot* distinguish between Π and any other problem Π' that differs from Π in only private parts of agent α_i.

Brafman proposes to distinguish algorithms for CMAPP problems depending on whether an agent can infer the private knowledge of other agents [9]. As mentioned before, he defines an algorithm as weakly private if no agent communicates a private fact (in the initial state, the goal, or any other intermediate state) to another agent during a run of the algorithm, and if the only description of its own actions that it needs to communicate to another agent is their public projection. An algorithm achieves weak privacy by just encrypting private facts. An algorithm is strongly privacy preserving if during the run it does not reveal any information other than what can be derived from the public part of the input problem or the solution.

It is worth noting that the amount of private information an agent can leak does not depend only on the nature of the resolution algorithm used to solve the CMAPP planning problems, but also on the nature of the instance of the CMAPP planning problem. More precisely, the definition provided by Brafman 2015 concerning strongly private search algorithms is related to the only problems admitting complete confidentiality. Indeed, when the confidentiality admitted by the planning domain is limited or partial, multiple runs of the resolution algorithm for problem instances in the domain that have different private parts of agent α_i might allow agents other than α_i to leak an amount of private knowledge of α_i, even if for different planning domains the resolution algorithm admits strong privacy. On the contrary, let's consider the confidentiality

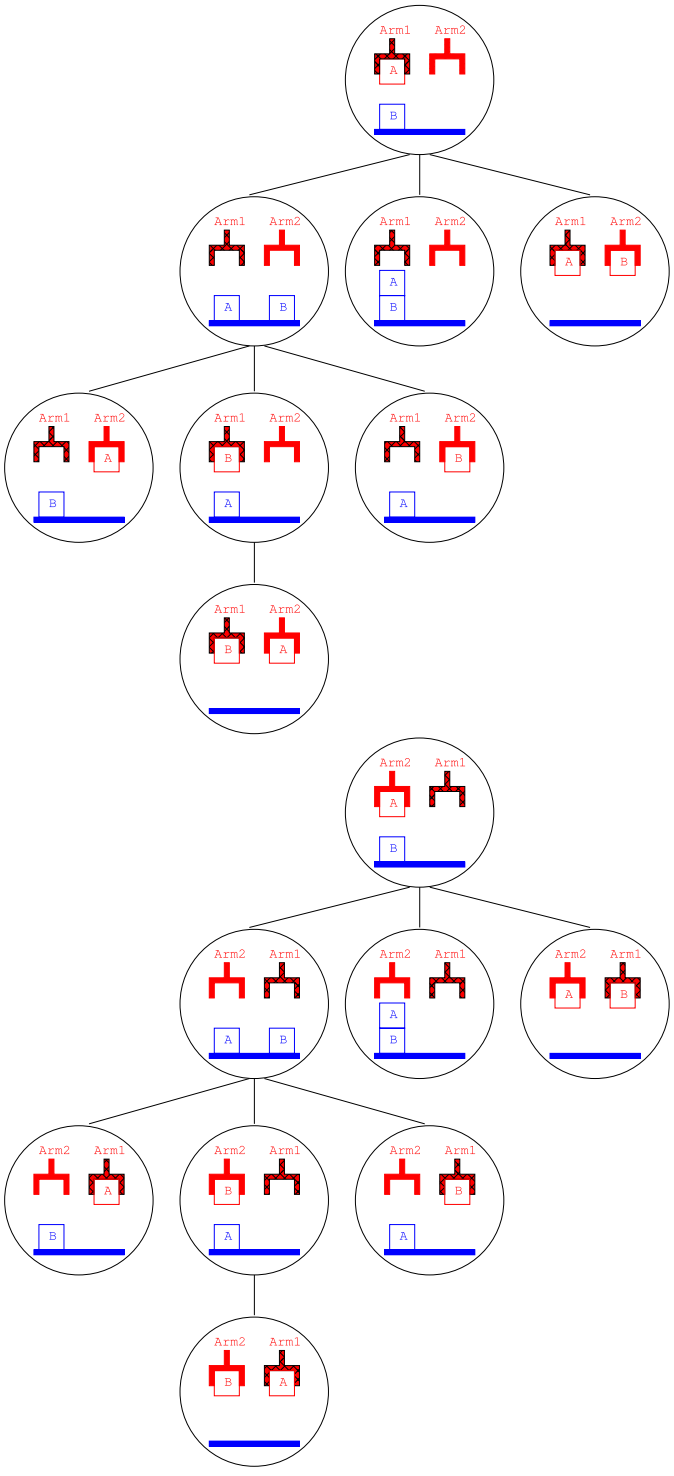


Fig. 1. Search trees of agent α_1 for two instances of Blocksworld with two agents α_1 and α_2 and two blocks `A` and `B`. α_1 controls both `Arm1` and `Arm2`. The search states sent by agent α_2 are omitted for simplicity. The private fact that an arm is either empty or holding a block is depicted in red, while depicted in blue the public part of the search state is. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

admitted by the planning domain as complete. If agent α_i changes its own private knowledge, the forward search tree of agents other than α_i remains the same, and hence the behavior of any resolution algorithm would remain the same for these agents. In this case, no further private knowledge of α_i can be leaked by multiple runs of the resolution algorithm with instances which differ from each other because of the private parts of α_i. Table 2 provides a sketch of the privacy kept/leaked for different problems and algorithms.

For the remainder of this section, we theoretically evaluate the form of confidentiality admitted by the twelve domains proposed by Štolba et al. (2016b) [40] for the distributed track of the first international competition on distributed and multi-agent planning (CoDMAP), and by the four domains MA-Blocksworld, MA-Blocksworld-Large, MA-Logistics, and MA-Logistics-Large proposed by Maliah et al. (2017) [23]. The difference w.r.t. the CoDMAP domains Blocksworld and Logistics is that for the domains of Maliah et al. (2017) [23] many private actions need to be executed between two consecutive public actions, and agents must choose among several paths for achieving goals. These twelve domains are used in our experiments to evaluate the efficacy of the proposed procedures, one of which, *secure*-CMAPP-BFWS, strongly preserves the agents' privacy over the domains in our benchmark that admit complete confidentiality. One of the contribution of this section is to identify such a set of domains.

Theorem 8. *The problems of CoDMAP domain* Blocksworld *admit limited confidentiality, if each agent controls only one (robot) arm. Otherwise, they admit partial confidentiality.*

Proof. Assume that each agent α_i controls exactly one arm. The only information that α_i can keep private is whether the arm controlled by α_i is empty or is holding a certain block. There is no problem Π' with the same initial public facts and private parts of any agent other than α_i, such that Π' differs from Π in whether the arm controlled by α_i is empty or is holding a certain block. It follows that $|C^{Prob}_{\Pi,\alpha_i}| = 1$, and since $C^{Tree}_{\Pi,\alpha_i} \subseteq C^{Prob}_{\Pi,\alpha_i}$ also $|C^{Tree}_{\Pi,\alpha_i}| = 1$ holds and hence these Blocksworld problems only admit limited confidentiality.

Assume that each agent α_i can control more than one arm, and consider two problems Π and Π' with the same set of blocks, the same initial arrangement of blocks, the same set of arms for any agent other than α_i, and such that for Π' α_i controls one arm more than for Π and the additional arm is initially empty. Then, $\Pi' \in C^{Prob}_{\Pi,\alpha_i}$, but $\Pi' \notin C^{Tree}_{\Pi,\alpha_i}$ if the total number of arms is lower than the number of blocks. Indeed, the set of blocks that the arms can hold is larger in Π' than in Π, the set of blocks that at a certain time are holding is the complement of the set of blocks that at that time are either on the table or on another block, and such a set of blocks is part of the public information. This can result in additional states sent to agents other than α_i, which are part of the forward search tree associated with these agents. Therefore, the search trees associated with an agent other than α_i for Π and Π' can be different, and hence $C^{Tree}_{\Pi,\alpha_i} \neq C^{Prob}_{\Pi,\alpha_i}$; thus, this class of problems does not admit complete confidentiality. On the contrary, consider another problem Π'' with the same set of blocks as Π, the same set of arms such that the number of arms controlled by α_i is two, say `Arm1` and `Arm2`, and the same initial arrangement of blocks except that a certain block is held by `Arm1` in Π while it is held by `Arm2` in Π''. Then any agent other than α_i cannot distinguish between Π and Π''. Fig. 1 illustrates such an example with two blocks `A` and `B`. All the actions of Blocksworld are public, and hence all the search states in the figure are public as well. The blue color indicates the public part of the search state. In this example, the blue part of the two depicted search tree of Π and Π'' is the same. Since $\{\Pi, \Pi''\} \subseteq C^{Tree}_{\Pi,\alpha_i}$, then $|C^{Tree}_{\Pi,\alpha_i}| > 1$; given also that $C^{Tree}_{\Pi,\alpha_i} \neq C^{Prob}_{\Pi,\alpha_i}$, we have that this class of problems admits partial confidentiality. □

Although, in general the problems of the CoDMAP Blocksworld domain admit partial confidentiality, it is worth noting that, for the Blocksworld problems used in the competition, each agent controls exactly one arm, and hence these problems admit only a very limited form of confidentiality.

The other CoDMAP domains admitting limited confidentiality are Depot, Sokoban, Woodworking, Wireless, and Taxi. In Depot, the agents are hoists, and the private information includes the driver who guides a truck, and whether a hoist is available or is lifting a crate. Stacking crates by hoists in Depot is similar to stacking blocks by arms in Blocksworld. The confidentiality admitted in Depot is the same as in Blocksworld when each agent controls only one hoist, and this is the case for all the Depot problems of the competition. In Sokoban the agents are players, and the private information consists in the location of players. When a player or a crate is at a location, such a location is not clear, and this is part of the public information. If each agent controls only one player (this is the case for all the Sokoban problems of the competition), then the confidentiality admitted by Sokoban is limited. In Woodworking, the agents are machines for wood processing equipment, and the private information is the property of these machines. Machines with different properties make true different public facts, and this limits the confidentiality of the Woodworking problems. In Wireless, the agents are network nodes, and for this domain the energy of the node is the only private information. The level of the energy is modeled by propositional fluents. There are three energy levels, and energy is consumed when the node generates new data or sends a message. Energy cannot be produced. The confidentiality for the CoDMAP problems is limited because the number of messages that can be transmitted is higher than the capability of transmission determined by the initial energy level of network nodes, and the set of exchanged messages is public. Finally, in Taxi the agents are taxis and passengers. Taxis have no private information, while the only private information for passengers is their final location (encoded by an initial static private proposition). A different final location for a passenger implies that at some time the passenger is at different

locations, which is part of the set of public facts. States where passengers are at different locations are public, shared among agents, and part of the forward search tree of each agent. This implies that $|C^{Tree}_{\Pi,\alpha_i}| = 1$, as the forward search trees of problems with different private parts of α_i include different public states.

Theorem 9. *The problems in CoDMAP domain* Elevators *admit partial confidentiality.*

Proof. In domain Elevators, the agents are elevators, and the public information concerns in whether or not a person is at a "public floor", i.e., a floor that is reachable by more than one elevator. The private information is the floor where each elevator is located, the capacity of each elevator, the floors that are reachable by each elevator, and the initial floors for persons that are initially in a "private floor". Consider two planning problems Π and Π' that have the same set of persons, the same set of floors, and the same set of initial public and private facts for every agent other than α_i. Then, $\Pi' \in C^{Prob}_{\Pi,\alpha_i}$. We prove that $\Pi' \in C^{Tree}_{\Pi,\alpha_i}$ if these two problems have the same set of elevators, and the set of floors that are reachable by each elevator e in Π is the same as in Π'. In this case, every floor where elevator e can be located is reachable from any other possible floor for elevator e and, in spite of the capacity of the elevator, every person who is initially at a floor that is private for elevator e can be moved at any other floor private for elevator e. Confidentiality is not always complete because, e.g., the floors that are reachable by elevator e is a private information, and $\Pi' \notin C^{Tree}_{\Pi,\alpha_i}$ if the set of (public) floors that are reachable by each elevator e in Π is not the same as in Π'. □

The other CoDMAP domains which admit partial confidentiality are DriverLog, Rovers, Satellites, MA-Blocksworld, and MA-Blocksworld-Large. The agents of DriverLog are drivers and the private knowledge includes the position of drivers. Confidentiality is partial for the CoDMAP problems of DriverLog because drivers can walk to more than one location, but they cannot reach every location in the problem by walking, and reaching different locations allows to achieve different public facts. The agents of Satellites are satellites, and the private information for this domain includes the properties of instruments equipping satellites. The use of different instruments makes achievable different public facts. Confidentiality for the CoDMAP problems is partial because satellites can be equipped by more than one instrument, each of which makes achievable the same public facts. The agents of Rovers are rovers, and their private information concerns the waypoints they can reach, their equipment, as well as the images and the rock samples they can take according to their equipment. The confidentiality of the domain is partial because, for problems with a different private part, rovers may communicate different data, the communicated data is public, and hence it is part of the forward-search tree. However, if each rover can reach any waypoint and for each instrument there is at least one rover that is equipped with the instrument, then all the data can be communicated, and problems with a different private part results in the same forward-search tree. The agents of MA-Blocksworld and MA-Blocksworld-Large are stacks of blocks. Each agent controls one or more stacks. Confidentiality for the problems of MA-Blocksworld and MA-Blocksworld-Large would be complete if all the agents had more than three stacks so that the order in which blocks are stacked can be arbitrarily changed without moving blocks to public stacks. However, for all the problems of MA-Blocksworld and MA-Blocksworld-Large there is at least one agent that control less than three stacks.

Theorem 10. *All the problems in domain* Zenotravel *admit complete confidentiality.*

Proof. In Zenotravel, agents are aircrafts, and the public information concerns the initial locations of the persons who are initially located at cities. The private information is the initial fuel level and locations of the aircrafts, and whether or not a person is aboard an aircraft. Consider two planning problems Π and Π' that have the same sets of persons, fuel levels, cities, and the same set of initial public and private facts for each agent other than α_i. Then, $\Pi' \in C^{Prob}_{\Pi,\alpha_i}$. We have that $\Pi' \in C^{Tree}_{\Pi,\alpha_i}$. Indeed, each fuel level of an aircraft is reachable from every other fuel level by moving the aircraft to decrease the fuel level or refueling the aircraft to increase the fuel level, and each location can be (directly) reached by an aircraft from any other location. □

The other CoDMAP domains admitting complete confidentiality are Logistics, MA-Logistics, and MA-Logistics-Large. The agents of Logistics are trucks and airplanes, and the private information in this domain concerns properties of the trucks and airplanes, such as their location. Confidentiality in the Logistics problems of the competition is complete because an airplane can reach each airport from every other airport, and a truck can reach every location inside a city area from every other location inside the same area. The difference between domains MA-Logistics and MA-Logistics-Large w.r.t. Logistics is that for these domains there is no airplane, trucks can move among adjacent cities in the same area, instead of among locations in the same city (like in Logistics), and cities are private. Confidentiality for the problems of MA-Logistics and MA-Logistics-Large is complete because a truck can move a package from every city to every other city in the same area. Table 3 gives a summary in the form of a domain taxonomy, in which problems in the considered domains are classified in terms of the admitted confidentiality.

Table 3
Classification of benchmark domains according to the type of admitted confidentiality.

Limited	Partial	Complete
Blocksworld	DriverLog	Logistics
Depot	Elevators	Zenotravel
Sokoban	Rovers	MA-Logistics
Taxi	Satellites	MA-Logistics-Large
Wireless	MA-Blocksworld	
Woodworking	MA-Blocksworld-Large	

5. Experimental analysis

We present the results of a large experimental study that has the following four main goals: (a) comparing the performance of CMAPP-BFWS and CMAPP-SIW with the state-of-the-art, (b) testing the effectiveness of CMAPP-BFWS using the proposed novelty-based heuristics, (c) evaluating the usefulness of using the novelty for filtering messages, and (d) evaluating the performance of CMAPP-BFWS in a heavily congested distributed network.

5.1. Experimental settings

We implemented the algorithms CMAPP-SIW and CMAPP-BFWS in C++, exploiting the Nanomsg open-source library to share messages [43]. Each agent uses three threads, two of which send and receive messages, while the other one conducts the search, so that the search is asynchronous w.r.t. the communication routines. The behavior of both CMAPP-SIW and CMAPP-BFWS depends on the order with which the messages are received by an agent. Each time a run is repeated, the agents' threads can be differently scheduled by the operating system, and so the behavior of the algorithm can also be different. Thereby, for each problem of our benchmark, we ran CMAPP-SIW and CMAPP-BFWS five times, and we measured the performance metrics of the algorithm as median values over the five runs. When the algorithm exceeded the CPU-time limit for more than two of the five runs, we consider the problem unsolved.

The benchmark used in our experiments includes the twelve CoDMAP domains [40], and the four domains MA-Blocksworld (shortly, MA-BW), MA-Blocksworld-Large (MA-BW-L), MA-Logistics (MA-Log), MA-Logistics-Large (MA-Log-L), which were derived by the work of Maliah et al. (2017) [23]. In the following, these latter four domains are abbreviated to MBS. For all considered domains, the action costs were uniformly defined (i.e. all actions have the same cost).

All tests were run on an InfiniBand Cluster with 512 nodes and 128 Gbytes of RAM, where each node has two 8-cores Intel Xeon E5-2630 v3 with 2.40 GHz. Given a CMAPP-planning problem, for each agent in the problem, we limited the usage of the available resources to 3 CPU cores and 8 GB of RAM. Moreover, unless otherwise specified, the timeout was 5 minutes, after which the termination of all threads was forced.

In the rest of the paper, for each experiment the average values are computed over the problems solved by all the compared approaches. Plan quality is measured in terms of total action cost; therefore, the lower the average plan quality, the better the performance. Time and quality scores are measured by the score functions used for the seventh and ninth International Planning Competitions, respectively. The score function is defined as follows. Concerning planning speed, if a planner P solves a problem π within 1 second, it gets *time score* 1; if it fails to solve π, its score is 0; finally, given 300 seconds as CPU-time limit, if it solves π in t seconds, then its score is $1 - \frac{\log(t)}{\log(300)}$. Concerning plan quality, if P generates a plan with l actions solving π, it gets *quality score* $\frac{l^*}{l}$, where l^* is the number of actions in the shortest plan over those computed by the compared planners for π. If P does not solve π, then it gets zero score (for both speed and quality). The time (quality) score of planner P is the sum of the time (quality) scores assigned to P over all the considered test problems. Higher values indicate better performance.

5.2. Performance of CMAPP-SIW and CMAPP-BFWS w.r.t. the state of the art

First, we compare CMAPP-SIW and CMAPP-BFWS w.r.t. other four existing approaches, PSM [48,49], the best performing configuration of MAPLAN [11], GPPP [24,25], and DPP [26]. Planners PSM and MAPLAN were the best in the CoDMAP competition, while GPPP and DPP were developed after the competition and show performance competitive with the state of the art of distributed collaborative multi-agent privacy-preserving planning.

Table 4 shows the results of this comparison. CMAPP-SIW and CMAPP-BFWS solve all the problems for a number of different domains. The limits of our approach are inherited from the width-based algorithms. Width-based algorithms, such as IW, perform poorly for problems with high width. Variants such as CMAPP-SIW and CMAPP-BFWS try to mitigate the high width of the problems by using serialization or heuristics. When the novelty is used for pruning, the algorithms may become incomplete, while completeness is not compromised if novelty is used as a preference. In general, novelty helps if the paths to the goal have low width, while problems that require reaching states with high width are more challenging.

The results in Table 4 also show that overall CMAPP-BFWS outperforms CMAPP-SIW and performs better than the other compared planners. Remarkably, the only type of information that agents share in our approach is related to the exchanged search states; on the contrary, e.g., MAPLAN also requires sharing the information for the computation of the

Table 4
Number of problems solved by CMAPP-BFWS, CMAPP-SIW, MAPLAN PSM GPPP and DPP for the benchmark problems of CoDMAP and MBS domains. The domains are grouped according with their level of confidentiality. The best performances are indicated in bold. "–" indicates domains with unsupported features or where the planner crashes.

Domain	CMAPP-BFWS	CMAPP-SIW	MAPLAN	PSM	GPPP	DPP
Limited conf.						
Blocksworld	**20**	**20**	**20**	**20**	**20**	16
Depot	**18**	8	12	17	**18**	15
Sokoban	14	4	**17**	16	12	16
Taxi	**20**	**20**	**20**	**20**	**20**	**20**
Wireless	2	0	**4**	0	3	2
Woodworking	12	1	15	**18**	10	12
Partial conf.						
DriverLog	**20**	**20**	16	**20**	9	18
Elevators	**20**	**20**	8	12	**20**	**20**
MA-BW	19	6	–	–	–	–
MA-BW-L	15	0	–	–	–	–
Rovers	**20**	**20**	**20**	19	6	**20**
Satellites	**20**	**20**	**20**	13	**20**	**20**
Complete conf.						
Logistics	**20**	18	18	18	**20**	**20**
MA-Log	**20**	**20**	–	–	–	19
MA-Log-L	**20**	17	–	–	–	19
Zenotravel	**20**	**20**	**20**	10	**20**	**20**
Overall (320)	280	214	190	184	178	237

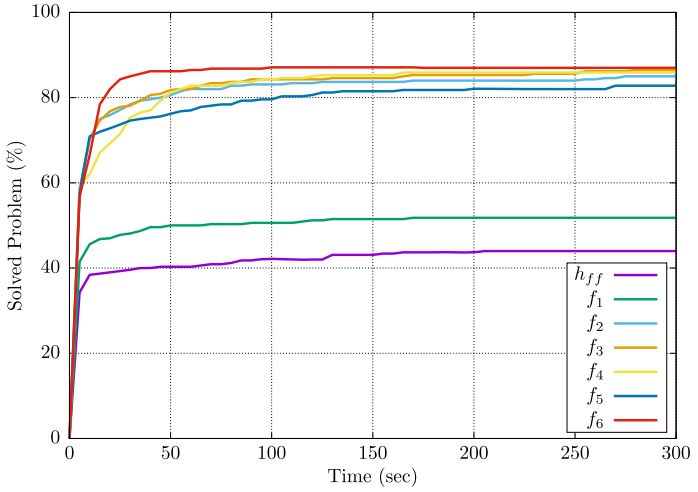


Fig. 2. Coverage as a function of the time for CMAPP-BFWS using seven heuristics for the benchmark problems of the CoDMAP and MBS domains.

search heuristics. In this sense, besides solving a larger set of problems, CMAPP-BFWS exposes less private knowledge to other agents.

Concerning the relative performance of the two best planners, CMAPP-BFWS and DPP, we observed that CMAPP-BFWS is usually faster than DPP: the total time score of CMAPP-BFWS and DPP are 235.8 and 183.0, respectively, and CMAPP-BFWS is on average more than one order of magnitude faster; on the other hand, CMAPP-BFWS computes plans with quality worse than the plans of DPP: the quality score of CMAPP-BFWS and DPP are 215.0 and 225.0 respectively. This is not surprising because the main component of the heuristic used by CMAPP-BFWS is substantially agnostic to the goal distance and hence the search does not prefer to expand states over shorter paths to the goals. The complementarity of CMAPP-BFWS and DPP indicates that these two planners can be fruitfully combined by running them together, in order to quickly obtain a first solution from CMAPP-BFWS, and to obtain a better solution from DPP using extra time.

Note that, for every domain CMAPP-BFWS performs consistently better than or equal to CMAPP-SIW, regardless from the level of confidentiality of the domains used for our analysis. Therefore, in the rest of the experimental analysis we focus only on CMAPP-BFWS.

Table 5
Number of problems solved by CMAPP-BFWS with seven different heuristics for the benchmarks problems of the CoDMAP and MBS domains. The domains are grouped according with their level of confidentiality. The best performance is in bold.

Domain	h_{FF}	f_1	f_2	f_3	f_4	f_5	f_6
Limited confidentiality							
Blocksworld	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Depot	5	9	17	17	18	**19**	18
Sokoban	**17**	**17**	**17**	**17**	**17**	16	14
Taxi	16	17	**20**	**20**	**20**	**20**	**20**
Wireless	**2**	**2**	**2**	**2**	**2**	**2**	**2**
Woodworking	3	4	11	**16**	**16**	10	12
Partial confidentiality							
DriverLog	19	**20**	**20**	**20**	**20**	**20**	**20**
Elevators	3	11	**20**	**20**	**20**	**20**	**20**
MA-BW	0	1	18	17	17	15	**19**
MA-BW-L	0	0	7	8	5	3	**15**
Rovers	15	19	**20**	**20**	**20**	**20**	**20**
Satellites	18	19	**20**	**20**	**20**	**20**	**20**
Complete confidentiality							
Logistics	3	7	**20**	**20**	**20**	**20**	**20**
MA-Log	0	0	**20**	**20**	**20**	**20**	**20**
MA-Log-L	0	0	**20**	**20**	**20**	**20**	**20**
Zenotravel	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Overall (320)	141	166	272	277	275	265	**280**

5.3. *Performance of* CMAPP-BFWS *using novelty-based heuristics*

The next experiment we conducted regards the usage of six novelty-based heuristics with algorithm CMAPP-BFWS: $f_1 = \langle w_{(h_{\mathsf{FF}})}, h_{\mathsf{FF}}\rangle$, $f_2 = \langle w_{(G_\perp, h_{\mathsf{FF}})}, G_\perp, h_{\mathsf{FF}}\rangle$, $f_3 = \langle w_{(G_u, G_\perp, h_{\mathsf{FF}})}, G_u, G_\perp, h_{\mathsf{FF}}\rangle$, $f_4 = \langle w_{(G_u, G_\perp, h^p_{\mathsf{FF}})}, G_u, G_\perp, h^p_{\mathsf{FF}}\rangle$, $f_5 = \langle w_{(G_\perp, \#r)}, G_\perp, \#r\rangle$, and $f_6 = \langle w_{(G_\perp, \#r_l)}, G_\perp, \#r_l\rangle$. Planner CMAPP-BFWS($h_{\mathsf{FF}}$) is the baseline for this comparison, since it does not use novelty-based preferences to guide the search. Table 5 shows the number of problems solved by CMAPP-BFWS using these heuristics for the benchmark problems of CoDMAP and MBS. For five out of sixteen considered domains, CMAPP-BFWS with h_{FF} solves almost all the problems. These are the domains with problems that require less interaction among agents.

CMAPP-BFWS with f_1 solves few more problems than h_{FF}, and the domains where CMAPP-BFWS(f_1) performs well are the same as those with h_{FF}. Surprisingly, CMAPP-BFWS with f_2 solves many more problems than with h_{FF} and f_1. The main difference between f_2 and f_1 is that the novelty-based exploration gives preference according to the number of unachieved goals $G_\perp$. This clearly results in a positive interplay with the search algorithm. Interestingly, CMAPP-BFWS with f_2 solves many problems of the CoDMAP domains such as Logistics, Depot, and Woodworking, and several problems from the MBS domains.

CMAPP-BFWS with f_3 or f_4 solves few more problems than with f_2, showing that the information about the number of unreachable goals from search states can be useful. Heuristic f_5 is computationally less expensive than f_3 and f_4, but the goal-directed component of f_5 is less accurate. The results in Table 5 show that the tradeoff between computational cost and accuracy of f_5 does not pay off. CMAPP-BFWS with f_5 is better than both with h_{FF} and with f_1, but it solves fewer problems than with f_2, f_3, and f_4. The reason of this behavior is that, when the number of incoming messages is high, heuristic f_5 is computationally still quite expensive.

The cheapest heuristic function to compute is f_6, since the most expensive step in the computation of our heuristics is the RPG construction, and f_6 constructs the RPG only once. The results in Table 5 indicate that f_6 is a good tradeoff between accuracy and computational cost, since CMAPP-BFWS with f_6 solves the largest set of problems. It solves several problems even for domain MA-Blocksworld, which are unsolved by using any other heuristic function.

Fig. 2 shows the coverage of CMAPP-BFWS with the seven heuristic functions using a time limit ranging from 0 to 300 seconds. With a time limit of few seconds, the best heuristic is f_4; with a time limit between 5 and 25 seconds, f_3 is the best; with a time limit between 25 seconds and 300 seconds, CMAPP-BFWS using heuristic f_6 solves the largest set of problems. Interestingly, the coverages obtained using a time limit of 150 seconds are substantially the same as using 300 seconds.

Table 6 shows the performance of CMAPP-BFWS with the proposed search heuristics in terms of average time, plan length, number of exchanged messages, number of expanded states, time score, and quality score. CMAPP-BFWS with f_4 is on average the fastest, and the average numbers of exchanged messages and expanded states of f_4 are the lowest, closely followed by f_2. Remarkably, the average number of exchanged messages and expanded states of CMAPP-BFWS with h_{FF} and f_1 are about one order of magnitude greater than with the other heuristics.

Table 6
Average time, plan length, thousands of exchanged messages, thousands of expanded states, time and quality scores of CMAPP-BFWS using seven heuristics for the benchmarks of the CoDMAP and MBS domains.

Metric	h_{FF}	f_1	f_2	f_3	f_4	f_5	f_6
Average CPU sec.	10.56	2.81	1.54	2.05	**1.47**	2.01	2.01
Average quality	61.91	**56.49**	62.0	61.38	62.31	101.14	69.16
Average k-Mess.	595.74	118.24	13.94	18.69	**11.44**	23.73	20.03
Average k-States	385.12	97.01	15.92	21.22	**13.3**	56.06	57.84
Time score	115.28	136.78	237.05	239.98	233.05	235.74	**249.24**
Quality score	119.55	142.76	224.07	**226.62**	214.95	187.4	222.44

Table 7
Performance of CMAPP-BFWS with *num_waiting* (abbreviated with nw) set to 1, *half*, and *all* in terms of average CPU time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	$nw = all$	$nw = 1$	$nw = half$
Average CPU seconds	**5.01**	8.57	5.13
Average quality	185.89	185.79	**184.75**
Average k-Messages	**75.32**	543.43	76.69
Average k-States	144.26	308.11	**144.22**
Solved problems	259	279	**280**
Time score	211.07	220.8	**224.44**
Quality score	222.81	239.46	**240.31**

Table 8
Performance of CMAPP-BFWS with $num\_waiting = half$, considering *who_send* (abbreviated with ws) set to *wait*, *not wait*, and *all* in terms of average CPU time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	$ws = wait$	$ws = not\ wait$	$ws = all$
Average CPU seconds	5.59	6.04	**5.5**
Average quality	180.0	**178.61**	179.64
Average k-Messages	**88.7**	109.72	92.79
Average k-States	227.6	228.48	**178.97**
Solved problems	279	278	**280**
Time score	**224.6**	223.33	224.44
Quality score	238.03	238.58	**240.39**

5.4. Performance of CMAPP-BFWS filtering messages according to their novelty

In this section we evaluate the usefulness of filtering messages according to their novelty. We experimentally study different conditions under which an agent can decide to transmit (a part of) its withheld states. We denote by w^{out} the maximum value of outgoing novelty a state s can have without being withheld. E.g., $w^{out} = 1$ means that states with outgoing novelty greater than one are withheld. In our experiments, unless differently specified, CMAPP-BFWS is used with $w^{out} = 1$, $num\_waiting = half$, $who\_send = all$, $num\_withheld\_states = group$, and heuristic function $f = f_6 = \langle w_{(G_\perp, \#r_l)}, G_\perp, \#r_l \rangle$. The novelty measure $w_{(G_\perp, \#r_l)}$ used for guiding the search is also used for filtering messages.

The first experiment we conducted concerns the decision about when an agent sends its withheld states. The results in Table 7 show that $num\_waiting = all$ is the configuration that sends the fewest states, but it is also the one with the far lowest coverage. This is probably due to the fact that requiring that all agents are in waiting state is a condition that reduces the transmission of withheld states too much. With $num\_waiting = 1$ many more states than with other configurations are sent, as every agent sends its *withheld* states as soon as a single agent is waiting. With $num\_waiting = half$, CMAPP-BFWS obtains a good tradeoff. Considering all the measures of performance, $num\_waiting = half$ is the best configuration except for the average number of sent messages and the average CPU time. However, for those measures the performance gap with respect to the best configuration is quite limited.

In Table 8, we experimentally evaluate the performance of CMAPP-BFWS for different configurations of *who_send*, that specifies which agents send withheld states. We can see that there is no big gap in the performance of these configurations. For the other experiments in the paper, we use $who\_send = all$ because it performs slightly better in terms of coverage and average time, while the average number of exchanged states is pretty close to the best value obtained for $who\_send = waiting$.

Table 9 shows the results for different configurations of *num_withheld_states*. With $num\_withheld\_states = none$, the withheld states are not sent; the obtained performance is similar to the one obtained with $num\_waiting = all$ (in Table 7), confirming that with this setting too few states are shared. By sending one state at a time, i.e., with $num\_withheld\_states =$

Table 9
Performance of CMAPP-BFWS with *num_waiting* = *half*, *who_send* = *all*, and *num_withheld_states* (abbreviated with *nws*) set to 1, *all*, and *group* in terms of average CPU time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	*nws=none*	*nws*=1	*nws=all*	*nws=group*
Average CPU seconds	**5.05**	5.28	5.22	5.27
Average quality	186.13	186.11	**185.27**	186.22
Average k-Messages	**79.31**	83.74	85.55	82.72
Average k-States	**146.67**	180.31	151.72	153.73
Solved problems	258	272	275	**280**
Time score	209.85	219.02	221.48	**224.44**
Quality score	222.2	231.85	236.78	**238.51**

Table 10
Performance of CMAPP-BFWS without outgoing novelty filtering, with outgoing novelty w^{out} equal to 1 and 2 in terms of average CPU time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	no filtering	$w^{out} = 1$	$w^{out} = 2$
Average CPU seconds	16.32	**6.14**	9.69
Average quality	179.79	**174.53**	181.26
Average k-Messages	1214.2	**100.6**	547.18
Average k-States	480.18	**236.85**	323.74
Solved problems	277	280	**285**
Time score	201.3	**224.44**	215.73
Quality score	229.51	**240.3**	239.25

1, we have an average number of exchanged messages and an average execution time very close to the configuration for which states are sent in group (*num_withheld_states* = *group*), although the average number of expanded states is higher, and eight fewer problems are solved. Finally, as expected, with *num_withheld_states* = *all* we have the highest number of exchanged messages; probably more than necessary, given that the solved problems are fewer than with *num_withheld_states* = *group*. This shows that increasing the number of sent messages has a computational cost, since the average number of expanded states within the time limit used for our experiments is lower than with *num_withheld_states* set to 1 or *group*.

In Table 10 we show the results of CMAPP-BFWS without filtering messages and using values of outgoing novelty w^{out} equal to 1 and 2. It is important to remark that the computation and memory cost of determining that the outgoing novelty of a state is k has complexity exponential in k, since all the tuples of size up to k but one may be stored and considered. For efficiency, we simplify the computation of the outgoing novelty to only 2 levels for $w^{out} = 1$, and only 3 levels for $w^{out} = 2$, i.e., for $w^{out} = 2$ the outgoing novelty of shared states can be equal to 1, 2, or greater than 2.

Configuration $w^{out} = 1$ shows the best performance. The benefits of this configuration are:

- A slight improvement of the coverage w.r.t. CMAPP-BFWS without message filtering.
- The overall number of exchanged messages is drastically reduced (by 91%). As previously noted, this reduction improves the privacy of planners.
- The average execution time is considerably reduced (by more than 50%).
- There is an increase in the plan quality. This is probably due to the fact that, by holding states with outgoing novelty greater than 1, the priority is given to states that can be reached using a number of actions at most equal to the number of propositions of the problem, i.e., states that in the worst case can be reached by plans shorter than with outgoing novelty greater than 1.

Also CMAPP-BFWS with $w^{out} = 2$ leads to a sharp decrease in the average CPU time compared to CMAPP-BFWS without novelty filtering. However this is less than with $w^{out} = 1$ for two reasons: (i) determining that the outgoing novelty for $w^{out} = 1$ is computationally much cheaper than for $w^{out} = 2$; (ii) CMAPP-BFWS with $w^{out} = 1$ sends fewer states, and probably fewer superfluous states. Compared to CMAPP-BFWS without novelty filtering, the average number of messages exchanged by CMAPP-BFWS with $w^{out} = 2$ is lower, and the number of solved problem is higher (eight more than without filtering messages).

Fig. 3 shows the percentage of sent messages w.r.t. the maximum total amount of messages generated for all the solved problems. We can observe that, compared to CMAPP-BFWS without filtering of messages, for CPU-time limits greater than 170 seconds, the use of outgoing novelty equal to 1 reduces the number of exchanged messages by one order of magnitude.

Table 11 shows the performance of *secure*-CMAPP-BFWS, which corresponds to algorithm CMAPP-BFWS with $f = \langle w_{(\#g)}, d \rangle$ and *num_withheld_states* = *none*. Procedure *secure*-CMAPP-BFWS with $w^{out} < \infty$ withholds *only* the states with the same public projection of a state that has been sent before; *secure*-CMAPP-BFWS with w^{out} equal to 1 or 2 with-

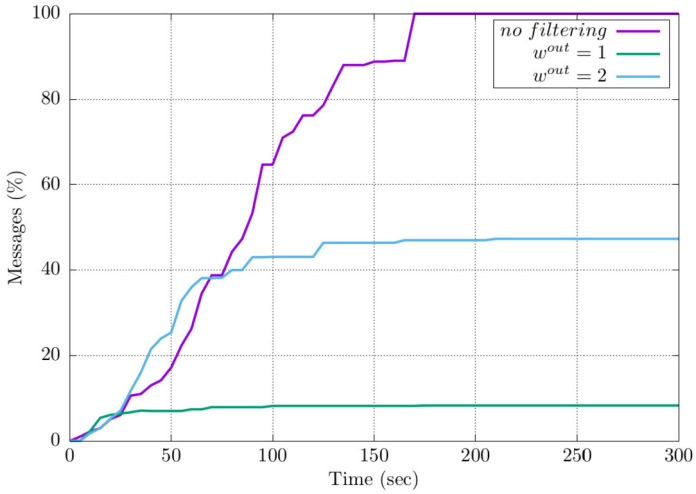


Fig. 3. Percentage of messages sent by CMAPP-BFWS without outgoing novelty filtering and by CMAPP-BFWS with outgoing novelty w^{out} equal to 1 and 2 using different CPU-time limits (from 0 to 300 seconds).

Table 11
Performance of *secure*-CMAPP-BFWS with outgoing novelty filtering w^{out} equal to 1, 2, and lower than ∞, in terms of average CPU time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	*secure*-CMAPP-BFWS with $w^{out} = 1$	$w^{out} = 2$	$w^{out} < \infty$
Average CPU seconds	**5.79**	10.42	15.55
Average quality	**172.56**	181.85	182.3
Average k-Messages	**87.15**	525.73	1036.47
Average k-States	**184.23**	304.79	412.9
Solved problems	278	**283**	277
Time score	**223.2**	211.57	202.84
Quality score	**240.76**	239.73	233.76

holds further states. Such a procedure substantially implements the "secure check" proposed by Brafman (2015) [9] for the exchanged messages. It is incomplete but strong privacy-preserving over domains with complete confidentiality, because the heuristic function f is independent from the private parts of the problem and the procedure does not send states whose public projection has been sent before (no withheld state is sent since $num\_withheld\_states = none$). As pointed up in Section 4, the domains in our benchmark for which the privacy can be strongly preserved by *secure*-CMAPP-BFWS are Logistics, Zenotravel, MA-Logistics, and MA-Logistics-Large.

Comparing the results in Table 11 with those in Table 10, we can observe a very limited decrease in terms of coverage, and an improvement in terms of transmitted messages and expanded states. Moreover, procedure *secure*-CMAPP-BFWS performs very similar to the other considered version of CMAPP-BFWS, better than the planners in the state of the art, and most importantly, differently from the other procedures and planners, it preserves the privacy strongly over the domains with complete confidentiality.

5.5. *Performance of* CMAPP-BFWS *in a heavily congested distributed network*

In this section we present an experiment aimed at testing how CMAPP-BFWS with filtering of messages performs in a heavily congested distributed network, as well as how sensitive is novelty filtering to randomised message ordering. With this aim, we introduced a mechanism that, by means of an artificial delay applied on each exchanged message, can simulate arbitrarily network delays during message transmission. These delays are distributed according to the gamma distribution that is an approximation of the delays in the Internet network [42]. In particular, Table 12 considers 5 different configurations of delays, with a standard deviation equal to 10% of the average delay.

For average delays smaller than 100 ms, in terms of coverage, we observe no significant performance gap w.r.t. CMAPP-BFWS with no delay in the transmission of the messages. Instead, with very high delays, as expected the number of solved problems decreases. However, even with an average delay of 10 seconds applied to each exchanged message, the number of solved problems does not decrease to few units. The results in Table 13 show that all domains but Depot, Taxi, MA-Log, and MA-Log-L do not require an intensive cooperation among agents. Only for these four domains, CMAPP-BFWS

Table 12
Performance of CMAPP-BFWS with no delay in the transmission of messages and with an average delay in the transmission of each message equal to 10 ms, 100ms, 1s, and 10s, in terms of average CPU-time, plan cost, thousands of sent messages, thousands of expanded states, number of solved problems, time and quality scores.

Metric	no delay	10ms	100ms	1s	10s
Average CPU seconds	**4.89**	5.0	6.56	22.14	162.92
Average quality	153.13	148.28	141.27	138.52	**136.26**
Average k-Messages	**17.94**	23.45	19.11	57.82	651.98
Average k-States	**62.14**	71.59	92.76	365.39	1374.01
Solved problems	279	**280**	**280**	246	194
Time score	**265.14**	261.1	215.12	127.06	68.48
Quality score	234.17	241.08	**252.83**	218.65	167.95

Table 13
Coverage of CMAPP-BFWS with no delay in the transmission of the messages, with an average delay in the transmission of each message equal to 10 ms, 100ms, 1s, and 10s. The domains are grouped according with their level of confidentiality. The best performances are indicated in bold.

Domain	no delay	10ms	100ms	1s	10s
Limited confidentiality					
Blocksworld	**20**	**20**	**20**	**20**	19
Depot	18	**19**	**19**	9	5
Sokoban	**14**	13	**14**	12	12
Taxi	**20**	**20**	18	0	0
Wireless	**2**	**2**	**2**	1	0
Woodworking	11	12	**13**	**13**	11
Partial confidentiality					
DriverLog	**20**	**20**	**20**	**20**	19
Elevators	**20**	**20**	**20**	**20**	19
MA-BW	**19**	**19**	**19**	**19**	**19**
MA-BW-L	**15**	**15**	**15**	**15**	**15**
Rovers	**20**	**20**	**20**	**20**	**20**
Satellites	**20**	**20**	**20**	**20**	**20**
Complete confidentiality					
Logistics	**20**	**20**	**20**	18	15
MA-Log	**20**	**20**	**20**	**20**	0
MA-Log-L	**20**	**20**	**20**	**20**	0
Zenotravel	**20**	**20**	**20**	19	**20**

with no delay, or small average delay, solves all or almost all the problems; while CMAPP-BFWS using 10 seconds as average delay solves no problem or very few problems.

Furthermore, when the delay increases, we can observe a significant increase of the execution time. The required CPU time varies from a matter of seconds for CMAPP-BFWS without delay, to hundreds of seconds for the highest average delay considered in our experiment. The higher number of expended states when the average delay is high is probably due to the fact that, in this case, the agents can spend more time for the state expansion. Moreover, concerning the plan quality score, the best value is found with an average delay equal to 100ms; the quality score obtained with 10 seconds as average delay is drastically lower, showing that without cooperation among agents the plan cost becomes high.

6. Related work

The use of width-based search for CMAPP planning has already been investigated. In particular, IW search was used for solving a classical planning problem obtained from the compilation of a CMAPP-planning problem [28]. However, this previous work applies to *centralized* CMAPP planning, while our work investigates width-based search in the context of the distributed CMAPP-planning problem.

MAFS [29–31] is a CMAPP-planning algorithm similar to ours. As discussed before, MAFS is a distributed best-first search that for each agent considers a separate search space. MAFSB is an enhancement of MAFS that uses a form of backward messages to reduce the number of search states shared by the agents [22]. In contrast, using width-based search techniques, the number of exchanged states can be limited by pruning the states that have a novelty greater than a given bound.

The work by Torreño et al. on distributed A* for partial-order CMAPP planning has the same motivations on preserving the agents' privacy as our work [44–47]. Differently from this approach, our CMAPP-planning algorithm searches in the space of world states, rather than in the space of partial plans, and it exchanges states among agents rather than partial plans. Another planner that conducts the search in a state space different from ours is PSM [48,49]. It uses non-deterministic finite state machines to represent sets of partial multi-agent plans, and exchanges their public projection among agents.

GPPP is a CMAPP planner that uses two different phases to find a solution plan [24,25]. First, the agents compute a joint coordination scheme by solving a special relaxed problem using only the public actions of the problem. Then, each agent searches for a local plan that achieves the preconditions of the public actions in the coordination scheme. If some agent fails, the process is repeated by computing a new coordination scheme. Similarly, Maliah et al. proposed the DPP system that uses the regression to build "virtual" preconditions that maintain the dependencies between the actions of the agents [26]. In this work, each agent identifies when a (public) action is required before another (public) action, even through a chain of other private actions, then creates a virtual effect in the required action that is consumed as precondition by the second action: those may be exchanged and used by agents to more effectively produce high level plans. Our approach is completely different, as it conducts a distributed best-first width search.

Concerning the related work on developing heuristics for CMAPP planning, Štolba and Komenda [35,36] proposed a *distributed* algorithm that computes a complete relaxed planning graph and, subsequently, extracts a relaxed plan from the distributed relaxed planning graph. When computing the heuristic function, in the worst case every agent is involved, and this can make the computation slow because of a potentially very heavy communication overhead. Further research has been focusing on avoiding too much (costly) communication among the agents. The most promising heuristics in this scope are "potential heuristics" [33,38]. Moreover, Štolba and Komenda developed MADLA [39], a CMAPP planner which exploits a multi-heuristic search which combines the use of a local and a distributed heuristic. Similarly, Torreño et al. [47] showed that the alternate use of two different heuristics, one of which is computed using the information of a single agent while the other one is computed using the global information, can pay off the communication cost. Differently from these techniques, the heuristics presented in Section 3.2 combine the novelty measure of the search states with an estimate of the cost required to achieve the problem goals, which is computed using the information of a single agent. Such an estimate is more inaccurate but computationally much cheaper than distributed heuristics.

MAPLAN [11] translates a CMAPP planning problem encoded by the MA-STRIPS language into its corresponding (distributed) SAS+ representation, and exploits four heuristics based on variants of the LM-cut heuristic [6] and the Fast-Forward (FF) heuristic [37]. The computation of these heuristics is done either by a single agent using the public projection of public actions or is distributed among agents. Other search heuristics are proposed in [26,32,46]. They are landmarks-based heuristics, and a heuristic that is based on the domain transition graph, augmented by a special node in the graph that represents when the agent does not know the whole domain of a variable. In contrast, our heuristics strongly relies on the novelty measure of the search states. Another important difference between our approach and the existing ones using heuristic search for distributed CMAPP planning is that with our approach the public projection of public actions is not shared. We conjecture that without sharing such a projection it is more difficult to infer the private preconditions and effects of the public actions, since the agents ignore their existence. While sharing the public projection of public actions may be useful to compute more accurate search heuristic, our approach is competitive with the state-of-the-art planners without using it.

Brafman [9] provided the notion of strongly private algorithm for a set of problems, under the condition that, for each agent α_i, each pair of problems in the set shares the same projection of the forward search tree obtained by removing the private part of α_i from each state in the tree. Moreover, Brafman showed that such a condition holds for the set of problems in domains Logistics, Satellites, and Rovers. Štolba et al. [41] give a different notion of strongly private algorithm that is based on the set of problems that share the same public projection as well as the same set of public solutions. They call such a set of problems a public equivalence class. Our definitions of problem-equivalent class and forward-search-tree-equivalent class are notions similar to the ones given by Brafman [9] and Štolba et al. [41]. We used them not only to identify problems admitting complete confidentiality, but also to distinguish problems admitting limited confidentiality. Finally, we used the introduced notions of problem classes to classify the planning domains of the CoDMAP competition, which are a set of domains different from those considered by Brafman [9].

7. Conclusions

Goal-directed search is the main computational approach that has been investigated in classical planning and, subsequently, in CMAPP planning. In CMAPP planning, the goal of agents is to compute a joint plan that achieves mutual goals by keeping certain information private to the individual agents. However, during the search each agent exchanges information with others, and the exchanged information may be used to infer private knowledge [9].

In this paper, we have investigated the width-based search framework, originally developed for single-agent problems, for effectively addressing the CMAPP planning problems. First we introduced a new search algorithm, CMAPP-SIW, which solves the synchronization issues arising with the usage of Iterative Width search in CMAPP planning. Differently from the existing approaches to collaborative multi-agent planning, CMAPP-SIW conducts a "blind" search, and hence it does not require that agents exchange heuristic values. As a consequence, we claim that the agents' privacy can be better preserved by CMAPP-SIW than by other existing approaches to CMAPP planning.

Then we investigated width-based exploration in the form of novelty-based preferences. In classical planning, novelty-based preferences provide an effective complement to goal-directed search. However, in order to preserve privacy, in the CMAPP-planning setting, the private actions and the private part of the public actions are not shared, and this makes the goal-directed heuristics less informed than in classical planning. Moreover, the encryption of the private knowledge that the agents share during the search affects the measure of novelty. Nevertheless, we showed that the combination of computationally cheap goal-directed heuristics and width-based search is effective also for CMAPP planning.

A theoretical study presented in this paper classifies the benchmark domains according with the type of admitted privacy. This study shows that only domains Zenotravel and Logistics, together with the MA-Logistics version of this latter domain, admit complete privacy; for these domains, any agent cannot distinguish between a pair of problems which differ in only private parts of another agent.

Finally, we have experimentally demonstrated that the novelty-based techniques are very useful to significantly reduce the number of messages transmitted among agents, better preserving their privacy levels, and improving performance. The results of our experiments show the effectiveness of our techniques, especially of CMAPP-BFWS, improving the state of the art in terms of problem coverage (solved benchmark problems). The *secure*-CMAPP-BFWS variant of CMAPP-BFWS has turned out to be competitive with the state-of-the-art planners used in our experiments, while having the significant advantage w.r.t. these planners that it preserves privacy more strongly. Moreover, we observed that our approach is robust to delays in the transmission of messages that could occur in overloaded networks.

This opens up the possibility of increasing privacy preserving properties of CMAPP planning algorithms. For instance, given the success of black-box planning for single agents [12], we plan to investigate the implications of fully protected models given as black-boxes, and the usage of privacy-preserving set operations for secure multi-party computation [17]. Another direction for future work regards a classification of planning domains according with the chance of maintaining the private information of agents confidential, considering structural properties of the domains such as the existence of reversible actions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

We would like to thank the reviewers for their thoughtful comments and efforts towards improving the paper. This research has been carried out with the support of resources from the National Collaborative Research Infrastructure Strategy (NeCTAR), the Department of Excellence Fund 2018-2022 of the Department of Information Engineering of the University of Brescia, and by AIPlan4EU, a project funded by EU Horizon 2020 research and innovation programme under GA n. 101016442 (since 2021). The work of A. E. Gerevini was supported by the EU ICT-48 2020 project TAILOR (n. 952215) and by the PRIN project RIPER (n. 20203FFYLK). Nir Lipovetzky has been partially funded by DST group.

References

[1] W. Bandres, B. Bonet, H. Geffner, Planning with pixels in (almost) real time, in: Proc. of AAAI, 2018.
[2] G. Bazzotti, A.E. Gerevini, N. Lipovetzky, F. Percassi, A. Saetti, I. Serina, Iterative width search for multi agent privacy-preserving planning, in: AI*IA 2018 - XVIIth International Conference of the Italian Association for Artificial Intelligence, Springer, 2018, pp. 431–444.
[3] A. Beimel, R.I. Brafman, Privacy preserving multi-agent planning with provable guarantees, arXiv:1810.13354, 2018.
[4] A.L. Blum, M.L. Furst, Fast planning through planning graph analysis, Artif. Intell. 90 (1997) 281–300.
[5] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell. 129 (2001) 5–33.
[6] B. Bonet, M. Helmert, Strengthening landmark heuristics via hitting sets, in: Proc. of ECAI, 2010.
[7] A. Bonisoli, A.E. Gerevini, A. Saetti, I. Serina, A privacy-preserving model for the multi-agent propositional planning problem, in: Proc. of ECAI, 2014.
[8] A. Bonisoli, A.E. Gerevini, A. Saetti, I. Serina, A privacy-preserving model for multi-agent propositional planning, J. Exp. Theor. Artif. Intell. 30 (2018) 481–504.
[9] R.I. Brafman, A privacy preserving algorithm for multi-agent planning and search, in: Proc. of IJCAI, 2015.
[10] R.I. Brafman, C. Domshlak, From one to many: planning for loosely coupled multi-agent systems, in: Proc. of ICAPS, 2008.
[11] D. Fišer, M. Štolba, A. Komenda, Maplan, in: Proc. of the ICAPS Competition on Distributed and Multi-Agent Planners, ICAPS, 2015.
[12] G. Frances, M. Ramírez, N. Lipovetzky, H. Geffner, Purely declarative action descriptions are overrated: classical planning with simulators, in: Proc. of IJCAI, 2017.
[13] A.E. Gerevini, N. Lipovetzky, N. Peli, F. Percassi, A. Saetti, I. Serina, Novelty messages filtering for multi agent privacy-preserving planning, in: Symposium on Combinatorial Search, 2019.
[14] A.E. Gerevini, N. Lipovetzky, F. Percassi, A. Saetti, I. Serina, Best-first width search for multi agent privacy-preserving planning, in: Proc. of ICAPS, 2019.
[15] J. Hoffmann, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (2001) 253–302.
[16] M. Katz, N. Lipovetzky, D. Moshkovich, A. Tuisov, Adapting novelty to classical planning as heuristic search, in: Proc. of ICAPS, 2017.
[17] L. Kissner, D. Song, Privacy-preserving set operations, in: Annual International Cryptology Conference, Springer, 2005, pp. 241–257.
[18] R. van der Krogt, Quantifying privacy in multiagent planning, Multiagent Grid Syst. 5 (2009) 451–469.
[19] N. Lipovetzky, H. Geffner, Width and serialization of classical planning problems, in: Proc. of ECAI, 2012.
[20] N. Lipovetzky, H. Geffner, Width-based algorithms for classical planning: new results, in: Proc. of ECAI, 2014.
[21] N. Lipovetzky, H. Geffner, Best-first width search: exploration and exploitation in classical planning, in: Proc. of AAAI, 2017.
[22] S. Maliah, R.I. Brafman, G. Shani, Increased privacy with reduced communication and computation in multi-agent planning, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2016.
[23] S. Maliah, R.I. Brafman, G. Shani, Increased privacy with reduced communication in multi-agent planning, in: Proc. of ICAPS, 2017.

[24] S. Maliah, G. Shani, R. Stern, Privacy preserving landmark detection, in: Proc. of ECAI, 2014.
[25] S. Maliah, G. Shani, R. Stern, Privacy preserving pattern databases, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2015.
[26] S. Maliah, G. Shani, R. Stern, Stronger privacy preserving projections for multi-agent planning, in: Proc. of ICAPS, 2016.
[27] S. Maliah, R. Stern, G. Shani, Privacy preserving LAMA, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2016.
[28] C. Muise, N. Lipovetzky, M. Ramirez, MAP-LAPKT: omnipotent multi-agent planning via compilation to classical planning, in: Proc. of ICAPS Competition on Distributed and Multi-Agent Planning, ICAPS, 2015.
[29] R. Nissim, R.I. Brafman, Multi-agent A* for parallel and distributed systems, in: Proc. of the ICAPS Workshop on Heuristics and Search for Domain-independent Planning, ICAPS, 2012.
[30] R. Nissim, R.I. Brafman, Cost-optimal planning by self-interested agents, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2013.
[31] R. Nissim, R.I. Brafman, Distributed heuristic forward search for multi-agent planning, J. Artif. Intell. Res. 51 (2014) 293–332.
[32] M. Štolba, D. Fišer, A. Komenda, Admissible landmark heuristic for multi-agent planning, in: Proc. of ICAPS, 2015.
[33] M. Štolba, D. Fišer, A. Komenda, Potential heuristics for multi-agent planning, in: Proc. of ICAPS, 2016.
[34] M. Stolba, D. Fiser, A. Komenda, Privacy leakage of search-based multi-agent planning algorithms, in: Proc. of ICAPS, 2019.
[35] M. Štolba, A. Komenda, Fast-forward heuristic for multiagent planning, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2013.
[36] M. Štolba, A. Komenda, Relaxation heuristics for multiagent planning, in: Proc. of ICAPS, 2014.
[37] M. Stolba, A. Komenda, Relaxation heuristics for multiagent planning, in: S.A. Chien, M.B. Do, A. Fern, W. Ruml (Eds.), Proc. of ICAPS, 2014.
[38] M. Štolba, A. Komenda, Computing multi-agent heuristics additively, in: Proceedings of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2016.
[39] M. Štolba, A. Komenda, The MADLA planner: multi-agent planning by combination of distributed and local heuristic search, Artif. Intell. 252 (2017) 175–210.
[40] M. Štolba, A. Komenda, D.L. Kovacs, Competition of distributed and multiagent planners (CODMAP), in: Proc. of AAAI, 2016.
[41] M. Štolba, J. Tožička, A. Komenda, The limits of strong privacy preserving multi-agent planning, in: Proc. of the ICAPS Competition on Distributed and Multi-Agent Planners, ICAPS, 2015.
[42] A.M. Sukhov, N.Y. Kuznetsova, What type of distribution for packet delay in a global network should be used in the control theory?, arXiv:0907.4468, 2009.
[43] M. Sustrik, Nanomsg, http://nanomsg.org/, 2016.
[44] A. Torreño, E. Onaindia, Ó. Sapena, An approach to multi-agent planning with incomplete information, in: Proc. of ECAI, 2012.
[45] A. Torreño, E. Onaindia, Ó. Sapena, Fmap: a heuristic approach to cooperative multi-agent planning, in: Proc. of the ICAPS Workshop on Distributed and Multi-Agent Planning, ICAPS, 2013.
[46] A. Torreño, E. Onaindia, Ó. Sapena, Fmap: distributed cooperative multi-agent planning, Appl. Intell. 41 (2014) 606–626.
[47] A. Torreño, Ó. Sapena, E. Onaindia, Global heuristics for distributed cooperative multi-agent planning, in: Proc. of ICAPS, 2015.
[48] J. Tožička, J. Jakubuv, A. Komenda, Generating multi-agent plans by distributed intersection of finite state machines, in: Proc. of ECAI, 2014.
[49] J. Tožička, J. Jakubuv, A. Komenda, Psm-based planners description for codmap 2015 competition, in: Proc. of ICAPS, 2017.
[50] J. Tožička, M. Štolba, A. Komenda, The limits of strong privacy preserving multi-agent planning, in: Proc. of ICAPS, 2017.