Concise Finite-Domain Representations for Factored MA-PDDL Planning Tasks

Daniel Fišer[1] and Anotnín Komenda[1]
[1]*Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague*
danfis@danfis.cz, antonin.komenda@fel.cvut.cz

Abstract: Planning tasks for the distributed multi-agent planning in deterministic environments are described in highly expressive, but lifted, languages, similar to classical planning. On the one hand, these languages allow for the compact representation of exponentially large planning problems. On the other hand, the solvers using such languages need efficient grounding methods to translate the high-level description to a low-level representation using facts or atomic values. Although there exist ad-hoc implementations of the grounding for the multi-agent planning, there is no general scheme usable by all multi-agent planners. In this work, we propose such a scheme combining centralized processes of the grounding and the inference of mutex groups. Both processes are needed for the translation of planning tasks from the Multi-agent Planning Description Language (MA-PDDL) to the finite domain representation. We experimentally show a space reduction of the multi-agent finite domain representation in contrast to the binary representation on the common benchmark set.

1 INTRODUCTION

The problem of decision making is one of the core areas in the spotlight of the artificial intelligence research from its early days. Classical planning formalizes and proposes algorithmic solutions for problems where the decisions has to be made in a sequence and in an environment known beforehand. The (distributed) multi-agent planning stemmed from the classical planning research and it focuses on sequential decision making of *cooperative* agents and in a *deterministic* environment. The domain-independent nature of multi-agent planning requires from the solvers to be able to efficiently plan for problems in various domains as logistics, transportation, manufacturing, multi-robotic systems, space, or military missions.

To this date, several multi-agent planners were proposed, implemented and compared to each other at the Competition of Distributed and Multi-agent Planners (Komenda et al., 2016). All competing planners were obliged to use a common input language for multi-agent planning—MA-PDDL (Kovacs, 2012), which became de-facto standard. In order to provide high expressiveness, the language describes the input planning problem in a high-order representation that is not directly used for planning. That is why, to our best knowledge, all existing multi-agent planners use a process called grounding, which enumerates possible decisions (actions) required for solving the planning problem. The process has to be efficient and complete in the sense that it does not miss to ground any action needed in the solution of the problem.

The grounding process in the existing distributed multi-agent planners, namely MAPlan (Fišer et al., 2015), PSM planner (Tozicka et al., 2016), and FMAP (Torreño et al., 2014) is based on implementation of the classical planning suite Fast Downward (Helmert, 2006). The grounding in Fast Downward is well tested and reasonably efficient, however it is not directly usable in the multi-agent planning as it is not designed as a distributed algorithm. Its benefit is that it grounds the high-order representation into grounded finite-domain representation, which is not limited to binary facts as in the well-known STRIPS model (Bylander, 1994). To ground a planning task to the finite-domain representation, the algorithm finds out, what values cannot hold together in one state. (Helmert, 2006) proposes one such *mutex group* inference mechanism, which is again, not directly usable for the multi-agent planning.

In this paper, we propose a novel and general scheme for the translation of high-order multi-agent problems in MA-PDDL to the finite-domain representation. The scheme uses centralized processes of grounding and inference of mutex groups as black boxes implementable by various existing techniques.

2 FACTORED MA-PDDL

The Planning Domain Definition Language (PDDL) is the de-facto standard language for representing classical planning tasks. PDDL was introduced for the first International Planning Competition (IPC) in 1998 (McDermott, 2000). PDDL is based on a subset of predicate logic and uses the LISP syntax for describing planning tasks (Fig. 2 lists an example PDDL). As PDDL was extended in many ways, most applications of PDDL uses only selected fragments.

Since PDDL is in a *lifted* (parameterized) form, it is able to represent (exponentially) large planning tasks compactly. In PDDL, a planning task describes a model of a world in terms of *objects*, *predicates* describing relations between the objects, and *actions* that manipulate these relations. The elements are described in the `:objects`, `:predicates`, and `:action` LISP expressions, respectively. The parameters are prefixed by `?` and typically limited by types using the syntax `?parameter - type`. The actions are defined over two expressions: `:precondition` and `:effect`. The `:precondition` expression is a logical formula describing the condition of applicability of the action. The `:effect` expression is a logical formula describing the result after application of the action. To solve a planning task, the planning algorithm—the planner—has to find a sequence of instantiated actions from the *initial state* (the PDDL expression `:init`) to one of the final states described by the *goal specification* (the PDDL expression `:goal`). Most of the existing planners translate the input PDDL specification into a propositional representation by *grounding* the predicates and actions. The process of grounding generates possible instantiations of the predicates and actions using the world objects. Such grounded *facts* (from predicates) and *grounded actions* are later used in the planning algorithm to find a solution. Some planners go even further and construct more concise representation (requiring smaller amount of bits), such as the finite domain representation (FDR) or SAS$^+$ (Bäckström and Nebel, 1995), using inferred *invariants* over reachable parts of the state space. FDR uses assignment of values to variables describing the facts which hold in a state of the world. As only one value from the finite domain of possible values can be assigned to a variable, the representation has to assure there is no reachable state requiring more than one value assigned. This process is based on automated inference of mutually exclusive sets of facts invariant over the reachable fragment of the planning task.

The MA-PDDL extension of PDDL for multi-agent planning was proposed by (Kovacs, 2012). In its original version, it included many aspects inherited from PDDL, which were not necessary for the minimalistic multi-agent planning model MA-STRIPS, proposed by (Brafman and Domshlak, 2008), which is used as the prevalent grounded planning model and as a basis for the multi-agent finite domain representations. Additionally, the original MA-PDDL did not describe privacy of facts and actions, which is a fundamental part of the MA-STRIPS model. Therefore the original MA-PDDL was simplified to a variant used in CoDMAP. The CoDMAP variant of MA-PDDL defines only two additional aspects to PDDL: (i) partitioning of the planning task to multiple agents and (ii) privacy of objects, predicates, and implicitly privacy of actions (for more details see (Komenda et al., 2016)). The MA-PDDL extension allows for defining agents in various ways, but most notably as the world objects. This principle simplifies reuse of many existing single-agent planning tasks.

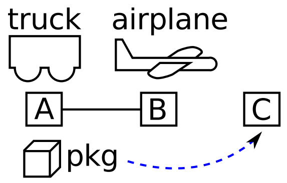

Figure 1: A schema of a simple logistic problem.

In factored MA-PDDL, each planning agent uses its own planning sub-task, denoted as a MA-PDDL factor. A MA-PDDL factor is defined by a pair of domain and problem description files which define information relevant only to one particular agent. Fig. 1 depicts a simple multi-agent logistics planning problem. The domain and problem MA-PDDL files (factors) of the example problem are listed in Fig. 2. There are two agents representing a truck (`truck`) and an airplane (`airplane`) transporting a package (`pkg`). The initial state of the planning task is depicted in the figure and the goal condition is that the package `pkg` is moved to the location `C`. `truck` can move only between the locations `A` and `B` and `airplane` can fly only between `B` and `C`. Both vehicles can load or unload `pkg` at the location where the package is present or if it is in the vehicle, respectively. The solution to the problem is simply to load `pkg` by `truck` at `A`, drive `truck` from `A` to `B`, unload the package at `B`, load it by the airplane at `B`, fly the airplane to `C` and unload the package at `C` from `airplane`.

The actions grounded from actions in the MA-PDDL factors are partitioned to the agents directly from the decomposition of the planning task, as each MA-PDDL factor contains only relevant actions. Public facts, objects, and constants which are common for more than one agent are bound over the same names. For instance the `pkg - package` of the truck

The domain definition for both the truck and the airplane agent:

```
(define (domain logistics-truck)
  (:requirements :factored-privacy :typing)
  (:types location vehicle package)
  (:predicates
    (at ?obj - object ?loc - location)
    (in ?obj1 - package ?v - vehicle)
    (:private
      (in-city ?v - vehicle ?loc - location)
    )
  )
  (:action load
    :parameters
      (?v - vehicle ?p - package ?l - location)
    :precondition
      (and (in-city ?v ?l) (at ?p ?l))
    :effect
      (and (not (at ?p ?l)) (in ?p ?v))
  )
  (:action unload
    :parameters
      (?v - vehicle ?p - package ?l - location)
    :precondition
      (and (in-city ?v ?l) (in ?p ?v))
    :effect
      (and (not (in ?p ?v)) (at ?p ?l))
  )
  (:action move
    :parameters
      (?v - vehicle ?f - location ?t - location)
    :precondition (in-city ?v ?f)
    :effect
      (and (not (in-city ?v ?f))
           (in-city ?v ?t))
  )
)
```

The problem definition for the truck agent:

```
(define (problem logistics-truck)
  (:domain logistics-truck)
  (:objects
    pkg - package B - location
    (:private A - location truck - vehicle)
  )
  (:init (at pkg A) (in-city truck A))
  (:goal (in-city truck A))
)
```

The problem definition for the airplane agent:

```
(define (problem logistics-airplane)
  (:domain logistics-airplane)
  (:objects
    pkg - package B - location
    (:private
       C - location airplane - vehicle)
  )
  (:init (in-city airplane C))
  (:goal (at pkg C))
)
```

Figure 2: A simple logistic problem as factored MA-PDDL.

agent, in the example, is the same package as `pkg` in the factored problem description of the airplane agent. The `:private` blocks specify what predicates and objects has to be treated secret of the agents. Using MCG and decomposition to the agents then specifies whether the grounded facts and actions has to be private or public. The agents' capabilities are clearly defined by partitioning to MA-PDDL factors. To highlight that a factored MA-PDDL is a result of factoring (and thus may contain `:private` declarations), the additional requirement `:factored-privacy` is used in the domain factor file.

3 FACTORED MA-FDR

A **factored MA-FDR problem** is a pair $\mathcal{M} = \langle \mathcal{A} = \{1,...,n\}, \Pi = \{\Pi_i\}_{i\in\mathcal{A}}\rangle$, where $\mathcal{A}$ is a set of agents and Π is a set of individual factors—each factor corresponding to an individual problem for each agent. An **MA-FDR factor** of an agent i is a tuple $\Pi_i = \langle \mathcal{V}_i = \mathcal{V}^{\text{pub}} \cup \mathcal{V}_i^{\text{priv}}, O_i, I_i, \mathcal{G}_i\rangle$.

$\mathcal{V}_i$ is a finite set of **variables**, where each variable $V \in \mathcal{V}_i$ has an associated finite domain $\mathcal{D}_V$. A **partial state** is a function s on a subset $\mathcal{V}_i(s)$ of $\mathcal{V}_i$, so that $s(V) \in \mathcal{D}_V$ for all $V \in \mathcal{V}_i(s)$; s is a **state** if $\mathcal{V}_i(s) = \mathcal{V}_i$. I_i is the **initial state** and the **goal** $\mathcal{G}_i$ is a partial state. The set of variables is partitioned into the set of **public variables** $\mathcal{V}^{\text{pub}}$, common to all factors, and the set of **private variables** $\mathcal{V}_i^{\text{priv}}$, known only to the agent i. $\mathcal{V}^{\text{pub}}$ and $\mathcal{V}_i^{\text{priv}}$ are pairwise disjoint, i.e., $\mathcal{V}^{\text{pub}} \cap \mathcal{V}_i^{\text{priv}} = \emptyset$ for every $i \in \mathcal{A}$ and $\mathcal{V}_i^{\text{priv}} \cap \mathcal{V}_j^{\text{priv}} = \emptyset$ for every $i, j \in \mathcal{A}$ such that $i \neq j$.

O_i is a set of operators. Each **operator** $o \in O_i$ is a tuple $o = \langle \text{pre}_o, \text{eff}_o\rangle$, where pre_o and eff_o are both partial states; pre_o is called a **precondition**, and eff_o is called an **effect**. An operator o is **applicable** in a state s if $s(V) = \text{pre}_o(V)$ for every $V \in \mathcal{V}_i$. In that case, the result of applying o in s, denoted as $s[o]$, is another state such that $s[o](V) = \text{eff}_o(V)$ if $V \in \mathcal{V}_i(\text{eff}_o)$ and $s[o](V) = s(V)$ otherwise.

A sequence of operators $\pi = \langle o_1, ..., o_n\rangle$ is applicable in a state s_0 if there are states $s_1, ..., s_n$ such that o_i is applicable in s_{i-1} and $s_i = s_{i-1}[o_i]$ for $1 \leq i \leq n$. The resulting state of this application is $s_0[\pi] = s_n$. A set of **local reachable states** $\mathcal{R}_i$ is a set of states $s \in \mathcal{R}_i$ such that there exists a sequence of operators π such that $I_i[\pi] = s$. A **local plan** is a sequence of operators π such that $I_i[\pi](V) = \mathcal{G}_i(V)$ for every $V \in \mathcal{V}_i(\mathcal{G}_i)$.

Let $\mathcal{V} = \bigcup_{i\in\mathcal{A}} \mathcal{V}_i$, a **global partial state** is a function g on a subset $\mathcal{V}(g)$ of $\mathcal{V}$ such that $g(V) \in \mathcal{D}_V$ for all $V \in \mathcal{V}(g)$; g is a **global state** if $\mathcal{V}(g) = \mathcal{V}$. The valuation of the initial states over all public variables is the same, i.e., for every $i, j \in \mathcal{A}$ it holds that $I_i(V) = I_j(V)$ for every $V \in \mathcal{V}^{\text{pub}}$, and similarly for the goals. The **global initial state** I is a global state such that for every $i \in \mathcal{A}$ and every $V \in \mathcal{V}_i$ it holds that $I(V) = I_i(V)$, and similarly for the **global goal** $\mathcal{G}$ it holds that $\mathcal{V}(\mathcal{G}) = \bigcup_{i \in \mathcal{A}} \mathcal{V}(\mathcal{G}_i)$ and $\mathcal{G}(V) = \mathcal{G}_i(V)$ for every $i \in \mathcal{A}$ and every $V \in \mathcal{V}_i(\mathcal{G}_i)$. A **global sequence of operators** $\pi = \langle o_1, ..., o_n \rangle$ is a sequence of operators from $\bigcup_{i \in \mathcal{A}} O_i$ applicable in global states. A set of **global reachable states** $\mathcal{R}$ is a set of states $g \in \mathcal{R}$ such that there exists a global sequence of operators π such that $I[\pi] = g$. A **global plan** is a global sequence of operators π such that $I[\pi](V) = \mathcal{G}(V)$ for every $V \in \mathcal{V}(\mathcal{G})$.

A solution to a factored MA-FDR (or just MA-FDR from now on) is a global multi-agent plan consisting of operators from different factors. The solution is searched for locally by individual agents in their respective factors (Torreno et al., 2017).

4 GROUNDING

Grounding of the lifted MA-PDDL representation is a process of instantiation of predicates and actions by replacing all occurrences of parameters by the world objects. For example, in the problem of the truck agent (in Fig. 2), the predicate `(at ?obj ?loc)` can be grounded using objects `pkg` and `A` to the fact `(at pkg A)`, and the action `(load ?v ?p ?l)` can be grounded using objects `truck`, `pkg`, and `A` to the grounded action `(load truck pkg A)`. Since the grounding replaces all occurrences of the parameters by the corresponding objects, grounding of an action requires also grounding of the predicates listed in its preconditions and effects. So the grounded action `(load truck pkg A)` requires grounded facts `(in-city truck A)`, `(at pkg A)`, and `(in pkg truck)`.

The grounding usually requires dealing with logical formulas (containing, e.g., conjunctions, quantifiers, or implications) that appear in the preconditions and effects of actions, and in the goal specification. The formulas need to be transformed into form that the particular planner “understands”, e.g., they need to be flattened into simple conjunctions of facts. However, the multi-agent grounding algorithm we propose uses a (single-agent or local) grounding algorithm as a *black box* that is called repeatedly. So, we are not interested in the particular way actions are grounded and, as we describe in the next section, the translation of the grounded actions into the MA-FDR operators is also independent of our algorithm. The only requirement on the black box is that it is somehow based on the reachability of facts from the initial state. More precisely, the black box is a procedure that has two inputs and one output. The inputs are a MA-PDDL specification and a set of reachable facts. The output is a set of facts that are reachable from the input facts through grounding of lifted actions and application of those actions on the reachable facts.

Although a correct grounding requires only that the grounded problem contains, at minimum, the grounded actions that appear in the solution, determining the minimum set of the grounded actions is as hard as planning itself. Therefore, the usual way to generate the grounding is to use a delete-free relaxation, i.e., delete effects are disregarded and only add effects are considered. The grounding starts with the initial state, which is always a set of grounded facts, and proceeds with finding actions that can be grounded so that the preconditions contain only already grounded facts, i.e., grounding of the action does not require grounding of any additional predicate. These actions are grounded and their add effects are grounded into new facts. This procedure is repeated until a fixpoint where no new facts can be added. For example, this algorithm with several improvements is used by (Edelkamp and Helmert, 1999). The grounding algorithm proposed by (Helmert, 2009) constructs Datalog program from the lifted representation and the program generates reachable facts and grounded actions. Both of these algorithms can be used as a black box for our algorithm, because they can be repeatedly called with a different set of reachable facts. Both use the delete-free relaxation, so we can always replace the input initial state with the set of reachable facts and re-run the grounding procedure.

Algorithm 1 shows the pseudo-code of the multi-agent grounding algorithm which we propose. Each agent runs the listed algorithm locally on its respective MA-PDDL factor. The highlighted function on the line 3, **Ground**, is the black box function that takes a MA-PDDL factor as its input and a set of the facts that are currently recognized as reachable and returns a new set of grounded operators and a new set grounded facts.

The algorithm starts with initialization of output sets (line 1) and then proceeds with the loop that runs until no agent can ground any new actions or facts. The loop starts with calling the black box function for grounding of new actions O and facts F (line 3). Then the public facts P are selected from F (line 4), and the new public facts are sent to all other agents (line 5). In the next step (line 6), the agent receives public facts R sent from any other agent (or from more agents at once if the public facts are available). Lastly (line 7

Algorithm 1: MA-PDDL problem grounding.

```
Input: MA-PDDL problem $\mathcal{P}_i$
Output: Set of grounded actions $G_O$, set of
        grounded facts $G_F$
1 $G_O \leftarrow \emptyset$; $G_F \leftarrow \emptyset$;
2 until fixpoint do
3     $O, F \leftarrow$ Ground($\mathcal{P}_i, G_F$); // black-box
4     $P \leftarrow$ SelectPublicFacts($F$);
5     SendToAll($P$);
6     $R \leftarrow$ ReceiveFromAny();
7     $G_O \leftarrow G_O \cup O$;
8     $G_F \leftarrow G_F \cup F \cup R$;
9 end
```

and 8), the grounded actions and all the grounded facts, including the ones that were received from other agents, are collected into output sets.

The loop runs until a fixpoint is reached globally by all agents, i.e., until no agents can ground any more actions or facts. The implementation determining the fixpoint is application dependent. It can be determined in a centralized way or by a distributed algorithm such as the snapshot algorithm (Mattern, 1987).

If the black box function provides all the public facts necessary, the proposed algorithm guarantees a correct grounding of all factors, because the agents communicate all public facts to each other, which is everything the agents can know from each other. Both aforementioned algorithms for the grounding (Edelkamp and Helmert, 1999; Helmert, 2009) fulfill this requirement and both can be used as the black box function in our algorithm.

5 INFERENCE OF MUTEX GROUPS AND CONSTRUCTION OF VARIABLES

A **mutex group** M is a set of facts of which maximally one can be part of any global reachable state $s \in \mathcal{R}$, i.e., it is an invariant with respect to all global reachable states stating that either zero or one fact from M can be part of any global reachable state. Mutex groups are especially useful for the construction of variables from a grounded (MA-)PDDL task. Since no two facts from a mutex group can be part of the same state, it is always safe to create one variable from each mutex group, so that each fact corresponds to one value of the variable. Sometimes, an additional value $\perp$ must be added to the variable to cover the situation when none of the facts from the mutex group is a part of the state. It is usually desirable to construct the set of variables as concise as possible so every fact is represented only by one value. For example, if we have two mutex groups $\{f_1, f_2, f_3\}$ and $\{f_2, f_3, f_4\}$, we create two variables V_1 and V_2 with the values $\mathcal{D}_{V_1} = \{f_1, f_2, f_3, \perp\}$ and $\mathcal{D}_{V_2} = \{f_4, \perp\}$. This way, the variable V_1 can be stored using only two bits and the variable V_2 using only one bit.

Moreover, the most concise representation can be created if we could somehow infer all the maximum sized mutex groups the grounded planning task contain. However, it is as hard as planning itself to infer a complete set of mutex groups. So in practice, the inference algorithms opt to some sort of approximation, i.e., a sound but incomplete algorithm is usually used. Similarly to the grounding phase, we build upon a black box function that is able to produce a set of mutex groups that are local to each particular factor. Basically any algorithm for the inference of mutex groups can be used (Edelkamp and Helmert, 1999; Helmert, 2009). Before we start with the description of the distributed algorithms for the construction of MA-FDR variables, let us first clarify two things.

First, the constructed variables must obey the definition of MA-FDR laid out in Section 3, i.e.:

- The private facts of each factor must be separated to the private variables and an agent cannot communicate any private fact to any other agent and the same holds for the private parts of the inferred mutex groups.
- The public variables must be created from the public facts and the public variables must be the same for each and every agent. In other words, the agents must somehow agree on the way the public variables are constructed.

Second, the algorithm we propose utilizes a simple property of mutex groups: Every subset of a mutex group is also a mutex group. It is easy to see that if at most one fact from a mutex group can be part of any reachable state, then the same must hold for any subset of that mutex group. This also means that a single fact is always a mutex group and we will further assume that the black box used for the inference of the local mutex groups will always return mutex groups covering all the known facts. E.g., if we have facts f_1, f_2, f_3 and the black box algorithm used would generate a mutex group $\{f_1, f_2\}$, then we assume that it generates also the trivial mutex group $\{f_3\}$ so that each fact belongs to at least one mutex group.

Algorithm 2 shows the pseudo code of the distributed algorithm for the inference of (global) mutex groups; each agent runs this algorithm locally with its MA-PDDL factor and its grounded facts and actions obtained from Algorithm 1. The algorithm returns

Algorithm 2: Inference of mutex groups.

```
Input: MA-PDDL problem $\mathcal{P}_i$, set of grounded operators $G_O$, set of grounded facts $G_F$
Output: Set of public mutex groups $\mathcal{M}^{\text{pub}}$, set of private mutex groups $\mathcal{M}^{\text{priv}}$
/* black box */
1  $\mathcal{M} \leftarrow$ MutexGroups($\mathcal{P}_i, G_O, G_F$);
2  $\mathcal{M}^{\text{pub}}, \mathcal{M}^{\text{priv}} \leftarrow$ SplitCandidates($\mathcal{M}$);
3  SendToAll($\mathcal{M}^{\text{pub}}$);
4  for each agent $j \neq i$ do
5      $\mathcal{M}_j \leftarrow$ ReceiveFromAgent($j$);
6      $\mathcal{M}^{\text{pub}} \leftarrow \{M \mid M_i \in \mathcal{M}^{\text{pub}}, M_j \in \mathcal{M}_j, M = M_i \cap M_j, M \neq \emptyset\}$;
7  end

8  function SplitCandidates($\mathcal{M}$)
9      $\mathcal{M}^{\text{pub}} \leftarrow \emptyset$; $\mathcal{M}^{\text{priv}} \leftarrow \emptyset$;
10     for each $M \in \mathcal{M}$ do
11         $P \leftarrow$ SelectPublicFacts($M$);
12         $\mathcal{M}^{\text{pub}} \leftarrow \mathcal{M}^{\text{pub}} \cup \{P\}$;
13         $\mathcal{M}^{\text{priv}} \leftarrow \mathcal{M}^{\text{priv}} \cup \{M \setminus P\}$;
14     end
15     return $\mathcal{M}^{\text{pub}}, \mathcal{M}^{\text{priv}}$;
16 end
```

private and public mutex groups that will be used to construct private and public variables, respectively.

On the line 1, the initial set of mutex group candidates is obtained using the black box function **Mutex-Groups**, which returns the sets of facts that are mutex groups with respect to that particular factor. Then, the mutex group candidates are split between private and public parts using the function **SplitCandidates**.

The set $\mathcal{M}^{\text{priv}}$ contains the mutex groups that contain private facts only. These mutex groups are final, because they correspond to the private part of each respective factor and this part is neither communicated to other agent nor it can be influenced by any other agent. The private variables are constructed directly from these mutex groups.

The set $\mathcal{M}^{\text{pub}}$ contains the mutex group candidates consisting of public facts only. These candidates need to be further refined to get global mutex groups from which the public variables can be constructed. The refinement of the candidates is described on the lines 3 to 7 and it is based on two simple rules. First, every subset of a mutex group is a mutex group (as already discussed). Second, a set of facts that is a local mutex group in all factors is certainly a global mutex group. In other words, given a set of facts M, if we can prove that the mutex group property of M cannot be violated

Algorithm 3: Construction of MA-FDR variables.

```
Input: Set of public mutex groups $\mathcal{M}^{\text{pub}}$, set of private mutex groups $\mathcal{M}^{\text{priv}}$, set of grounded facts $G_F$
Output: Set of variables $\mathcal{V}^{\text{pub}}$, $\mathcal{V}^{\text{priv}}$
1  $\mathcal{V}^{\text{pub}} \leftarrow$ ConstructVariables($\mathcal{M}^{\text{pub}}, G_F$);
2  $\mathcal{V}^{\text{priv}} \leftarrow$ ConstructVariables($\mathcal{M}^{\text{priv}}, G_F$);

3  function ConstructVariables($\mathcal{M}, G_F$)
4      $\mathcal{V} \leftarrow \emptyset$ /* output set of variables */
5      $E \leftarrow \emptyset$ /* set of encoded facts */
6      while $E \neq G_F$ do
7          $M \leftarrow$ SelectMutexGroup($\mathcal{M}$);
8          Create a new variable $V$;
9          $\mathcal{D}_V \leftarrow M \cup \{\bot\}$;
10         $\mathcal{V} \leftarrow \mathcal{V} \cup \{V\}$;
11         $E \leftarrow E \cup M$;
12         $\mathcal{M} \leftarrow \{X \mid Y \in \mathcal{M}, X = Y \setminus U, X \neq \emptyset\}$;
13     end
14     return $\mathcal{V}$;
15 end
```

in any individual factor, then M has the mutex group property with respect to all global reachable states.

So, each agent sends to every other agent its local copy of $\mathcal{M}^{\text{pub}}$ and also receives $\mathcal{M}^{\text{pub}}$ from every other agent (lines 3 and 5). Then it computes intersections of all possible combinations of mutex groups from all agents, e.g., if we have three factors $1, 2, 3$ and three sets of mutex groups $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ for each agent, respectively, then we compute intersections $M_1 \cap M_2 \cap M_3$ for every $M_1 \in \mathcal{M}_1$, every $M_2 \in \mathcal{M}_2$ and every $M_3 \in \mathcal{M}_3$. That is, it uses a brute force to find the subsets of mutex groups that are mutex groups in all individual factors. The non-empty sets are the global mutex groups from which it is safe to construct MA-FDR public variables. Algorithm 2 computes the intersections of combinations iteratively using mutex groups from one agent after another. This way the empty intersections are filtered out as early as possible, so it reduces computational burden, because not all combinations must be enumerated.

Note that all agents compute intersections from the same sets, so every agent computes the exactly same set of mutex groups. This enables to compute the exactly same set of public variables by each agent without actually communicating with each other. So the only data that need to be communicated between agents in this phase are the public parts of the local mutex group candidates.

Algorithm 3 shows the pseudo code for the algorithm that constructs variables from the given sets of

Table 1: The minimal number of bits required for storing a global state given the constructed variables and the time (in seconds) spent in the whole translation process; **#ps**: the number of problems, **ma**: the proposed algorithm, **bin**: binary encoding of variables, **priv**: private variables, **pub**: public variables.

domain	#ps	priv		pub		priv + pub		time
		ma	bin	ma	bin	ma	bin	
blocksworld	20	**340**	1 068	**1 314**	3 669	**1 654**	4 737	30.8
depot	20	**514**	1 238	5 559	5 559	**6 073**	6 797	86.4
driverlog	20	**344**	2 270	**1 076**	3 821	**1 420**	6 091	123.2
elevators	20	**1 934**	2 405	**1 376**	3 070	**3 310**	5 475	40.3
logistics	20	**1 536**	1 571	**952**	1 726	**2 488**	3 297	23.1
rovers	20	**4 720**	7 610	1 635	1 635	**6 355**	9 245	86.3
satellites	20	**1 119**	4 265	2 552	2 552	**3 671**	6 817	65.9
sokoban	20	**321**	2 328	**1 483**	2 902	**1 804**	5 230	21.5
taxi	20	0	0	**865**	1 085	**865**	1 085	26.8
wireless	20	**1 619**	1 815	**11 440**	11 724	**13 059**	13 539	126.6
woodworking	20	**65**	142	**7 469**	7 527	**7 534**	7 669	145.0
zenotravel	20	**1 834**	2 895	**1 240**	4 710	**3 074**	7 605	106.4
Σ	240	**14 346**	27 607	**36 961**	49 980	**51 307**	77 587	882.4

private and public mutex groups. It is, again, assumed that every fact is contained in at least one mutex group. Both the private and the public variables are created using the function ConstructVariables. The main loop of the function runs as long as there are some facts that are not encoded as a value in any variable. In each cycle, a mutex group is selected (line 7) and a new variable is created from that mutex group with added $\perp$ value (lines 8 to 10). Since we do not want to have one fact encoded twice as two values in two different variables, the encoded facts are removed from all the remaining mutex groups (line 12).

The selection of mutex groups on the line 7 determines the quality of the variable encoding. The optimal encoding, in a number of bits required for encoding a state, is NP-Complete, so the planners usually implement a suboptimal but fast algorithm. These algorithms can be implemented inside the function SelectMutexGroup. The only restriction on the function is that in the case of public mutex groups, the function returns the same mutex groups in the same order for all agents. This is easy to implement, because all agents have exactly the same set $\mathcal{M}^{\text{pub}}$, therefore using some sort of a strict sorting is enough. This ensures that each agent has exactly the same set of public variables.

6 EXPERIMENTAL EVALUATION

The proposed algorithm was experimentally evaluated on all domains from CoDMAP (Komenda et al., 2016). The experiments ran on the Intel Core i5-6200U (2.30 GHz) processor with 8 GB memory. The algorithm was implemented into Fast Downward's (Helmert, 2006) preprocessor. Each agent ran in a separate process and the agents communicated over the TCP/IP protocol. We have measured the mini-

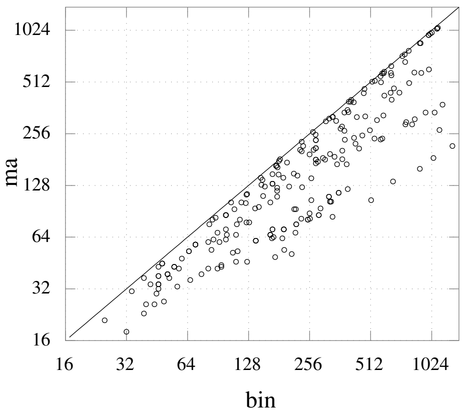

Figure 3: The minimal number of bits required for storing the constructed variables as a scatter plot for all tested problems.

mal number of bits required for the encoding of a global state given the constructed public and private variables and the time spent in the whole translation process. The number of bits required for the binary encoding is used as a baseline for the comparison. For example, having three variables V_1, V_2, and V_3 with the values $D_{V_1} = \{f_1, f_2, \perp\}$, $D_{V_2} = \{f_3, \perp\}$, and $D_{V_3} = \{f_4, f_5, f_6, \perp\}$, respectively, the minimal number of bits required for the encoding of a global state is $2 + 1 + 2 = 5$. Whereas in the binary encoding (used, e.g., in Fast Forward planner (Hoffmann and Nebel, 2001)), each of the values $f_1, ..., f_6$ is encoded in the separate variable, therefore 6 bits are required (one bit per fact).

Table 1 and Fig. 3 show the experimental results. The average time spent in the translation process of a single problem was a little more than 3.5 seconds. Since the typical amount of time allocated for the planners in planning competitions is 30 minutes (5 minutes for the agile track), the average time spent in the translation can be considered negligible, leaving most of the time for the solving of the problems. This is counterintuitive, considering that the inference of the mutex groups requires the computation of intersections between mutex groups from all agents, which could generate an exponential number of mutex groups. However, the experiments show that the proposed iterative method of the computation significantly reduces the number of intersections that need to be computed, at least in the tested domains.

The comparison of the minimal number of bits required for storing a global state shows that the encoding of the private parts of the states is considerably more concise with our algorithm than the simple binary encoding in all tested domains except for the taxi domain, which does not contain any private facts. In

3 out of 12 domains (depot, rovers, satellites), our algorithm produces binary encoding of the public variables. However, in the rest, our algorithm produced more concise encoding than the baseline.

Considering both private and public parts of the global states, the binary encoding requires, overall, 1.5 times more bits than the one produced by our algorithm. In the domains depot, wireless, and woodworking, the encoding resulting from our algorithm is very close to the baseline binary encoding. However, in the rest of the domains, our algorithm proved to generate a substantially more concise encoding.

7 CONCLUSIONS

We have proposed a novel general scheme for the translation of multi-agent planning tasks from MA-PDDL to MA-FDR using the grounding and the inference of mutex groups. The grounding process iteratively runs a centralized grounding algorithm, which generates grounded facts and actions from the agent's MA-PDDL factor. All public facts are send to all other agents, which add them into their own sets of grounded facts. In the next iteration, all that grounded facts are used to ground more facts and actions, and this process continues until a fixpoint is reached. The construction of the MA-FDR variables is based on the inference of mutex groups. Each agent locally uses a centralized algorithm for the inference of mutex groups to find the candidate mutex groups. The mutex groups are split between the public and the private parts and, similarly to the grounding process, the public parts are communicated to all other agents. Then they are processed so that the global public mutex groups are generated. The private mutex groups are kept local, i.e., they are not communicated with other agents while preserving privacy of the agents' factors. The translation scheme was implemented with the state-of-the-art grounding and mutex group inference algorithms (Helmert, 2006) and used on the CoDMAP benchmark set to show a reduction of the multi-agent finite domain representation memory requirements in the comparison with the binary representation of the planning task. The finite domain representations show around 33% space reduction to the binary representation.

In future work, we plan to improve the computational efficiency of the translation process with tailored grounding and mutex group inference algorithms. Another interesting direction is to use the proposed scheme in other distributed multi-agent planners and experimentally compare its effect on the complete planning process efficiency.

ACKNOWLEDGEMENTS

This research was supported by the Czech Science Foundation (grant no. 15-20433Y) and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13).

REFERENCES

Bäckström, C. and Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656.

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 28–35.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204.

Edelkamp, S. and Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In *Recent Advances in AI Planning, 5th European Conference on Planning (ECP)*, pages 135–147.

Fišer, D., Štolba, M., and Komenda, A. (2015). MAPlan. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP)*, pages 8–10.

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

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5–6):503–535.

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

Komenda, A., Stolba, M., and Kovacs, D. L. (2016). The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115.

Kovacs, D. L. (2012). A multi-agent extension of PDDL3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (WIPC), 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 19–27.

Mattern, F. (1987). Algorithms for distributed termination detection. *Distributed Computing*, 2(3):161–175.

McDermott, D. V. (2000). The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55.

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

Torreno, A., Onaindia, E., Stolba, M., and Komenda, A. (2017). Cooperative multi-agent planning: A survey (in print). *ACM Computing Surveys*.

Tozicka, J., Jakubuv, J., Komenda, A., and Pechoucek, M. (2016). Privacy-concerned multiagent planning. *Knowl. Inf. Syst.*, 48(3):581–618.