ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-237

A Single-Agent Approach to Multiagent Planning

Matthew Crosby [1] and **Anders Jonsson** [2] and **Michael Rovatsos** [1]

Abstract. In this paper we present a novel approach to multiagent planning in domains with concurrent actions and associated concurrent action constraints. In these domains, we associate the actions of individual agents with subsets of objects, which allows for a transformation of the problems into single-agent planning problems that are considerably easier to solve. The transformation forces agents to select joint actions associated with a single subset of objects at a time, and ensures that the concurrency constraints on this subset are satisfied. Joint actions are serialised such that each agent performs their part of the action separately. The number of actions in the resulting single-agent planning problem turns out to be manageable in many real-world domains, thus allowing the problem to be solved efficiently using a standard single-agent planner. We also describe a cost-optimal algorithm for compressing the resulting plan, i.e. merging individual actions in order to reduce the total number of joint actions. Results show that our approach can handle large problems that are impossible to solve for most multiagent planners.

1 INTRODUCTION

Many real-world planning domains, e.g. those inspired by applications in transportation, logistics, manufacturing and job-shop scheduling, involve multiple agents acting concurrently in the same environment, either to achieve a shared goal, or multiple (potentially conflicting) different goals. Over the past twenty years, multiagent planning has received much attention in both the automated planning [1, 15] and multiagent systems [7, 8] communities, and contributions in the area address various sub-problems, such as the coordination of agents that plan independently [4], methods for merging individual agents' plans [16], and the computation of plans that are acceptable to a set of self-interested planning agents with different goals [3].

For multiagent planning problems with concurrent actions, the problem of synthesising a plan for all agents involved is significantly harder than its single-agent counterpart. This is especially true when there are explicit constraints on which concurrent actions are allowed. Having all agents act in parallel may lead to a potentially exponential increase in the number of actions that have to be considered in each step, and it is unlikely that we can come up with algorithms that scale to large numbers of agents with many interdependencies between them in the worst case. However, it turns out that, in many practical domains, most of these interdependencies arise from agents jointly manipulating the state of *objects* in relatively limited numbers of ways, and that the number of agents involved in such manipulation is usually rather small. Also, many actions that can be taken by an agent (including those not involving such joint object manipulation) do not depend on what other agents do at the same time.

[1] University of Edinburgh, UK, email: {mcrosby1,mrovatso}@inf.ed.ac.uk
[2] Universitat Pompeu Fabra, Spain, email: anders.jonsson@upf.edu

In this paper, we propose a new planning method that transforms multiagent planning problems into single-agent problems and builds on the above observations. The transformation forces agents to select joint actions associated with a single subset of objects at a time, and ensures that the concurrency constraints on this subset are satisfied. Such constraints relate to whether an action cannot be, or must be, performed by certain numbers of agents concurrently. Joint actions are further serialised such that each agent performs their part of the action on their own. The result is a single-agent planning problem in which agents perform actions individually, one at a time. The main benefit of our approach is that the number of actions is vastly reduced, leading to a significant speed-up in planning time.

Once the single-agent problem has been solved, we re-introduce concurrency and minimise overall plan length using a cost-optimal compression algorithm. The resulting plan is not necessarily optimal with respect to the original multiagent problem but can be computed much faster. In addition, the plan merging process is provably tractable given certain restrictions on the compressed plan.

We emphasise that our work is exclusively concerned with the offline, centralised computation of a plan that involves multiple agents acting concurrently and cooperatively. This means that we disregard many aspects that are important for multiagent planning, and focus only on the purely algorithmic issue of finding an action sequence that will lead to a joint, known goal in environments where large numbers of possible actions can be taken by the agents at every step. However, even in settings in which we are more interested in coordination among independent agents and multiple goals, the ability to compute joint plans is an important one. For example, it can be used to detect conflicts among individual agent goals, or for any agent to propose a joint plan that can then be negotiated with others.

The remainder of this paper is structured as follows. In Section 2 we introduce the type of multiagent planning problem that we consider in this paper. Section 3 describes the transformation of multiagent planning problems to single-agent planning problems, as well as the compression algorithm for translating a solution to the single-agent problem back to a concurrent multiagent plan. In Section 4 we present the results of an empirical evaluation of our algorithm in several multiagent benchmark domains. We describe related work in Section 5 and conclude with a discussion of the work in Section 6.

2 MULTIAGENT PLANNING

We consider the problem of centralised multiagent planning in which agents share a common goal. Agents may have different action capabilities and can perform actions in parallel, forming joint actions. The problem is to find a plan, i.e. a sequence of joint actions that brings the system from the initial state to the goal. Concurrency constraints disallow some joint actions, either because two or more individual actions cannot be performed in parallel or, on the contrary, because

they have to be performed in parallel. Each joint action has an associated cost, and lower-cost solutions are preferred. In this work, we do not attempt to output cost-optimal plans, but the capability of dealing with costs is included to make the approach as general as possible.

We take a PDDL view of planning in which actions are instantiated from action templates, each with an associated set of parameters. An example action template from the LOGISTICS domain is given by load-truck(?p, ?t, ?l) where ?p is an object representing a package, ?t is an object representing a truck, and ?l is an object representing a location. We assume that the agent itself always appears as a parameter, and in general, nothing prevents the parameters from including other agents as well. In the example action template, the agent is the truck. If agents have different action capabilities, we can define a static predicate capable-a(?a) for each action template a, add this predicate to the precondition of a, and use the initial state of a planning problem to indicate which agents are capable of performing actions of type a. In an example LOGISTICS problem with an object truck1 representing a truck, this would lead to the following fluent being added to the initial state: capable-load-truck(truck1).

In PDDL, each planning problem is defined by a set of objects that are assigned to the parameters of action templates to form actions. We assume that concurrency constraints can be formulated in terms of such objects. In particular, for each action template, we use a subset of its parameters to define the associated concurrency constraints. The intuition is that objects later associated to these parameters are objects common to all agents that limit their interaction. In the example action template, the concurrency constraints are defined in terms of the parameter ?p, i.e. the object representing a package. For ease of presentation, we assume that planning domains are untyped, but it is straightforward to extend our approach to typed domains.

We proceed to define multiagent planning problems of the type we consider. Let F be a set of ground fluents. A *literal* l is a positive or negative fluent from F, i.e. $l = f$ or $l = \neg f$ for some $f \in F$. For a set of literals L, let $L^+ = \{f | f \in L\}$ denote the set of fluents that appear as positive in L, and $L^- = \{f | \neg f \in L\}$ the set of fluents that appear as negative. L is well-defined if there exists no fluent $f \in F$ such that $f \in L$ and $\neg f \in L$, i.e. if $L^+ \cap L^- = \emptyset$.

A state $s \subseteq F$ is a subset of fluents that are true in that state, while all other fluents are assumed to be false. An action $a = \langle \mathrm{pre}(a), \mathrm{eff}(a) \rangle$ is composed of well-defined sets of literals $\mathrm{pre}(a)$ and $\mathrm{eff}(a)$, where $\mathrm{pre}(a)^+$ is the set of positive pre-conditions, $\mathrm{pre}(a)^-$ the set of negative pre-conditions, $\mathrm{eff}(a)^+$ the set of add effects, and $\mathrm{eff}(a)^-$ the set of delete effects. The precondition $\mathrm{pre}(a)$ of a holds in a state s if $\mathrm{pre}(a)^+ \subseteq s$ and $\mathrm{pre}(a)^- \cap s = \emptyset$, and executing a in s results in a new state

$$\theta(s, a) = \left(s \setminus \mathrm{eff}(a)^-\right) \cup \mathrm{eff}(a)^+.$$

We define a multiagent planning problem (MAP) as a tuple $\Pi = \langle N, O, F, I, G, \{A_i\}_{i=1}^n, \phi, c \rangle$, where

- $N = \{1, \ldots, n\}$ is the set of agents,
- O is the set of objects defining the problem such that $N \subseteq O$,
- F is the set of fluents,
- $I \subseteq F$ is the initial state,
- $G \subseteq F$ is the goal state,
- A_i is agent i's action set, and each action $a_i \in A_i$ has an associated set of objects $O(a_i) \subseteq O$,
- $\phi : \mathcal{O} \to \mathbb{N} \times \mathbb{N}$ is a set of concurrency constraints on subsets of objects, where $\mathcal{O} = \{O' \subseteq O : \exists i \in N \text{ and } a_i \in A_i \text{ s.t. } O(a_i) = O'\}$ is the set of subsets of objects associated with actions,
- $c : A_1 \times \cdots \times A_n \to \mathbb{R}$ is a cost function.

We write $A = A_1 \times \ldots \times A_n$ for the joint action set assuming a concurrent, synchronous execution model. We assume that the action set A_i of each agent i includes a no-op action $noop_i$ with empty precondition and effect. Although we consider the goal as being shared, each agent may have its own, personal goal; in this case the aim is to satisfy the goal of all agents at once.

We define the preconditions and effects of a joint action $a = (a_1, \ldots, a_n) \in A$ as the union of the preconditions and effects of its constituent single-agent actions a_i, i.e. $\mathrm{pre}(a) = \cup_i \mathrm{pre}(a_i)$ and $\mathrm{eff}(a) = \cup_i \mathrm{eff}(a_i)$. Note that the resulting precondition and effect may be ill-defined. Given a joint action $a = (a_1, \ldots, a_n)$ and a subset of objects $O' \in \mathcal{O}$, let $\sigma(a, O')$ be the number of individual actions associated with O', i.e. $\sigma(a, O') = |\{i \in N : O(a_i) = O'\}|$.

As an example, consider a LOGISTICS problem with three agents: two trucks truck1 and truck2 and an airplane plane3, i.e. the set of agents is $N = \{1, 2, 3\}$. An example joint action is given by $a = (\text{load-truck(p1, truck1, l1)}, \text{load-truck(p1, truck2, l1)}, noop_3)$. The set of objects associated with the first two actions is $\{\text{p1}\}$, i.e. the package to be loaded, implying that $\sigma(a, \{\text{p1}\}) = 2$ since there are two actions associated with this subset.

Each subset of objects $O' \in \mathcal{O}$ has an associated concurrency constraint $\phi(O') = (l(O'), u(O'))$, where $l(O')$ and $u(O')$ are lower and upper bounds (satisfying $1 \leq l(O') \leq u(O') \leq n$) on the number of actions associated with O'. Note that $|\mathcal{O}|$ is bounded by the number of actions, ensuring that the number of concurrency constraints is tractable. A joint action $a \in A$ satisfies the concurrency constraints iff for each subset of objects $O' \in \mathcal{O}$, $\sigma(a, O') = 0$ or $l(O') \leq \sigma(a, O') \leq u(O')$. Action a is applicable in a state s if and only if $\mathrm{pre}(a)$ and $\mathrm{eff}(a)$ are well-defined, a satisfies the concurrency constraints, and $\mathrm{pre}(a)$ holds in s. We assume that as long as the concurrency constraints hold, we can always combine actions with the same associated subset of objects, i.e. that a corresponding joint action has well-defined precondition and effect.

If an agent i is free to take an action $a_i \in A_i$ on their own, we associate a_i with the agent itself, i.e. $O(a_i) = \{i\}$, and define a concurrency constraint $\phi(\{i\}) = (1, 1)$, i.e. at most one individual action may be associated with $\{i\}$. In LOGISTICS, action templates drive-truck and fly-airplane are of this type, i.e. no object common to agents are affected by the associated actions. In contrast, no-op actions have no associated objects, i.e. $O(noop_i) = \emptyset$ for each i.

The cost function c is defined over the joint action set A. We allow arbitrary cost functions such that the cost of a joint action may be unrelated to the costs of its constituent actions. Although the cost function may require exponential space to represent in the worst case, we can often represent it more compactly. For example, in the IPC domains we consider, the cost of a joint action a simply equals the number of individual actions in a different from the no-op action.

A *plan* $\pi = \langle a^1, \ldots, a^k \rangle$ is a sequence of joint actions $a^j \in A$ such that a^1 is applicable in the initial state I, and a^j is applicable in the state $\theta(I, \langle a^1, a^2, \ldots, a^{j-1} \rangle)$ (where θ is canonically extended to sequences of actions), for all $2 \leq j \leq k$. We say that π *solves* the MAP Π if the goal state G is satisfied following the application of all actions in π, i.e. $G \subseteq \theta(I, \pi)$. The cost of a plan π is given by $C(\pi) = \sum_{j=1}^k c(a^j)$.

3 TRANSFORMATION

In this section we describe our algorithm for transforming multiagent planning problems to single-agent problems. The goal is to construct a single-agent problem that can be solved much more efficiently than the original multiagent problem, without losing the expressivity of

the multiagent problem. We also describe a cost-optimal algorithm for compressing a solution to the resulting problem.

3.1 Transformation Into Single-Agent Problem

The aim of the transformation is to start with a MAP $\Pi = \langle N, O, F, I, G, \{A_i\}_{i=1}^{n}, \phi, c\rangle$ and construct a single-agent planning problem, i.e. a tuple $\Pi' = \langle F', I', G', A'\rangle$. We then solve the single-agent problem Π' and translate the solution back to Π. In what follows we describe how to construct the components of Π'.

We describe fluents in PDDL format, i.e. each fluent is associated with a predicate. We first introduce a set of objects $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ that represent agent counts, i.e. a number of agents between 1 and n. The set of fluents F' includes all fluents in F, plus the following fluents:

- A fluent free indicating whether we are free to take any action.
- For each agent i and each subset $O' = \{\mathsf{o}_1, \ldots, \mathsf{o}_k\} \in \mathcal{O}$, a fluent $\mathsf{use}(i, \mathsf{o}_1, \ldots, \mathsf{o}_k)$ indicating that i is using the subset O'.
- For each $O' = \{\mathsf{o}_1, \ldots, \mathsf{o}_k\} \in \mathcal{O}$ and agent count ct_j, a fluent $\mathsf{count}(\mathsf{o}_1, \ldots, \mathsf{o}_k, \mathsf{ct}_j)$ indicating that there are j agents using O'.
- For each $O' = \{\mathsf{o}_1, \ldots, \mathsf{o}_k\} \in \mathcal{O}$ and agent count ct_j, a fluent $\mathsf{sat}(\mathsf{o}_1, \ldots, \mathsf{o}_k, \mathsf{ct}_j)$ indicating whether ct_j satisfies the concurrency constraint on O', i.e. whether $l(O') \leq j \leq u(O')$.
- For each pair of agent counts $(\mathsf{ct}_j, \mathsf{ct}_k)$, a fluent $\mathsf{consec}(\mathsf{ct}_j, \mathsf{ct}_k)$ indicating that ct_j and ct_k are consecutive agent counts.

The initial state I' is defined as

$$I' = I \cup \{\mathsf{free}\} \cup \{\mathsf{consec}(\mathsf{ct}_j, \mathsf{ct}_k) : 1 \leq j < n, k = j + 1\} \cup \{\mathsf{sat}(O', \mathsf{ct}_j) : O' \in \mathcal{O}, l(O') \leq j \leq u(O')\}.$$

In other words, we are initially free to take any action, consecutive counts are encoded in fluents of type consec, and concurrency constraints are encoded in fluents of type sat. The goal state is defined as $G' = G \cup \{\mathsf{free}\}$, where free ensures that joint actions have to finish before we can check whether the goal has been satisfied.

We now discuss the ways in which joint actions are handled in our single-agent planning version of the problem. We only allow joint actions that involve a single subset O' of objects, i.e. all constituent actions (apart from no-op actions) are associated with O'. We also serialise joint actions such that each agent performs their part of the action separately. To achieve this effect, for each agent i and each action $a_i \in A_i$ different from the no-op action (i.e. $a_i \neq noop_i$), we introduce four actions, listed below. The description of each action includes additional parameters, preconditions and effects (each action also includes the same preconditions and effects as a_i). For simplicity we write O' instead of $O(a_i)$.

lone-a_i()
pre += $\{\mathsf{free}, \mathsf{sat}(O', \mathsf{ct}_1)\}$
eff += $\emptyset$

start-a_i()
pre += $\{\mathsf{free}\}$
eff += $\{\neg\mathsf{free}, \mathsf{use}(i, O'), \mathsf{count}(O', \mathsf{ct}_1)\}$

do-$a_i(\mathsf{ct}, \mathsf{ct}')$
pre += $\{\neg\mathsf{use}(i, O'), \mathsf{count}(O', \mathsf{ct}), \mathsf{consec}(\mathsf{ct}, \mathsf{ct}')\}$
eff += $\{\mathsf{use}(i, O'), \neg\mathsf{count}(O', \mathsf{ct}), \mathsf{count}(O', \mathsf{ct}')\}$

end-$a_i(\mathsf{ct}, \mathsf{ct}')$
pre += $\{\neg\mathsf{use}(i, O'), \mathsf{count}(O', \mathsf{ct}), \mathsf{consec}(\mathsf{ct}, \mathsf{ct}'), \mathsf{sat}(O', \mathsf{ct}')\}$
eff += $\{\mathsf{free}, \forall i \in N : \neg\mathsf{use}(i, O'), \neg\mathsf{count}(O', \mathsf{ct})\}$

The action lone-a_i indicates that agent i can apply action a_i in mutual exclusion on the set of objects O', i.e. $l(O') \leq 1 \leq u(O')$, encoded in the fluent $\mathsf{sat}(O', \mathsf{ct}_1)$. The action start-$a_i$ is applicable if we are free to take actions. The intention of this action is to start a joint action associated with O' involving multiple agents. The effect is to delete free, effectively making other actions of type lone and start inapplicable, and updating the use of O' such that agent i uses O' and the count associated with O' is 1.

The action do-a_i is applicable whenever at least one agent is using O', encoded in the fluent $\mathsf{count}(O', \mathsf{ct})$. The precondition also requires i not to be using O', preventing i from contributing two or more individual actions. The effect is that i is using O', and the count on O' is incremented. The action end-a_i is applicable whenever do-a_i is, but also requires the next count to satisfy the concurrency constraint on O'. The intention is to end the joint action, and the effect is to add the fluent free and delete the count on O'. In our implementation we use a forall effect to delete $\mathsf{use}(i, O')$ for each agent i, but it would be relatively straightforward to encode the transformation in STRIPS by defining actions that "step down" to delete this fluent one agent at a time.

Because of the way preconditions are defined, an action of type start always has to be succeeded by a sequence of actions of type do, followed by an action of type end. No actions other than those associated with the same subset of objects are applicable until after end: actions of type lone and start because the precondition free does not hold, and actions of type do and end associated with another subset of objects O'' because the precondition $\mathsf{count}(O'', \mathsf{ct})$ does not hold for any count ct. In a solution to Π', a sequence of actions of type $\langle\mathsf{start}, \mathsf{do}, \ldots, \mathsf{do}, \mathsf{end}\rangle$ corresponds to a joint action that involves the individual actions of all agents appearing in the sequence.

Lemma 1. *The number of actions of the planning problem Π' is bounded by $2n \sum_{i \in N} |A_i|$.*

Proof. For each action a_i of agent i, the transformation contains $2n$ actions: one copy of lone-a_i and end-a_i, and $n-1$ copies of start-a_i and do-a_i (one for each pair $(\mathsf{ct}, \mathsf{ct}')$ that satisfies $\mathsf{consec}(\mathsf{ct}, \mathsf{ct}')$). There is a total of $\sum_{i \in N} |A_i|$ such individual actions of agents. Thus the transformation contains exactly $2n \sum_{i \in N} |A_i|$ actions. □

In general, the number of joint action of the original MAP Π is exponential in the number of agents. In contrast, the single-agent planning problem Π' has a small polynomial number of actions.

3.2 Example

We use the maze domain [6] to illustrate the transformation from multiagent to single-agent planning. The maze domain was designed to test more complex concurrency constraints, covering situations in which actions must/cannot be performed in parallel by several agents. It also includes resources which cannot be used by an agent once other agents have used them. A problem of the domain consists of a grid of locations connected vertically and horizontally. Each agent has to travel from an initial location to a goal location. Each connection between neighbouring locations is of one of three types:

- Door: can only be traversed by one agent at a time.
- Bridge: can be crossed by multiple agents at once, but is destroyed after the first crossing.
- Boat: can only be used by two or more agents at once, and only in the same direction.

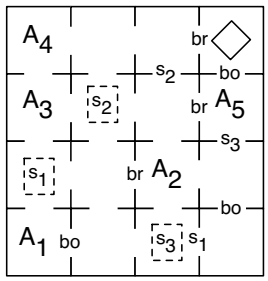

Figure 1. An example problem for the maze domain. The unannotated connections are doors, the annotations are br for bridge and bo for boat. The boxed s_i represents the switch which opens the door labelled s_i. The diamond represents the goal location for the agents. In the problem instances used in the evaluation each agent may have a separate goal location.

In addition, some doors are initially locked, and can only be unlocked by pushing an associated switch at an arbitrary location.

The domain has four action templates:

- move(?a, ?d, ?l_1, ?l_2), where ?a is an agent, ?d is a door, and ?a moves from ?l_1 to ?l_2 via ?d,
- cross(?a, ?br, ?l_1, ?l_2), where ?a is an agent, ?br is a bridge, and ?a moves from ?l_1 to ?l_2 via ?br,
- row(?a, ?bo, ?l_1, ?l_2), where ?a is an agent, ?bo is a boat, and ?a moves from ?l_1 to ?l_2 via ?bo,
- push(?a, ?s, ?l, ?d), where ?a is an agent, ?s is a switch, ?l is the location of ?s, and ?d is the door to be unlocked.

Figure 1 shows an example problem of the maze domain with five agents in a 4×4 grid. The initial agent locations are denoted A_1-A_5, and agents have the same goal location, marked with a diamond. Unannotated connections represent unlocked doors, while s_1, s_2, and s_3 denote locked doors that can be opened by pushing the switch with the same label. Bridges and boats are labelled br and bo, respectively.

We define concurrency constraints for each action template in the following way. The move template is associated with the subset of parameters {?d}, i.e. the door, the cross template with {?br}, i.e. the bridge, the row template with {?bo, ?l_1}, i.e. the boat and initial location, and the push template with {?s}, i.e. the switch.

Each door object d can only be traversed by one agent at a time, so we define $\phi(\{\mathsf{d}\}) = (1, 1)$. Each bridge br can be crossed by multiple agents at a time, so we define $\phi(\{\mathsf{br}\}) = (1, n)$, i.e. any number of agents can use it in parallel. Each combination {bo, l} of a boat and initial location can only be used by at least two agents, so we define $\phi(\{\mathsf{b}, \mathsf{l}\}) = (2, n)$. The reason we need to include the initial location in addition to the boat is that otherwise, agents could simultaneously row in opposite directions which, unlike crossing a bridge, we do not want to allow. Finally, each switch s can only be pushed by one agent at a time, so we define $\phi(\{\mathsf{s}\}) = (1, 1)$.

We illustrate our approach for transforming a multiagent problem of the maze domain into a single-agent problem. Two of the action templates, move and push, have associated subsets of objects whose concurrency constraints equal $(1, 1)$. As a consequence, a joint action only ever admits a single action of each type. In turn, this means that actions of type start-move, do-move, and end-move (and the corresponding actions for push) are not needed, since each of these are only part of a joint action involving more than one agent. Likewise, we can remove actions of type lone-row since agents can never row on their own. The single-agent formulation of the problem thus includes the following action templates:

lone-move(?a, ?d, ?l_1, ?l_2),
lone-cross(?a, ?br, ?l_1, ?l_2),
start-cross(?a, ?br, ?l_1, ?l_2),
do-cross(?a, ?br, ?l_1, ?l_2, ?ct_1, ?ct_2),
end-cross(?a, ?br, ?l_1, ?l_2, ?ct_1, ?ct_2),
start-row(?a, ?bo, ?l_1, ?l_2),
do-row(?a, ?bo, ?l_1, ?l_2, ?ct_1, ?ct_2),
end-row(?a, ?bo, ?l_1, ?l_2, ?ct_1, ?ct_2),
lone-push(?a, ?s, ?l, ?d).

3.3 Compressing Single-Agent Plans

Once we have computed a plan π' that solves the single-agent planning problem Π', we compress the plan by merging actions of π' into more complex joint actions. One way to obtain a concurrent plan is to apply a SAT-based planner [10, 14], but the resulting plans typically contain more than one simultaneous action for each agent (e.g. a truck in LOGISTICS can load multiple packages simultaneously). Moreover, these planners assume that the cost of a joint action equals the sum of the costs of the constituent single-agent actions, which is not necessarily the case in our framework.

Instead, given a sequential solution π' to Π', our approach is to compute a compressed plan π that satisfies the following restrictions:

1) π contains the same individual actions as π'.
2) π preserves the order in which actions appear in π', although consecutive actions in π' may appear together as part of a joint action.

Out of all such compressed plans, we compute the plan that minimises global cost. We remark that there are ways to compress plans that do not respect the above restrictions. In particular, we could consider joint plans in which the order of two actions of different agents is reversed, which may result in shorter joint plans overall. However, we conjecture that there exists no tractable algorithm for computing the cost-optimal joint plan in this case, since in the worst case we would have to consider all different ways in which to order actions.

The first compression step is to merge sequences of actions of type $\langle \mathsf{start}, \mathsf{do}, \ldots, \mathsf{do}, \mathsf{end} \rangle$, since such sequences in fact represent single joint actions. We also revert actions to their original action in the MAP Π, stripping them of the prefix as well as additional parameters, preconditions, and effects. Finally, we insert a no-op action for each agent that does not perform another action. Let $\pi'' = \langle a^1, \ldots, a^m \rangle$ be the resulting sequence of joint actions of the original MAP Π, and let $N(a^j) \subseteq N$ be the subset of agents performing an action different from the no-op action for each $1 \leq j \leq m$.

We then attempt to compress the plan further by merging actions in a cost-optimal way. Table 1 describes an algorithm called COMPRESS which achieves this in polynomial time using dynamic programming. The algorithm COMPRESS takes the plan $\pi'' = \langle a^1, \ldots, a^m \rangle$ described above as input and returns a compressed plan with optimal cost among the plans satisfying restrictions 1) and 2). For each step j of the plan π'', $A(j)$ denotes the optimal cost of a compressed plan involving actions a^1 through a^j, and $B(j)$ denotes the index of the action such that $a^{B(j)}, \ldots, a^j$ should be merged into a single joint action to achieve cost $A(j)$.

The dynamic programming part of the algorithm involves three variables: an index k, a set of agents S, and a joint action a. Initially, $S = \emptyset$ and each agent performs a no-op action. For every k starting from j in descending order, the algorithm checks whether some agent in $N(a^k)$ already appears in S. If so, we cannot merge actions $a^k, \ldots, a^j$ since the action sequence contains two actions associated with the same agent.

```
1  algorithm COMPRESS(π'')
2  A(0) ← 0
3  for each j = 1, ..., m
4       A(j) ← ∞
5       B(j) ← 0
6       k ← j
7       S ← ∅
8       a ← (noop_1, ..., noop_n)
9       while k > 0 and N(a^k) ∩ S = ∅ do
10           S ← S ∪ N(a^k)
11           a ← a ⊕ a^k
12           if a is well-defined and A(k − 1) + c(a) ≤ A(j) then
13                A(j) ← A(k − 1) + c(a)
14                B(j) ← k
15           k ← k − 1
16 π ← ⟨⟩
17 j ← m
18 while j > 0
19      a ← (noop_1, ..., noop_n)
20      for each k = B(j), ..., j
21           a ← a ⊕ a^k
22      append a to the beginning of π
23      j ← B(j) − 1
24 return π
```

Table 1. Algorithm COMPRESS(π').

If merging is possible, we include all agents in $N(a^k)$ in S and update a. The operation $a \oplus a^k$ returns a joint action such that agents in $N(a^k)$ perform their action in a^k and other agents perform their action in a. The resulting joint action a is well-defined if its precondition and effect are well-defined and it satisfies the concurrency constraint on each subset of objects in $\mathcal{O}$. If a is well-defined, we retrieve the cost $A(k-1)$ of the optimal compressed plan for actions a^1 through a^{k-1} and add the cost $c(a)$ of a. If this cost is the lowest found for j so far, we update $A(j)$ and $B(j)$. Ties are broken in a way that favors merging as many actions as possible.

Constructing the compressed plan π amounts to using the indices $B(j)$ to merge actions. The complexity of COMPRESS is $O(N \cdot m)$, where the N derives from the maximum number of iterations of the inner loop (we can add at most N distinct agents to S without repetition). The operation $a \leftarrow a \oplus a^k$ can be computed in time $O(|N(a^k)|)$ by changing only the actions of agents in $N(a^k)$.

4 EVALUATION

In this section we present results from an empirical evaluation of our algorithm in several multiagent planning domains. We first discuss how our algorithm could be applied to multiagent planning domains taken from the International Planning Competition, or IPC for short. We then present results of applying our transformation to multiagent planning problems from the maze domain.

Several single-agent benchmark domains from the IPC are typically used as multiagent benchmarks, since some objects of planning problems are naturally thought of as agents. Examples of domains from the IPC used to evaluate multiagent planners in the literature include DEPOTS, LOGISTICS, ROVERS and SATELLITE.

In the multiagent formulation of LOGISTICS, agents are trucks and airplanes, and the objects over which concurrency constraints are defined are packages. The concurrency constraint defined on each package p is $\phi(\{\mathsf{p}\}) = (1, 1)$, i.e. only a single agent can manipulate p at a time. As previously mentioned, the concurrency constraint on each agent i is also defined as $\phi(\{i\}) = (1, 1)$, i.e. actions exclusive to each agent can only be applied by the agent itself. Following our discussion from the previous section, we can eliminate all actions of type start, do, and end. Since only actions of type lone remain, fluent free is no longer needed (no action adds or deletes it). Thus in LOGISTICS, our transformation of a multiagent problem into a single-agent problem results in the standard single-agent formulation.

The case is similar for the other IPC domains. DEPOTS is essentially a combination of the BLOCKSWORLD and LOGISTICS domains in which agents are trucks and hoists that have to move and stack crates at different locations. The common objects are crates, which can only be manipulated by one agent at a time. In ROVERS, agents are Mars rovers that have to collect samples, take images, and communicate findings to a lander. The only common object is the lander, and rovers can only communicate information to the lander one at a time. In SATELLITE, agents are satellites charged with taking images in different directions. Each action is exclusive to each agent, implying that there are no actions that require concurrency constraints.

In each of the three domains, all concurrency constraints equal $(1, 1)$, i.e. only a single agent can manipulate each object set at a time. Our transformation is thus equal to the standard single-agent formulation. In the IPC domains, the cost of a joint action a is defined as the number of individual actions in a different from the no-op action. In this case, compressing a single-agent plan into a shorter joint plan does not change the cost of the plan, although the makespan of the plan (i.e. the length of the joint plan) may be shorter.

In experiments, we used the LAMA planner [13] to solve the single-agent problems of the IPC domains, and ran our compression algorithm to produce a joint plan. On average, the compressed plans were 21% shorter than the single-agent plans. Although these results are not groundbreaking from a multiagent perspective, they still serve as a lesson: when actions associated with the same common objects cannot be applied concurrently, a simple way to reduce the complexity is to force actions to be applied individually, drastically reducing the number of joint actions of the multiagent planning problem.

We also ran experiments with our approach in the maze domain described in the previous section. We randomly generated 4 problems of the maze domain for 5, 10, 15, and 20 agents, and grid sizes of 4x4, 8x8, 12x12, 16x16, and 20x20. It was hard to guarantee that problems were solvable, and we found that the best combination of connections was to define 70% of connections as doors, 10% as bridges, 10% as boats, and 10% as locked doors with associated switches. Still, some problems remained unsolvable, or at the very least, LAMA was unable to solve them. Unlike the example problem in Figure 1, each agent had its own random destination.

Table 2 shows results of experiments in the maze domain. As before, we defined the cost of a joint action as the number of individual actions different from the no-op action. The table shows, for each combination of agents and grid sizes, the average length of the single-agent plan (L') and of the joint plan after merging and compression (L). The average was only taken over problems solved; in two cases, none of the four problems were solved by LAMA. The solutions frequently included cross and row actions in addition to move actions, but push actions were rare, presumably because agents could not reach switches or because other action sequences were shorter.

The results in the maze domain highlight that our approach makes it possible to solve large multiagent planning problems with tens of agents and relatively complex interaction between agents (in the form of cross and row actions that can or have to be taken concurrently). We are aware of no other multiagent planning approach that could handle problems of this type and dimension.

	$n=5$		$n=10$		$n=15$		$n=20$	
	L'	L	L'	L	L'	L	L'	L
4×4	8	5	28	16	39	24	43	21
8×8	29	24	52	41	56	39	88	60
12×12	38	31	43	31	–	–	97	49
16×16	59	50	56	38	153	133	127	98
20×20	55	47	–	–	81	63	132	104

Table 2. Results in the maze domain; see the text for explanations.

5 Related work

Methods for cooperative, centralised multiagent planning mostly focus on techniques for reducing the combinatorial explosion that results from agents acting concurrently in the same environment. Early on, Lansky [11] suggested a decomposition of a STRIPS-style planning domain into several potentially overlapping local search spaces ("regions"), and applied a hierarchical constraint satisfaction technique to combine "region plans" in combination with backtracking. Brafman and Domshlak [2] investigate loosely coupled planning domains by looking at "coordination points" in a MAP based on the distinction between public and private fluents. Nissim *et al*'s [12] distributed multiagent planning algorithm exploits exactly this type of loose coupling, solving a distributed CSP for those parts of the global plan where individual agents' contributions overlap, while allowing the agents to plan independently for other parts of the planning problem. The empirical evaluations of all of these approaches show that the degree of coupling present in the domains determines the planning complexity in practice. While the approach of Jensen and Veloso [9] uses OBDDs for concurrent multiagent non-deterministic planning, the number of joint actions could still be exponential in the number of agents in the worst case.

Boutilier and Brafman [1] address the lack of a proper treatment of concurrent action in most STRIPS-based formalisms, and propose a sound and complete partial-order planner that can deal with concurrency constraints (where individual agents' actions have to be or cannot be executed in parallel). The focus of their paper is on appropriate representations for multiagent planning with concurrent actions, and it does not address issues of scalability or empirical performance in real-world domains.

In [5], Crosby also defines concurrent action constraints on the objects in a domain and it is shown how a translation can be made to temporal planning so that these problems can be solved using a temporal planner. However, Crosby does not include an equivalent of the function $\phi : \mathcal{O} \rightarrow \mathbb{N} \times \mathbb{N}$, meaning that it is not possible to associate different concurrency constraints together based on specifically chosen objects that can appear in a ground action. Instead, concurrency constraints are always linked to all actions that contain a particular constrained object in their preconditions or effects. This means that there are certain constraints that are expressible in our approach that would not be in his. However, the domains used in this paper would all be expressible using the methods found in [5].

6 CONCLUSION

In this paper, we presented a new planning method for synthesising multiagent plans with concurrent actions and concurrency constraints among those actions. Our method is based on transforming the original multiagent planning problem to a single-agent problem, while specifying concurrency constraints in terms of sets of objects associated with actions. As the number of possible interactions among individual agents' actions is manageable in many real-world domains if we adopt this perspective, our experiments show that the potentially exponential blow-up in the number of total joint actions that have to be considered can often be avoided in practice.

The second contribution is a plan compression algorithm that is based on dynamic programming and which minimises total action cost when iteratively merging sequential actions to exploit concurrency in the joint multiagent plan. This could be particularly important in domains with more complex cost functions, but also results in plans that are shorter than those returned by standard single-agent planners in conventional benchmark domains with unit action costs.

In future work, we would like to extend our work to domains with other types of concurrency constraints that cannot be captured by objects, but still permit transformations to single-agent planning. For example, a domain that includes actions for both painting and passing-through a doorway may only permit one agent to pass through at a time while multiple agents can paint the door concurrently. This suggests a mapping of concurrency constraints onto the affordances of objects, though details will be left for another paper.

REFERENCES

[1] C. Boutilier and R. Brafman, 'Partial-order planning with concurrent interacting actions', *Journal of Artificial Intelligence Research*, **14**, 105–136, (2001).

[2] R. Brafman and C. Domshlak, 'From One to Many: Planning for Loosely Coupled Multi-Agent Systems', in *ICAPS*, volume 18, pp. 28–35, (2008).

[3] R. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz, 'Planning Games', in *IJCAI*, volume 21, pp. 73–78, (2009).

[4] Jeffrey S. Cox and Edmund H. Durfee, 'Efficient and distributable methods for solving the multiagent plan coordination problem', *Multiagent and Grid Systems*, **5**(4), 373–408, (2009).

[5] Matthew Crosby, 'A temporal approach to multiagent planning with concurrent actions', *PlanSIG*, (2013).

[6] Matthew Crosby, *Heuristic Multiagent Planning*, Ph.D. dissertation, University of Edinburgh, UK, 2014.

[7] Mathijs M. de Weerdt and Brad J. Clement, 'Introduction to planning in multiagent systems', *Multiagent and Grid Systems*, **5**(4), 345–355, (2009).

[8] E. Durfee, 'Distributed Problem Solving and Planning', in *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*, 121–164, (1999).

[9] R. Jensen and M. Veloso, 'OBDD-based Universal Planning for Synchronized Agents in Non-Deterministic Domains', *Journal of Artificial Intelligence Research*, **13**, 189–226, (2000).

[10] H. Kautz, B. Selman, and J. Hoffmann, 'SatPlan: Planning as Satisfiability', in *Abstracts of the 5th International Planning Competition*, (2006).

[11] A. Lansky, 'Localized search for multiagent planning', in *Proceedings of the Twelfth International Joint Conference on Artificial intelligence (IJCAI 1991)*, pp. 252–258, (1991).

[12] R. Nissim, R. Brafman, and C. Domshlak, 'A general, fully distributed multi-agent planning algorithm', in *AAMAS*, volume 9, pp. 1323–1330, (2010).

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

[14] J. Rintanen, 'Planning with specialized SAT solvers', in *AAAI*, pp. 1563–1566, (2011).

[15] S. Seuken and S. Zilberstein, 'Formal Models and Algorithms for Decentralized Decision Making Under Uncertainty', *Journal of Autonomous Agents and Multi-Agent Systems*, **17**(2), 190–250, (2008).

[16] H. Tonino, A. Bos, M. de Weerdt, and C. Witteveen, 'Plan coordination by revision in collective agent based systems', *Artificial Intelligence*, **142**(2), 121–145, (2002).