A Formal Analysis of Required Cooperation in Multi-agent Planning

Yu Zhang, Sarath Sreedharan and Subbarao Kambhampati
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
{yzhan442, ssreedh3, rao@asu.edu}

Abstract

It is well understood that, through cooperation, multiple agents can achieve tasks that are unachievable by a single agent. However, there are no formal characterizations of situations where cooperation is *required* to achieve a goal, thus warranting the application of multiple agents. In this paper, we provide such a formal characterization for multi-agent planning problems with sequential action execution. We first show that determining whether there is required cooperation (RC) is in general intractable even in this limited setting. As a result, we start our analysis with a subset of more restrictive problems where agents are homogeneous. For such problems, we identify two conditions that can cause RC. We establish that when none of these conditions hold, the problem is single-agent solvable; otherwise, we provide upper bounds on the minimum number of agents required. For the remaining problems with heterogeneous agents, we further divide them into two subsets. For one of the subsets, we propose the concept of *transformer agent* to reduce the number of agents to be considered which is used to improve planning performance. We implemented a planner using our theoretical results and compared it with one of the best IPC CoDMAP planners in the centralized track. Results show that our planner provides significantly improved performance on IPC CoDMAP domains.

Introduction

Despite the increased interest in multi-agent planning, one question has remained largely unaddressed: "*under what conditions are multiple agents actually needed to solve a planning problem?*" This question is of fundamental importance as it clearly demarcates two distinct uses of multiple agents in a given planning problem: (i) the situations where multiple agents are used because there is no single agent plan for the problem, and (ii) those situations where multiple agents are used to improve execution efficiency, even though a single agent plan is in fact feasible. This latter class of problems can arguably be viewed as an easier form of multi-agent planning problems, in as much as they can be solved by first generating a single-agent plan, and then using a post-processing step to optimize the execution cost by deploying multiple agents. Without keeping such demarcation in mind when designing benchmark domains, it can be misleading to compare and evaluate multi-agent planners – in the extreme case, very fast planners can be designed to solve only problems that do not require cooperation. Unfortunately, we shall see that most of the domains used in CoDMAP, a proto multi-agent planning competition at ICAPS 2015, need multiple agents only for plan efficiency rather than feasibility.

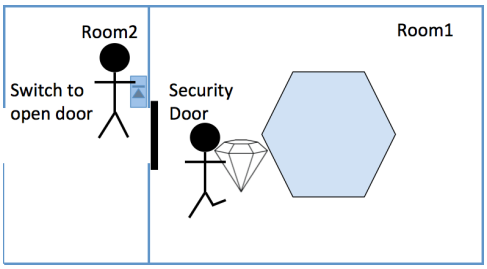

Figure 1: **Burglary problem** – The goal of this problem is to steal a diamond ($diamond1$) from $room1$, in which the diamond is secured, and place it in $room2$. The diamond is protected by a security system. If the diamond is taken, the system locks the door ($door1$) of $room1$, so that the insiders cannot exit. There is a switch ($switch1$) to manually open $door1$ but it is located in $room2$. The domain and problem are presented in Fig. 2. A solution is illustrated in the figure.

The aim of this paper is precisely to shed light on this central question of when multiple agents need to be involved in the plan to solve a problem.[1] In particular, we hope to address the following: $Q1$) Given a multi-agent planning (MAP) problem to solve, what are the conditions that make cooperation between multiple agents *required*; $Q2$) How do these conditions affect planning for MAP problems; $Q3$) Can we determine the minimum number of agents required for a MAP problem. The answer to the first question separates multi-agent planning problems from single-agent planning (SAP) problems in a fundamental way. The answer to the second question can inform the design of future multi-agent planning competitions (e.g., CoDMAP in ICAPS 2015) so that MAP problems that require cooperation can be deliberately introduced to obtain a clearer under-

[1] Our focus in this paper is on the number of agents involved in the plan, rather than the planning process by which the plan is made; see related work for details.

Initial State:

$location(agent1)\ room1$
$location(agent2)\ room1$
$location(diamond1)\ room1$
$doorLocked(room1)\ false$
$location(switch1)\ room2$

Goal State:

$location(diamond1)\ room2$

Operators:

$WalkThrough(agent, door, fromRoom, toRoom)$:

prv: $doorLocked(door)\ false$
pre: $location(agent)\ fromRoom$
$post$: $location(agent)\ toRoom$

$Steal(agent, diamond, room, door)$:

prv: $location(agent)\ room$
pre: $location(diamond)\ room$
$post$: $doorLocked(door)\ true$
$post$: $location(diamond)\ agent$

$Switch(agent, switch, room, door)$:

prv: $location(switch)\ room$
prv: $location(agent)\ room$
$post$: $doorLocked(door)\ false$

$Place(agent, diamond, room)$:

prv: $location(agent)\ room$
pre: $location(diamond)\ agent$
$post$: $location(diamond)\ room$

Figure 2: The problem and domain descriptions of the Burglary problem in Fig. 1 using SAS$^+$ in which the value is immediately specified after each variable.

standing of the capabilities of the competing planners. The answer to the third question has real-world applications, e.g., determining the minimum number of agents to be assigned.

However, as we shall see shortly, determining whether a MAP problem requires cooperation is in general no easier than finding a plan for the problem itself, which is PSPACE-complete. Hence, instead of providing exact answers to questions $Q1 - Q3$ above, we provide answers in restrictive settings (i.e., for subsets of MAP problems). First of all, given that the most obvious reason for a MAP problem to require cooperation is when capabilities of different agents are needed, we divide MAP problems into two classes: the class of problems where agents have the same capabilities (i.e., homogeneous agents) and the class where agents have different capabilities (i.e., heterogeneous agents).

For the first and more restrictive class with homogeneous agents, it may appear that cooperation is only required when joint actions are present (which are actions that must be executed by multiple agents at the same time). This intuition, however, is falsified by a simple problem with only sequential actions as shown in Fig. 1, which is referred to as the **Burglary problem** in the later part of this paper. We show that, in this class of problems, RC can be caused by the "traversability" or the "causal loops" in the causal graph of the agent state variables. Our main theorem shows that these two causes are exhaustive when agents are homogeneous: if the causal graph of agents is traversable and contains no causal loops, a single agent alone is sufficient for the problem ($Q1$). However, none of them individually represent a sufficient or necessary condition for RC. When these two causes are present in a problem, we show that upper bounds of the number of agents required can be provided ($Q3$).

For the second class where agents can have different capabilities, the analysis becomes more complex. Hence, we further divide the problems in the second class into two subclasses: the subclass of problems where the causes of RC in the first class do not appear, and the subclass of the remaining problems. For the first subclass, one observation is that the number of agents that are required can often be significantly reduced if agents are simply made to be "transformable" ($Q3$). Although this does not necessarily lead to the reduction of agents in the final plan for these problems, we show that the planning performance can be significantly improved ($Q2$). The second subclass corresponds to the most difficult setting and an analysis needs to be provided in future work. Finally, as a practical contribution of our investigation we develop a planner called RCPLAN, based on the idea of transformable agents above to efficiently solve MAP problems. We show that RCPLAN outperforms one of the best performers in the IPC CoDMAP competition.

Related Work

The term "multi-agent planning" has traditionally been quite loosely defined as "planning in the presence of multiple agents" (c.f. (Jonsson and Rovatsos 2011)). This definition blurs multi-agent plans and distributed planning by confounding two distinct types of agents: agents that are involved in the planning process ("planning agents") and agents involved in plan execution ("execution agents").

A multi-agent plan is one that involves multiple execution agents; whereas distributed planning involves using multiple *planning agents*, which, normally, also happen to be the execution agents. However, these two types of agents have orthogonal properties: it is possible to have multiple planning agents working together make a plan involving a single execution agent, just as it is possible to have a single planning agent make a plan involving multiple execution agents (aka "centralized multi-agent planning").

When we refer to "multi-agent planning" problems in this paper, our focus is on the plans of these problems for execution agents, regardless of how many planning agents were involved in the planning process. Such plans may be the result of centralized (Kvarnstrom 2011; Muise, Lipovetzky, and Ramirez 2015) or distributed planning (Durfee and Lesser 1991; Decker and Lesser 1992; Nissim and Brafman 2012; Torreno, Onaindia, and Sapena 2012).

Our analysis of required cooperation in multi-agent plans is similar in spirit to Cushing et. al.'s analysis of temporal planning (Cushing et al. 2007a; 2007b). Just as concurrency is sometimes seen as a way to improve the running time of the plan, execution agents are someimes viewed as "resources" that can be added to the plan to improve its efficiency (Pape 1990). While Cushing et. al. focus on characterizing conditions where concurrency is required to solve a planning problem, we focus on conditions where coopera-

tion between multiple execution agents is required to solve a planning problem.

Our analysis of required cooperation uses SAS$^+$ formalism (Backstrom and Nebel 1996) with *causal graphs* (Knoblock 1994; Helmert 2006), which are often discussed in the context of factored planning (Bacchus and Yang 1993; Amir and Engelhardt 2003; Brafman 2006; Brafman and Domshlak 2013). A causal graph captures the interactions between state variables; intuitively, it can also capture the interactions between agents since they affect each other through these variables (Brafman and Domshlak 2013).

Multi-agent Planning (MAP) Problem

In this paper, we focus on required cooperation (RC) in scenarios with instantaneous actions and sequential execution. The possibility of RC can only increase when we extend the model to the temporal domains in which concurrent or synchronous actions must be considered. We develop our analysis of RC based on SAS$^+$ (Backstrom and Nebel 1996).

Definition 1. *A SAS$^+$ problem is given by a tuple $P = \langle V, A, I, G\rangle$, where:*

- *$V = \{v\}$ is a set of state variables. Each variable v is associated with its domain $D(v)$.*
- *$A = \{a\}$ is a set of actions (i.e., ground operators). Each action a is a tuple $\langle pre(a), post(a), prv(a)\rangle$, in which $prv(a)$ denotes prevail conditions which are preconditions that persist through a.*
- *I and G denote initial and goal state, respectively.*

A plan in SAS$^+$ is often defined to be a total-order plan, which is a sequence of actions. For details of SAS$^+$, see (Backstrom and Nebel 1996). To extend the SAS$^+$ formalism to MAP, we minimally modify the definitions.

Definition 2. *A SAS$^+$ MAP problem is given by a tuple $P = \langle V, \Phi, I, G\rangle$ ($|\Phi| > 1$), where $\Phi = \{\phi\}$ is the set of agents; each agent ϕ is associated with a set of actions $A(\phi)$.*

Definition 3. *A plan π_{MAP} in MAP is a sequence of agent-action pairs $\pi_{MAP} = \langle(a_1, \phi(a_1)), ..., (a_L, \phi(a_L))\rangle$, in which $\phi(a_i)$ represents the agent executing the action a_i and L is the length of the plan.*

We assume that the reference of the executing agent is encoded and appears as the first argument in an action, similar to the operators (ungrounded actions) in Fig. 2.

Required Cooperation for MAP Problems

Next, we formally define the notion of *required cooperation* and a few other terms used. We assume throughout the paper that more than one agent is considered (i.e., $|\Phi| > 1$).

Definition 4 (k-agent Solvable)**.** *Given a MAP problem $P = \langle V, \Phi, I, G\rangle$ ($|\Phi| \geq k$), the problem is k-agent solvable if $\exists \Phi_k \subseteq \Phi$ ($|\Phi_k| = k$), such that $\langle V, \Phi_k, I, G\rangle$ is solvable.*

Definition 5 (Required Cooperation (RC))**.** *Given a MAP problem $P = \langle V, \Phi, I, G\rangle$, there is required cooperation if P is solvable but not 1-agent solvable.*

In other words, given a MAP problem that satisfies RC, any plan must involve more than one agent. Note also that to satisfy RC, a MAP problem must first be solvable.

Lemma 1. *Given a solvable MAP problem $P = \langle V, \Phi, I, G\rangle$, determining whether it satisfies RC is PSPACE-complete.*

Proof. First, it is not difficult to show that the RC decision problem belongs to PSPACE, since we only need to verify that $P = \langle V, \phi, I, G\rangle$ is unsolvable for all $\phi \in \Phi$, given that the initial problem is known to be solvable. Then, we complete the proof by reducing from the PLANSAT problem, which is PSPACE-complete in general (Bylander 1991). Given a PLANSAT problem (with a single agent), the idea is that we can introduce a second agent with only one action. This action directly achieves the goal but requires an action of the initial agent (with all preconditions satisfied in the initial state) to provide a precondition that is not initially satisfied. We know that this constructed MAP problem is solvable. If the algorithm for the RC decision problem returns that cooperation is required for this MAP problem, we know that the original PLANSAT problem is unsolvable; otherwise, it is solvable. □

Definition 6 (Minimally k-agent Solvable)**.** *Given a MAP problem $P = \langle V, \Phi, I, G\rangle$ ($|\Phi| \geq k$), P is minimally k-agent solvable if it is k-agent solvable, and not $(k-1)$-agent solvable.*

Corollary 1. *Given a solvable MAP problem $P = \langle V, \Phi, I, G\rangle$, determining the minimally solvable k ($k \leq |\Phi|$) is PSPACE-complete.*

Hence, directly querying for RC is intractable even when the problem is known to be solvable. Instead, we aim to identify conditions that can potentially cause RC. First, note that although actions (or ground operators) are unique for each agent, they may be identical except for the executing agent.

Definition 7 (Action Signature (AS))**.** *An action signature is an action with the reference of the executing agent replaced by a global AG_{EX} symbol.*

For example, an action signature in the IPC logistics domain is $drive(AG_{EX}, pgh\text{-}poT, pgh\text{-}airport)$. As a result, different agents can share the same action signatures. We denote the set of action signatures for any $\phi \in \Phi$ as $AS(\phi)$.

Definition 8 (Agent Variable (Agent Fluent))**.** *A variable (fluent) is an agent variable (fluent) if it is associated with the reference of an agent.*

Agent variables are used to specify agent state. For example, $location(truck\text{-}pgh)$ is an agent variable since it is associated with an agent $truck\text{-}pgh$. We use $V_\phi \subseteq V$ to denote the set of agent variables that are associated with an agent ϕ (i.e., variables that are present in the initial state or actions of ϕ), and V_o to denote the set of non-agent variables. Furthermore, we assume that agents can only interact with each other through non-agent variables (i.e., V_o).[2] This assumption implies that agent variables are associated with one and

[2]It is possible to compile away exceptions by breaking an agent variable (with more than one reference of agent) into multiple agent variables and introducing non-agent variables to correlate them. Given that this compilation only increases the problem size linearly (in the number of agents) for each such agent variable, it does not influence our later discussions.

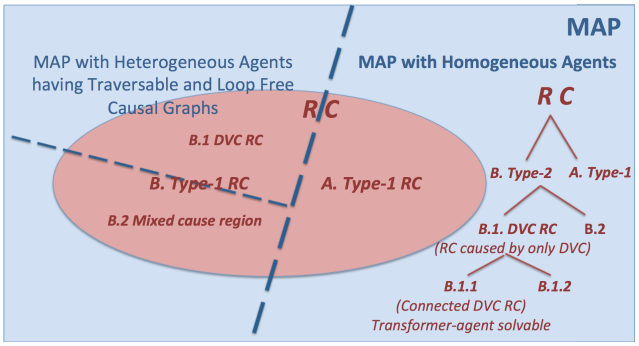

Figure 3: Division of MAP problems into MAP with heterogeneous and homogeneous agents. Consequently, RC problems are also divided into two classes: Type-1 RC involves problems with homogeneous agents (A) and Type-2 RC involves problems with heterogeneous agents (B). Type-1 RC is only caused when the causal graph is non-traversable or contains loops. Type-2 RC problems are further divided into DVC RC problems ($B.1$) where RC is caused only by the heterogeneity of agents, and RC problems with mixed causes ($B.2$). $B.1.1$ and $B.1.2$ represent DVC RC problems with and without connected state spaces, respectively.

only one reference of an agent. Thus, we have $V_\phi \cap V_{\phi'} \equiv \emptyset$ ($\phi \neq \phi'$).

Definition 9 (Variable (Fluent) Signature (VS))**.** *Given an agent variable (fluent), its signature is the variable (fluent) with the reference of agent replaced by AG_{EX}.*

For example, $location(truck\text{-}pgh)$ is an agent variable for $truck\text{-}pgh$ and its signature is $location(AG_{EX})$. We denote the set of variable signatures for V_ϕ as $VS(\phi)$, and use VS as an operator such that $VS(v)$ returns the signature of a variable v (it returns any non-agent variable unchanged).

Classes of Required Cooperation (RC)

In this paper, we focus on MAP problems with goals that do not involve agent variables (i.e., $G \cap V_\phi = \emptyset$) (because having agent variables in goals forces RC in a trivial way). We divide MAP problems into two classes to facilitate the analysis of RC. The division of MAP problems (the rectangle shaped region) as shown in Fig. 3 correspondingly also divides RC problems (the oval shaped region) into two classes based on the heterogeneity of the agents:

Agent Heterogeneity: Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the heterogeneity of the agents can be characterized by these conditions: 1) *Domain Heterogeneity (DH)*: $\exists v \in V_\phi$ and $D(V') \setminus D(v) \neq \emptyset$, in which $V' = \{v' | v' \in V_{\phi'} (\phi' \neq \phi)$ and $VS(v) = VS(v')\}$. 2) *Variable Heterogeneity (VH)*: $\exists \phi \in \Phi, VS(\Phi \setminus \phi) \setminus VS(\phi) \neq \emptyset$. 3) *Capability Heterogeneity (CH)*: $\exists \phi \in \Phi, AS(\Phi \setminus \phi) \setminus AS(\phi) \neq \emptyset$. $D(V)$ above denotes the joint domain of all $v \in V$.

We define heterogeneous agents as a set of agents in which DH, VH or CH is satisfied for any agent. This condition is also referred to as the heterogeneity condition. In contrast, we define homogeneous agents as a set of agents in which neither DH, VH nor CH is satisfied for any agent. This allows us to divide RC problems into:

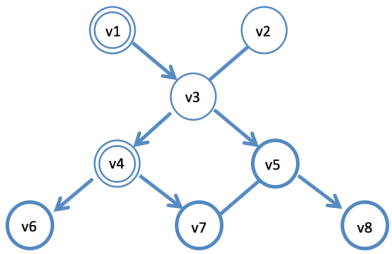

Figure 4: Example of a causal graph.

Definition 10 (Type-1 (Homogeneous) RC)**.** *An RC problem belongs to type-1 RC if the heterogeneity condition is not satisfied (i.e., agents are homogeneous).*

Definition 11 (Type-2 (Heterogeneous) RC)**.** *An RC problem belongs to type-2 RC if $\exists \phi \in \Phi$, such that DH or VH or CH is satisfied.*

It is worth noting that when considering certain entities (e.g., truck and plane in the logistics domain) as agents rather than as resources (such as in CoDMAP competition (Stolba, Komenda, and Kovacs 2015)), many problems in the IPC domains have RC and belong to type-2.

Analysis of Type-1 (Homogeneous) RC

We start with type-1 RC which represents a class of more restrictive problems and hence are easier to analyze.

Type-1 RC Caused by Traversability: One condition that can cause RC in type-1 RC problems is the traversability of the state space. One obvious example is related to non-restorable resources such as energy. For example, a robot may have all the capabilities to achieve the goal but insufficient amount of battery power. Since traversability is associated with the evolution of variables and their values, we analyze it using causal graphs.

Definition 12 (Causal Graph)**.** *Given a MAP problem $P = \langle V, \Phi, I, G \rangle$, the causal graph G is a graph with directed and undirected edges over the nodes V. For two nodes v and v' ($v \neq v'$), a directed edge $v \rightarrow v'$ is introduced if there exists an action that updates v' while having a prevail condition associated with v. An undirected edge $v - v'$ is introduced if there exists an action that updates both.*

An example of a causal graph for an agent is in Fig. 4. When we use agent variable signatures to replace agent variables in the graph, the causal graphs for all agents in type-1 problems are the same. We refer to any of these graphs as an *individual causal graph signature* (ICGS). Next, we define the notions of *closures* and *locally traversable state space*.

Definition 13 (IC and its OC)**.** *Given a causal graph, an inner closure (IC) is any set of variables for which no other variables are connected to them with undirected edges; an outer closure (OC) of an IC is the set of nodes that have directed edges going into nodes in the IC.*

In Fig. 4, $\{v_2, v_3\}$ and $\{v_4\}$ are examples of ICs. The OC of $\{v_2, v_3\}$ is $\{v_1\}$ and the OC of $\{v_4\}$ is $\{v_3\}$.

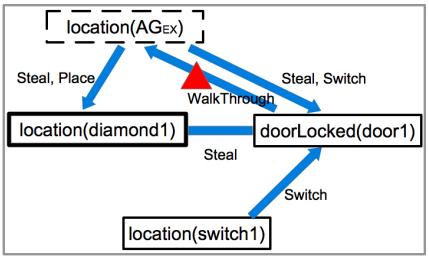

Figure 5: ICGS for the Burglary problem to illustrate causal loops that cause RC in type-1 RC problems. Actions (without arguments) are labeled along with their corresponding edges. Variables in G are shown as bold-box nodes and agent variable signatures are shown as dashed-box nodes.

Definition 14 (Locally Traversable State Space)**.** *An inner closure (IC) in a causal graph has a locally traversable state space if and only if: given any two states of this IC, denoted by s and s', there exists a plan that connects them, assuming that the state of the outer closure (OC) of this IC can be changed freely within its state space.*

In other words, an IC has a locally traversable state space if its traversal is only dependent on the variables in its OC. This also means that when the OC of an IC is empty, the state of the IC can change freely. In the case of non-restorable resources, the ICs that include variables for these resources would not satisfy this requirement (hence non-restorable). When all the ICs in the causal graph of an agent satisfy this, the causal graph is referred to as being *traversable*.

Type-1 RC Caused by Causal Loops: A problem with a traversable causal graph, however, may still require RC. Let us revisit the Burglary problem in Fig. 1. We construct the *individual causal graph signature* (ICGS) for this type-1 RC example in Fig. 5. It is not difficult to verify that this ICGS is traversable, given that $location(diamond1)$ and $doorLocked(door1)$ form an IC with the other two variables as its OC. More specifically, when assuming that the agent location can change freely, we can also update the location of the diamond as well as the status of the door to arbitrary values. One key observation is that a single agent cannot address this problem due to the fact that $WalkThrough$ with the diamond to $room2$ requires $doorLocked(door1) = false$, which is violated by the $Steal$ action to obtain the diamond in the first place. This is clearly related to the causal loop in Fig. 5:

Definition 15 (Causal Loop)**.** *A causal loop in a causal graph is a directed loop that contains at least one directed edge (undirected edges are considered as edges in both directions when checking for loops).*

When Cooperation Is Not Required

The following theorem establishes that the two causes discussed above are exhaustive – when none of them are present in a solvable MAP problem with homogeneous agents, it can be solved by a single agent.

Theorem 1. *Given a solvable MAP problem with homogeneous agents, and for which the individual causal graph signatures (ICGSs) are traversable and contain no causal loops, any single agent can also achieve the goal.*

Proof. Given no causal loops, the directed edges in the ICGS divide the variables into stratified levels, in which: 1) variables at each level do not appear in other levels; 2) higher level variables are connected to lower level variables with only directed edges going from higher levels to lower levels; 3) variables within each level are either not connected, or connected with undirected edges. For example, the variables in Fig. 4 are divided into the following levels (from high to low): $\{v_1\}$, $\{v_2, v_3\}$, $\{v_4\}$, $\{v_5, v_7\}$, $\{v_6, v_8\}$. Note that this division is not unique.

The intuition is to show that there exists a single agent plan that can lead to any goal state given an initial state. We prove the result by induction on the level. Suppose that the ICGS has k levels and the following holds: given any trajectory of states for all variables, there exists a plan whose execution traces of states include this trajectory in the correct order.

When the ICGS has $k+1$ levels: given any state s for all variables from level 1 to $k+1$, we know from the assumption that the ICGS is traversable that there exists a plan that can update the variables at the $k+1$ level from their current states to the corresponding states in s. This plan, denoted by π, requires the freedom to change the states of variables from level 1 to k. Given the induction assumption, we know that we can update these variables to their required states in the correct order to satisfy π; furthermore, these updates (at level k and above) do not influence the variables at the $k+1$ level (hence do not influence π). Once the states of the variables at the $k+1$ level are updated to match those in s, we can then update variables at level 1 to k to match their states in s accordingly. Using this process, we can incrementally build a plan whose execution traces of states contain any trajectory of states for all the variables in the correct order.

Furthermore, the induction holds when there is only one level given that ICGS is traversable. Hence, the induction conclusion holds. The main conclusion directly follows. □

Note that Theorem 1 provides an answer for the inverse of the first question (Q1) in the introduction: Theorem 1 is used to determine when cooperation is *not* required instead of when it is. More specifically, the conjunction of the conditions (traversable ICGS and no causal loop) is a sufficient condition for a MAP problem with homogeneous agents to be single-agent solvable (i.e., no RC). However, the absence of any of these conditions does not necessarily lead to RC. Theorem 1 provides an insight into the separation of SAP and MAP for problems with homogeneous agents.

Towards an Upper Bound for Type-1 RC

When the causal graph is not traversable or there are causal loops in type-1 RC problems, we find that upper bounds on the k in Def. 6 can be provided. We first investigate when causal loops are present and show that the upper bound on k is associated with how the causal loops containing agent variable signatures (agent VSs) can be broken in the individual causal graph signature (i.e., ICGS). The observation is that certain edges in these loops can be removed when

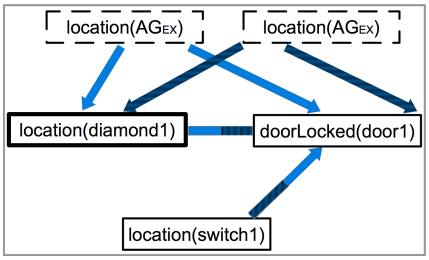

Figure 6: Causal loop breaking for the Burglary problem, in which the loop is broken by removing the edge marked with a triangle in Fig. 5. Two agent variable signatures (VSs) are introduced to replace the original agent VS.

there is no need to update the associated agent VSs. In our Burglary problem, when there are two agents in $room1$ and $room2$, respectively, there is no need to $WalkThrough$ to change locations (to access the switch after stealing the diamond). Hence, the associated edges can be removed to break the loops as shown in Fig. 6.

Lemma 2. *Given a solvable MAP problem with homogeneous agents having traversable ICGSs, if all causal loops contain agent VSs and all the edges going in and out of agent VSs are directed, the minimum number of agents required is upper bounded by* $\Pi_{v \in CR(\Phi)}|D(v)|$ *(*$|D(v)|$ *denotes the size of the domain for variable* v*) when assuming that the agents can choose their initial states.*

$CR(\Phi)$ is created by: 1) adding the set of agent VSs in the causal loops to $CR(\Phi)$; 2) adding in any agent VS to $CR(\Phi)$ if there is a directed edge going into it from any variable in $CR(\Phi)$; 3) iterating 2 until no agent VSs can be added.

Proof. For each variable in $CR(\Phi)$, denoted by v, we introduce a set of variables $N = \{v_1, v_2, ..., v_{|D(v)|}\}$ to replace v. Any edges connecting v with other variables are duplicated on all variables in N, except for the directed edges that go into v. Each variable $v_i \in N$ has a domain with a single value; this value for each variable in N is different and chosen from $D(v)$. These new variables do not affect the traversability of the ICGS and all loops are broken.

From Theorem 1, we know that a virtual agent ϕ^+ that can assume the joint state specified by $CR(\Phi)$ can achieve the goal. We can simulate ϕ^+ using $\Pi_{v \in CR(\Phi)}|D(v)|$ agents as follows. We choose the agent initial states according to the permutations of states for $CR(\Phi)$, while choosing the same states for all the other agent VSs according to ϕ^+. Given a plan for ϕ^+, we start from the first action. Given that any permutations of states for $CR(\Phi)$ is assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action with the following three cases (we show that all three cases can be simulated):

1) If this action updates an agent VS in $CR(\Phi)$, we do not need to execute this action based on the following reasoning. Given that all edges going in and out of agent VSs are directed, we know that this action does not update V_o. (Otherwise, there must be an undirected edge connecting a variable in V_o to this agent VS. Similarly, we also know that this action does not update more than one agent VS.). As a result, it does not influence the execution of the next action.

2) If this action updates an agent VS that is not in $CR(\Phi)$, we know that this action cannot have variables in $CR(\Phi)$ as preconditions or prevail conditions, since otherwise this agent VS would be included in $CR(\Phi)$ given its construction process. Hence, all agents can execute the action to update this agent VS, given that all the agent VSs outside of $CR(\Phi)$ are always kept synchronized in the entire process (in order to simulate ϕ^+).

3) Otherwise, this action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in ϕ^+'s plan to achieve the goal. Hence, the conclusion holds. □

The requirement on the traversability of ICGS in Lem. 2 is further relaxed below:

Corollary 2. *Given a solvable MAP problem with homogeneous agents, if all the edges going in and out of agent VSs are directed in the causal graphs, the minimum number of agents required is upper bounded by* $\Pi_{v \in VS(\Phi)}|D(v)|$*, assuming that the agents can choose their initial states.*

Proof. Given a valid plan π_{MAP} for the problem, we can solve the problem using $\Pi_{v \in VS(\Phi)}|D(v)|$ agents as follows: first, we choose the agent initial states according to the permutations of state for $VS(\Phi)$.

The process is similar to that in Lemma 2. We start from the first action. Given that all permutations of $VS(\Phi)$ are assumed by an agent, we can find an agent, denoted by ϕ, that can execute this action: if this action updates some agent VSs in $VS(\Phi)$, we do not need to execute this action; otherwise, the action must be updating only V_o and we can execute the action on ϕ.

Following the above process for all the actions in π_{MAP} to achieve the goal. Hence, the conclusion holds. □

The bounds above are upper bounds. Nevertheless, for our Burglary problem, the assumptions for both are satisfied and the 2 is returned for both, which happens to be exactly the k for which the problem is minimally k-agent solvable. In future work, we plan to establish the tightness of these bounds.

Analysis of Type-2 (Heterogeneous) RC

For the class of problems with heterogeneous agents, the most obvious cause for required cooperation (RC) in type-2 RC problems is the requirement of capabilities from different agents (due to domain, variable and capability heterogeneity). In the logistics domain, for example, the domain of the location variable for a truck agent can be used to force the agent from visiting certain locations in a city (domain heterogeneity – DH). When there are packages that must be transferred between different locations within a city, at least one truck agent that can access each location is required (hence RC). In the rover domain, a rover that is equipped with a camera sensor would be associated with the agent variable $equipped\_for\_imaging(rover)$. When we need both $equipped\_for\_imaging(rover)$ and $equipped\_for\_rock\_analysis(rover)$, and no rovers are

equipped with both sensors (variable heterogeneity – VH), we have RC. In the logistics domain, given that the truck cannot fly (capability heterogeneity – CH), when a package must be delivered from a city to a non-airport location of another city, at least a truck and a plane are required.

We note that 1) the presence of DH, VH or CH (i.e., the heterogeneity condition) in a solvable MAP problem does not always cause RC. In other words, a solvable MAP problem that satisfies the heterogeneity condition may not have RC (e.g., when the problem does not need the different capabilities); 2) the presence of the heterogeneity condition in a type-2 RC problem is not always the *sole cause* of RC. For example, the conditions that cause RC in type-1 problems may also cause RC in type-2 problems (see the mixed cause region in Fig. 3).

Due to the complexities above, to continue the analysis, we further divide MAP problems with heterogeneous agents into two subsets as shown in Fig. 3, which in turn divides type-2 RC problems into two subclasses – problems in which RC can only be caused by the heterogeneity of the agents (termed *DVC RC*), and the remaining problems. In other words, no causes for type-1 (homogeneous) RC are present in DVC RC problems. For DVC RC, we can improve the planning performance using a simple compilation.

DVC RC in Type-2 RC

In particular, we show that the notion of *transformer agent* allows us to significantly reduce the number of agents to be considered in planning for DVC RC problems, which can be solved first with a single or a small set of transformer agents. The transformer-agent plans can then be expanded to use agents in the original problems.

Definition 16 (DVC RC)**.** *A DVC RC problem is an RC problem in which all agents have traversable causal graphs with no causal loops.*

DVC RC problems can be solved by the construction of transformer agents (defined below), which are *virtual* agents that combine all the domain values, variable signatures and action signatures of the agents in the original MAP problem (i.e., Φ). To ensure that this combination is valid, we make the following assumption: agent variables for different agents are positively (i.e., no negations in preconditions or prevail conditions) and non-exclusively defined. Exclusively defined variables can be compiled away. Two variables are exclusively defined when associating them with the same agent can introduce conflicts or lead to undefined states. For example, $using\_gas(AG_{EX})$ and $using\_kerosene(AG_{EX})$ can lead to undefined states, if an agent can use either gas or kerosene but the state in which both are used is undefined (i.e., the agent cannot be flying and driving at the same time). This issue can be compiled away, e.g., $using(AG_{EX}) = \{gas, kerosene\}$, potentially with a few complications to handle the transition between the values. Given a MAP problem $P = \langle V, \Phi, I, G\rangle$,

Definition 17 (Transformer Agent)**.** *A transformer agent is an agent ϕ^* that satisfies:* 1) $\forall v \in V_\Phi$, $\exists v^* \in V_{\phi^*}$, $D(v^*) = D(V)$, *in which* $V = \{v | v \in V_\Phi$ *and* $VS(v^*) = VS(v)\}$. 2) $VS(\phi^*) = VS(\Phi)$. 3) $AS(\phi^*) = AS(\Phi)$.

An intuitive way to think about a transformer agent is that it is a single agent that can "transform" into any agent in the original MAP problem. A transformer agent can use any other agent's capabilities (but not simultaneously). Before discussing how the initial states of these transformer agents can be specified, we introduce a subset of DVC RC problems that can be solved by efficiently.

Connected DVC RC: A subset of DVC RC problems, referred to as *Connected DVC RC* ($B.1.1$ in Fig. 3), can be solved by a single transformer agent. To define *Connected DVC RC*, we first define *state space connectivity* for agents.

Definition 18 (State Space Connectivity)**.** *Given two agents ϕ and ϕ' that have traversable causal graphs with no causal loops, denote their state spaces as S_ϕ and S'_ϕ, respectively, S_ϕ and S'_ϕ are connected if $\exists s \in S_\phi, \exists s' \in S'_\phi$, $VS(s) \cap VS(s') \neq \emptyset \wedge \forall v \in s_\cap, VS(s)[v] = VS(s')[v]$, in which $s_\cap = VS(s) \cap VS(s')$ and $VS(s)[v]$ denotes the value of variable v in state s.*

Intuitively, when two agents have connected state spaces, the transformer agent is allowed to transform from one agent to the other agent and vice versa in the shared states (i.e., s and s' above). This is necessary to ensure that a single transformer agent can traverse the state spaces of both agents. For example, the prerequisite for a truck-plane agent to be able to deliver a package that is at the airport of a city to a non-airport location in another city is that the truck and plane agents in the original problem must be able to meet at the destination airport to transfer the package.

Definition 19 (Connectivity Graph)**.** *In a DVC RC (or DVC MAP) problem, a connectivity graph is an undirected graph in which the nodes are the agents and any two nodes are connected if they have connected state spaces.*

Definition 20 (Connected DVC RC)**.** *A connected DVC RC problem is a DVC RC problem in which the connectivity graph is a single connected component .*

The result of Theorem 1 can be extended below:

Lemma 3. *Given a connected DVC RC problem, it is solvable by a single transformer agent for any specification of its initial state.*

Proof. Given that the causal graphs are traversable and contain no causal loops for all agents in DVC RC, the only condition that can cause RC is the heterogeneity condition (i.e., DH, VH or CH). Given that the state spaces of agents are connected, a transformer agent can traverse the state spaces of all agents. Hence, the problem is solvable by this transformer agent based on Theorem 1. □

Corollary 3. *A DVC RC problem in which all the goal variables lie in a single connected component in the connectivity graph can be solved by a single transformer agent, given a proper specification of the initial state.*

The initial state of the transformer agent only needs to lie within the connected component. We refer to problems that can be solved by a single transformer agent as *transformer-agent solvable* ($B.1.1$) in Fig. 3. Many problems in the IPC domains belong to Connected DVC RC (e.g., logistics and rover domains) and are transformer-agent solvable.

Lemma 4. *Given a DVC RC problem $P = \langle V, \Phi, I, G\rangle$ in which all the goal variables lie in a single connected component in the connectivity graph, the single transformer-agent plan that solves this problem can be expanded to a multi-agent plan using Φ.*

Proof. The proof is by construction. Given the initial state of the transformer agent, from previous discussions, we know that the transformer agent "is assuming the form" of an agent in Φ for which the transformer agent is executing an action (i.e., the first action in the single transformer-agent plan). Given that this agent has a traversable causal graph with no causal loops, we can plan it to reach the current state of the transformer agent while keeping the values of variables in V_o, and then let it execute the first action. The same process continues until the last action of the transformer agent is executed, and the goal is achieved. □

For example, in the logistics domain, suppose that we have a package to be delivered to a non-airport location in city c. The package is initially at the airport in city b, the plane agent is at the airport in city a, and the truck agent is at a non-airport location in city c. We solve this problem with a *truck-plane* agent initially at the airport in city b. This transformer agent can fly to c with the package, "transform" to the truck agent, drive to the non-airport location to deliver. To create the plan for the original problem, we need to first expand the single transformer-agent plan by sending the plane agent from city a to b. We then follow the transformer-agent plan until the package arrives at the airport in city c. Next, we expand the plan again by sending the truck agent to the airport in c to pick it up. The "transformation" forces the plane agent to unload and the truck agent to load the package at the airport. The plan then follows through.

Corollary 4. *Given a DVC RC problem in which the connectivity graph is separated into connected components, the number of transformer agents to solve the problem is upper bounded by the number of connected components (assuming proper specifications of the initial states); the plan can be expanded to use agents in the original problem.*

For DVC RC problems with more than one connected component in the connectivity graph ($B.1.2$ in Fig. 3), we can similarly expand the multiple transformer-agent plans into plans for Φ in the original problem.

Hence, given a MAP problem with heterogeneous agents, we can first construct the causal graphs for all agents and execute algorithms to determine whether the causal graphs are traversable and loop free. If the problem is not determined to belong to DVC MAP, we can use any single-agent planner to solve the problem by considering agents as resources. Otherwise, we first create the connectivity graph. We do not need to construct the exact graph; a partial graph (with a subset of the edges) only increases our estimation of the number of transformer agents needed. When more time is allowed, we can continue completing the graph. At any time during this process, if the graph is determined to be connected, we can stop immediately, in which case we know that we have a connected DVC RC problem. When the upper bound is estimated from the graph, we can create a set of transformer agents accordingly to solve the problem. If a plan is not found, we know that the MAP problem is unsolvable; otherwise, we can then expand the plan into a plan for the original problem. A similar process can be used to implement a planner with homogeneous agents in which case the results from Lemma 2 and Corollary 2 can also be utilized.

Using the Transformer Agent Compilation

We now turn our attention to performance in practice. Specifically, we show how the transformer agent compilation can be used to improve the planning performance. We compare with one of the best performing centralized planners, MAP-LAPKT (planner entry is SIW+-then-BFS(f)) (Stolba, Komenda, and Kovacs 2015; Muise, Lipovetzky, and Ramirez 2015), in CoDMAP, MAP-LAPKT also uses a compilation approach in which a MAP problem is converted into a single-agent planing problem; an off-the-shelf planner is then used to solve it. MAP-LAPKT's performance is very close to the best MA planner in CoDMAP (which is ADP). We could not compare against ADP since their approach does not use compilation and is incorporated as a heuristic in the planning process.

We implemented a planner called RCPLAN. First, we determine whether the problem belongs to connected DVC MAP (e.g., whether the causal graph is traversable and contains no causal loops, and whether the connectivity graph is a single connected component, which are often determined by the domain). If it is, we compile the problem into a problem with a single transformer agent based on Def. 17. We then solve this new problem with an existing planner (i.e., FastDownward). Finally, we use Metric FF to expand the tranformer-agent plan to a plan for the original problem as explained in Lemma 4.

To remove the influence of the underlying planner, we replace the internal planner in MAP-LAPKT with our Fast-Downward (without optimizations). For CoDMAP competition domains, we use 11 out of 12 domains, since the Wireless domain does not satisfy our compilation criteria (i.e., the goals contain agent variables). Due to the additional expansion process, we expect RCPLAN to improve performance particularly over large problems (in terms of number of agents). Hence, besides the CoDMAP domains, we also generate larger problems for three CoDMAP domains (i.e., Rover, Blocksworld and Zenotravel) using standard IPC generators.

The results are listed in Table 1. Both RCPLAN and MAP-LAPKT are given 30min to solve each problem. We use the IPC Agile (for time) and IPC Sat (for plan quality) scores. [3] The experiments were run on a 3.0 GHz quad-core linux machine with 7GB memory. We can see that RCPLAN performs better than MAP-LAPKT in time perfor-

[3] For each problem: the IPC Agile score is $\frac{1}{(1+\log_{10}(T/T^*))}$ where T is the time taken by a given planner and T^* is the time taken by the fastest planner for that problem; similarly, the formula for IPC Sat score is $\frac{Q}{Q^*}$ where Q is the plan quality produced by a given planner and Q^* is the highest quality produced for the problem. For most domains, we use the inverse of the plan length as the quality. The final scores are the sum of scores for all problems.

Problem Type	Coverage		IPC Agile Score (Time Score)		IPC Sat Score (Quality Score)		# Domains	# Agents
	RCPLAN	MAP-LAPKT	RCPLAN	MAP-LAPKT	RCPLAN	MAP-LAPKT		
CoDMAP Problems	**219 (98.6%)**	214 (96.4%)	**214.37**	186.73	187.31	**204.08**	11	2 - 20
Large Problems	**51 (98.1%)**	41 (78.8%)	**44.54**	34.90	**49.38**	38.81	3	2- 50

Table 1: Performance Comparison between RCPLAN and MAP-LAPKT

mance (i.e., IPC Agile) consistently, although slightly worse in IPC Sat for CoDMAP problems. This performance improvement is mainly due to the reduction of instantiated actions in the compiled transformer agent problem. For example, for one of the Zenotravel problem with 6 agents, RCPLAN instantiated 9152 actions while the number of agent actions used in MAP-LAPKT is 54912. Our planner achieves a higher IPC Sat score for the larger problems due to the coverage. We can always post-process plans to improve quality (Nakhost and Mller 2010). Our results confirm that many IPC domains (also chosen in CoDMAP as MAP domains) are in fact (single) transformer-agent solvable!

Conclusions

In this paper, we introduced the notion of required cooperation (RC). First, we showed that directly querying for RC is intractable. As a result, we started the analysis with a class of more restrictive problems where agents are homogeneous. We identified an exhaustive set of causes of RC for this class and provided bounds for the number of agents required in different problem settings. For the remaining problems where agents are heterogeneous, we showed that a subclass of problems in which RC is only caused by the heterogeneity of the agents (i.e., DH, VH or CH) can be solved with a smaller number of transformer agents than with the agents in the original problems.

This RC analysis makes theoretical contributions to the understanding of multi-agent planning problems, informs the design of future multi-agent planning competitions, and presents practical applications, e.g., determining how many agents to be assigned to each given task when agent resources are limited. We implemented a planner using our theoretical results and compared it with one of the best IPC CoDMAP planners. Results show that our planner improved performance on most IPC CoDMAP domains, which incidentally implies that only a subset of MAP problems were covered (i.e., Connected DVC RC) in this competition.

Acknowledgments This research is supported in part by the ONR grants N00014-13-1-0176, N00014-13-1-0519 and N00014-15-1-2027, and the ARO grant W911NF-13-1-0023. The authors would also like to thank Sushovan De for help in coming up with the Burglary problem.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI*, 929–935.

Bacchus, F., and Yang, Q. 1993. Downward refinement and the efficiency of hierarchical problem solving. *AIJ* 71:43–100.

Backstrom, C., and Nebel, B. 1996. Complexity results for sas+ planning. *Computational Intelligence* 11:625–655.

Brafman, R. I., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *AIJ* 198(0):52 – 71.

Brafman, R. I. 2006. Factored planning: How, when, and when not. In *AAAI*, 809–814.

Bylander, T. 1991. Complexity results for planning. In *IJCAI*, volume 1, 274–279.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007a. When is temporal planning really temporal? In *IJCAI*, 1852–1859.

Cushing, W.; Weld, D. S.; Kambhampati, S.; Mausam; and Talamadupula, K. 2007b. Evaluating temporal planning domains. In *ICAPS*.

Decker, K. S., and Lesser, V. R. 1992. Generalizing the partial global planning algorithm. *International Journal of Cooperative Information Systems* 1:319–346.

Durfee, E., and Lesser, V. R. 1991. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21:1167–1183.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Jonsson, A., and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *ICAPS*, 114–121.

Knoblock, C. 1994. Automatically generating abstractions for planning. *AIJ* 68:243–302.

Kvarnstrom, J. 2011. Planning for loosely coupled agents using partial order forward-chaining. In *ICAPS*.

Muise, C.; Lipovetzky, N.; and Ramirez, M. 2015. Mapplapkt: Omnipotent multi-agent planning via compilation to classical planning. *(CoDMAP-15)* 14.

Nakhost, H., and Mller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *ICAPS*, 121–128.

Nissim, R., and Brafman, R. I. 2012. Multi-agent a* for parallel and distributed systems. In *AAMAS*, volume 3, 1265–1266.

Pape, C. L. 1990. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *ICRA*.

Stolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of distributed and multiagent planners (CoDMAP). `http://agents.fel.cvut.cz/codmap/results-summer`.

Torreno, A.; Onaindia, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. In *European Conference on Artificial Intelligence*, volume 242, 762–767.