Quantifying Privacy in Multiagent Planning

Roman van der Krogt

Cork Constraint Computation Centre

Department of Computer Science

University College Cork, Cork, Ireland

tel: +353-21-425 5458; fax: +353-21-425 5424; email: roman@4c.ucc.ie

Abstract

Privacy is often cited as the main reason to adopt a multiagent approach for a certain problem. This also holds true for multiagent planning. Still, a metric to evaluate the privacy performance of planners is virtually non-existent. This makes it hard to compare different algorithms on their performance with regards to privacy. Moreover, it prevents multiagent planning methods from being designed specifically for this aspect.

This paper introduces such a measure for privacy. It is based on Shannon's theory of information and revolves around counting the number of alternative plans that are consistent with information that is gained during, for example, a negotiation step, or the complete planning episode. To accurately obtain this measure, one should have intimate knowledge of the agent's domain. It is unlikely (although not impossible) that an opponent who learns some information on a target agent has this knowledge. Therefore, it is not meant to be used by an opponent to understand how much he has learned. Instead, the measure is aimed at agents who want to know how much privacy they have given up, or are about to give up, in the planning process. They can then use this to decide whether or not to engage in a proposed negotiation, or to limit the options they are willing to negotiate upon.

Keywords: multiagent planning, privacy

1 Introduction

Alice and Bob each own a small transportation company. Both companies are often faced with "empty rides", where they have to bring cargo somewhere, but have no cargo for the return trip. Needless to say, the margins are thin on such trips. In order to increase their profits, they decide to try to reduce the number of empty rides by cooperating. When Bob has cargo from, say, Paris to Roma, and Alice from Nice to Bruxelles, Bob will send his truck on the tour Paris-Roma-Nice-Bruxelles and charge Alice a reduced fee. Of course, this also applies to Alice taking on tasks from Bob, and so they both profit from such a scheme. After a successful trial, Alice and Bob decide to automate the process of finding empty rides. As they both find it unacceptable to give up their autonomy, they decide to extend the planning software that each of them uses with an interaction module. The interaction modules exchange information about the plans, and build a "global plan", capturing information from both plans that may be used in finding the empty rides. This works well, and there is much rejoicing at both companies' increased revenue.

Some time later, Eve asks to join the system. Figuring that there is more chance of compatible rides with three companies than with two, Alice and Bob allow Eve to install a similar interaction module to her planning system. Things go well for a while, but then an important client of Bob's does not prolong their contract, citing a cheaper alternative. Shortly after, Alice experiences the same. Upon investigation, it turns out that Eve has learned about the cost structures of Alice and Bob through the information that was exchanged among the three parties and has used this knowledge to her advantage. Suddenly, Alice and Bob have a very different opinion on automated interaction...

While exaggerated and simplified, the above scenario presents a real concern that self-interested agents may have when it comes to multiagent planning. The agents employed a (simplistic) multiagent planning (MAP) technique in order to preserve their autonomy, only to find out that their privacy was grossly intruded upon. For self-interested agents that value their sense of autonomy and privacy this is unacceptable. Unfortunately, though, while privacy is often cited as one of the reasons behind multiagent planning techniques,

there has been very little attention paid to defining just what privacy means in the context of MAP.

In principle, multiagent planning offers a way to cooperate while being in control of which information is shared and with whom. Yet it is impossible to cooperate without sharing *any* information. At the very least, a pair of cooperating agents has to agree on which sub tasks are being carried out by one agent on behalf of the other, an approach taken by, for example, the MPOPR system [33] and the plan merging approach of De Weerdt [5]. Several approaches, such as (Generalised) Partial Global Planning (GPGP; see, for example, [7]) go even further and share detailed parts of their plans, or even their full plans, as in the case of (IG-)DGP [14] and the unnamed system proposed by Zhang et al. [42]. In the latter approaches, more information is exchanged than in the former. Clearly, this must lead to a poorer performance when it comes to privacy. This raises the obvious question of how to measure this performance. Just how much privacy is lost by exchanging certain information? How can we evaluate which method is better than another when privacy is concerned? Unfortunately, these questions are not easily answered, as evidenced by the lack of existing work despite numerous referrals to 'privacy'. Firstly, we have to decide what the value of the information is that is obtained. As plans have potentially many parallel actions (that may possibly overlap in time) and have an unknown length, this is not straightforwardly defined. We then have to compare this to a base case, where we are faced with the same questions. Moreover, we cannot answer these questions without some model of the other agent's capabilities.

In this paper, we present an answer to these questions in the form of a metric for privacy loss based on Shannon's Theory of Information [23]. We show how the concept of "uncertainty" that underpins Shannon's work can be interpreted in the context of plans, and how we can derive a measure from this for the information that is gained during negotiations on plan construction or coordination. Intuitively, the uncertainly relates to the number of possible plans the agent may have.[1] In this work, we are primarily interested in the "form"

[1] Our earlier work [31, 32] proposed a definition for the classical case (totally ordered plans in unit time) based on the number of different actions that are possible in a given time step. The present work allows for parallel, durative actions.

of the plan (i.e. modulo different objects). Therefore, we consider just the action types, not the fully instantiated actions. To count the possible plans, we need additional information on the planning domain. This information is used to rule out valid, yet illogical plans from being taken into consideration. For example, in a lot of domains, it is possible to "undo" an action (e.g. in the well-known Blocks domain [13], we can put a block onto the table and then back to its original position). Sequences of doing a certain action followed by an immediate undoing of the same are certainly possible, but almost never rational. Because of the additional information that is required, the measure is particularly useful for agents that are the *target* of privacy invasion, and want to assess their risks. For example, during negotiations, they can evaluate the different options on their privacy aspects, and take this into account when making bids or proposals. When used as such, the measure provides a *worst-case analysis*, for in reality the other agent (usually) has less complete knowledge and hence a higher degree of uncertainty. Since one cannot know what the other agent believes about ones domain, we believe this is the safest option.

The remainder of the paper is organised as follows. In the next section we give a formal definition of the multiagent planning problem as we consider it in this paper. Then, we give a brief overview of our approach in Section 3, followed by a definition of the additional information we require in Section 4. In Section 5, then, we propose an algorithm to compute our measure. Before concluding, we give an overview of related work in Section 6.

2 The Multiagent Planning Problem

In this section we will define what we understand by a multiagent planning problem. A standard definition for multiagent planning does not exist, unfortunately, and most people have their own intuition. In order to prevent any confusion, we define the problem as considered in this paper. Intuitively, we consider a group of agents. Each agent has its own, private planning problem. This consists of *(i)* the set of operators that this agent may carry out, *(ii)* that part of the (common) initial state the agent is aware of, and *(iii)* a set of goals. Notice that all agents have the same initial state, although they may have a limited view

of it. Thus, the initial states may differ, but have to be consistent. Since, in general, the goals of different agents may be mutually exclusive, we deal with a form of *over subscription* planning [30].

Definition 1. Formally, we consider an instance of the multiagent planning problem for a set of n agents $A = \{1 \dots n\}$ to be a set $\Pi_A = \bigcup_{i \in A} \Pi_i$. Following [3], we define the problem Π_i of agent i is a tuple $\Pi_i = (F_i, O_i, I_i, G_i)$, where

- F_i is a set of ground atomic formulae, the propositions used to describe the domain of agent i;
- $O_i \subset \mathbb{O}$ is the set of operators that agent i may execute. Each operator o has parameters $\mathit{param}(o)$, preconditions $\mathit{prec}(o) \subset F_i$, and effects $\mathit{eff}(o) \subset F_i$, where we distinguish the positive effects $\mathit{eff}^+(o)$ and the negative effects $\mathit{eff}^-(o)$. Moreover, with each action o is associated a duration $\mathit{dur}(o) \in \mathbb{N}^*$. The set of all operators of all agents is denoted by $\mathbb{O}$. Notice that agents may have overlapping sets of capabilities;
- $I_i \subset F_i$ describes the initial state visible to agent i. The global initial state is denoted by $I = \bigcup_{i \in A} I_i$ and is conflict-free; and
- $G_i \subset F_i$ is the set of goals i wants to attain. In contrast to I, the set of all goals $G = \bigcup_{i \in A} G_i$ may have conflicts.

The solution to a multiagent planning problem is a multiagent plan, a (partially) ordered sequence of actions. This plan can be split into separate plans for each agent.

Definition 2. We consider a plan to be a (partially) ordered sequence of actions $\Delta_A = \langle \{o_1^{a_1}, \dots, o_m^{a_m}\}, \prec, \tau \rangle$, where each $o_i^{a_i} \in O_{a_i}$ is the instantiation of an action belonging to agent a_i, $\prec$ specifies the ordering of the actions and $\tau : \mathbb{O} \to \mathbb{N}^*$ is a (partial) function that assigns a start time $\tau(o_i^{a_i})$ to each action.

Specifically, $o_1 \prec o_2$ means that o_1 finishes before o_2 may start. The transitive closure of $\prec$ is denoted by $\prec^*$; its transitive reduction by $\prec^-$. If neither $o_1 \prec^* o_2$ nor $o_2 \prec^* o_1$, then o_1 and o_2 can be executed in parallel. The relation $\preceq$ and its transitive closure $\preceq^*$ take into

account this possible parallelism, i.e. $o_1 \preceq o_2 \Leftrightarrow o_1 \not\succ^* o_2$. The actual start times $\tau(o_i^{a_i})$ of the actions have to respect the order as specified by $\prec$ (i.e. for actions $o_1^{a_1}$ and $o_2^{a_2}$, $o_1^{a_1} \prec o_2^{a_2}$, implies that $\tau(o_2^{a_2}) \geq \tau(o_1^{a_1}) + dur(o_1^{a_1})$), but allows for slack in a plan. Notice that actions that can be executed in parallel may or may not start at the same time, or indeed, overlap in time.

Definition 3. The sub plan of Δ_A for a particular agent $i \in A$ is $\Delta_i = \langle \{o_j \mid o_j^i \in \Delta\}, \prec_i \cup \prec_A, \tau_i \rangle$, where $\prec_i$ lists the ordering constraints between the actions of the plan of agent i, $\prec_A$ lists the ordering constraints of i with respect to other agents (i.e. it lists those tuples of which exactly one of the actions belongs to i) and τ_i is the restriction of τ to actions o_j^i. To preserve all orderings in the plan, we define $\prec_i$ as follows: for two actions $o_k, o_l \in \Delta_i$, $o_k \prec_i o_l$ iff either *(i)* $o_k \prec o_l$ or *(ii)* $o_k \prec^* o_l$ and for all actions $o_j \in \Delta_A$, $o_k \prec^* o_j \prec^* o_l \Rightarrow o_j \notin \Delta_i$.

Notice that $\prec_i$ is not simply $\{o_k \prec o_l \mid o_k^i \prec o_l^i\}$, i.e. all tuples of which both actions belong to the plan of i, as this would cause unintentional parallelisation when a plan consists of several parts, with actions of other agents in between. The space of all plans is denoted by $\mathbb{P}$; this includes both the individual plans and multiagent plans.

For the purpose of this paper, we are mainly concerned with the individual plans of the agents. For convenience, we shall therefore refer to the sub plan for a particular agent simply as a plan $\Delta = \langle \{o_1, \ldots, o_m\}, \prec, \tau \rangle$, omitting the identification of the owner agent and the distinction between $\prec_i$ and $\prec_A$.

3 Overview of The Approach

3.1 Information Theory

Our ideas revolve around Shannon's Theory of Information [23]. It is based on the notion that the amount of information contained in a message exchange can be measured by the amount with which the uncertainty about certain facts decreases. Whereas Shannon followed a rigorous route in deriving his famous function, we follow the more intuitive explanation given by Schneider [22] in setting out the background.

Information is closely linked to uncertainty. Suppose we have M differently coloured balls in a hat, and we intend to randomly draw one. Now we are faced with a certain degree of *uncertainty* regarding the colour of the ball we will draw. When we draw a ball, we get some *information* (on the colour of this ball) and our uncertainty (regarding the colour of this ball) *decreases*. Shannon's work gives an answer to the questions of how to measure this uncertainty. If we assume an equal probability for all of the balls, we would like to say that we have an "uncertainty of M colours". However, we would also like our measure of uncertainty to be additive, which leads to the following formula for the uncertainty H:

$$H = \log_2(M) \tag{1}$$

What happens if there are fewer colours then balls, with some colours more likely than others? First, let us rearrange Equation 1 in terms of the probability $P = \frac{1}{M}$ that any colour is drawn: $H = \log_2(M) = -\log_2(\frac{1}{M}) = -\log_2(P)$. Now, let P_i be the probability of drawing colour i, with $\sum_{i=1}^{M} P_i = 1$. The "surprisal" [27] of drawing the i^{th} colour is defined by analogy with $-\log_2(P)$ to be

$$u_i = -\log_2(P_i) \tag{2}$$

In the generalised case, uncertainty is the average surprisal for the infinite string of colours drawn (returning each ball before drawing a new one). For a string of finite length N, with colour i appearing N_i times, this average is $\sum_{i=1}^{M} \frac{N_i}{N} u_i$. For an infinite string, the frequency $\frac{N_i}{N}$ approaches P_i, the probability of drawing this colour. The average surprisal therefore is:

$$\sum_{i=1}^{M} P_i u_i \tag{3}$$

Substituting for the surprisal (cf. Equation 2), we get Shannon's general formula for uncertainty:

$$H = -\sum_{i=1}^{M} P_i \log_2(P_i) \tag{4}$$

The unit for uncertainty is bits per symbol. The H function forms a symmetrical (multidimensional) curve that peaks when all symbols are equally likely and falls towards zero when one of the symbols becomes dominant.

At the start of this section, we said that information can be considered to be the decrease in uncertainty. Using Equation 4, we can express information. Information relates to communication and uncertainty as follows.

Definition 4. Suppose we have an uncertainty H_{before} before an event (such as the transmission of a message) and that uncertainty after the event is H_{after}. Then the information that was gained in the event equals

$$R = H_{before} - H_{after} \tag{5}$$

This is what Shannon calls the *rate of information transmission*. Thus, information always relates two points in time, and the uncertainties at those times. Information relates to privacy loss in planning as follows. Consider the case of two agents, entering into negotiations for some aspect of their planning problems. Before the negotiations, the uncertainty with regard to the other agent's plan equals some amount H_{before}. During their negotiations, agents may learn about certain aspects of the other agent's plan. In particular, they may receive information about certain actions (not) being executed at a certain time point. After the negotiations, the uncertainty will therefore be reduced to some amount H_{after}. The amount of information thus gained equals $H_{before} - H_{after}$. This is precisely the privacy that the other agent has lost.

3.2 Running Example

Before we discuss the details of our approach, let us give a simplified example of our approach. Consider the typical *Logistics* benchmark domain, in which an agent is tasked with transporting goods within a city. This domain has three types of actions: *move*, *load* and *unload*, which for the sake of simplicity, we assume to be mutually exclusive. Furthermore, assume that $dur(move) = 3$ and $dur(load) = dur(unload) = 2$. Suppose that we observe

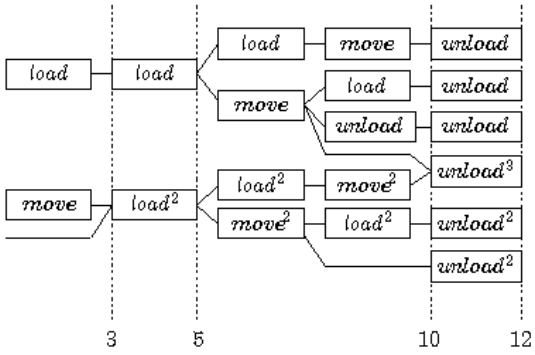

Figure 1: A graph of all possible (incomplete) plans of length 12, with a *load* action at time 3 and an *unload* action at time 12

a package being loaded at time 3 and being unloaded (at a different location) at time 10. What implications does this have for uncertainty about the plan between time 0 (the start of the plan) and, say, time 12 (when the *unload* action just finished)? Well, information theory suggests that we look at the number of possible options. Doing so, we can picture a graph that encodes all possible plans of length 12 (considering just the action types, not the specific objects involved), and compare that with the graph that has all possible plans of length 12, with a *load* action at time 3 and an *unload* action at time 12. Figure 1 shows the latter graph. Four time points are depicted in the graph: 3 and 5, which are the start and end of the *load* action, and 10 and 12, corresponding to the *unload* action. Before time step 3, we have three possibilities: either we have a *load* action, we have a *move* action, or nothing happens. Notice that an *unload* action can not take place, as this would require a preceding *load* action. The top half of the graph shows the possible plans that start with a *load* action at either time 0 or 1, before the *load* action that we have observed. Notice how, again, an *unload* action cannot take place at time 5 here, since a *move* is required between a *load*-*unload* pair. These are two of the constraints that the plan must adhere to.

The bottom half shows the possible plans for the other two options. Looking at the constraints on the remainder of the plan, we can see it makes no difference whether we start with a *move* action, or with no action. Therefore, we merge the *load* nodes at time 3, and record that there are two possible ways to arrive at this node (as signified by the superscript 2 in the label). For the same reason, we merge the *load*-*load*-*move* branch from the top half

with the X-$load^2$-$load^2$-$move^2$ branch from the bottom (where X is either a *move* or no action), obtaining an $unload^3$ node. Counting the total number of *unload* nodes at time 10 (taking into account nodes with multiple paths), we observe that a total of 10 possible plans exist, in which a *load* action is executed at time 3 and the corresponding *unload* action at time 10. Notice how we are observing a number of *incomplete* plans here.[2] For example, one would expect at least two more *unload* actions for the top-most path, which has 3 *load* actions and only one *unload* action.

The same graph for all plans of length 12 is left as an exercise to the reader. It has a total of 58 plans. Recall from Equation 1 that the uncertainty associated with M possible options is $H = \log_2(M)$. Hence, an agent who agrees with another agent to *load* an item at time 3, and to deliver it at time 10, incurs a privacy loss of $R = H_{before} - H_{after} = \log_2(58) - \log_2(10) = 2.536$. This is a 57% decrease over $H_{before} = \log_2(58) = 5.858$.

However, not all of these possible plans are equally likely. For example, one of the possible (incomplete) plans of length 12 has 6 consecutive *load* actions. This might be a very rare occurrence. Therefore, we will add probabilities (or rather: frequencies) to the graph to derive a weighted uncertainty, as per Equation 4. To capture those probabilities, as well as the constraints on the plans as we have seen above, we first present an alternate model of the planning problem.

4 Modeling the Constraints and Probabilities of a Planning Domain

As we have shown in the previous paragraph, in order to compute the number of possible plans, we need to list the constraints on those plans (e.g. an *unload* action can only occur after a corresponding *load* action), as well as probabilities, so that we can weigh the different plans in determining the total uncertainty. This knowledge is captured in a tuple $\langle \mathcal{C}, \mathcal{P} \rangle$

[2] Notice that this is different from a *partial* plan, as used in partial order planners [38]. A partial plan may have any number of flaws (such as unsatisfied preconditions, etc.), whereas our *incomplete* plans have no such flaws. They are unfinished in the sense that the identified constraints suggest that additional actions should be appended.

where $\mathcal{C}$ is a set of constraints, and $\mathcal{P}$ is a set of probabilities. The constraints represent the particular forms valid plans can take; the probabilities define the likelihoods of actions under these constraints. Domain analysis tools such as TIM [11] can support the construction of the set of constraints.

Again, we note that substantial knowledge about the agent's performance is required. This is especially true for the details of $\mathcal{P}$. Since it is unrealistic to assume that an opponent has such knowledge, this metric is not meant to be a measure that a particular agent can use to quantify the amount of information it has learned. Rather, it can be used by agents to estimate their loss. A typical use might be for an agent to weigh the potential benefit of a negotiation step versus the potential loss in information it may incur.

4.1 The Constraints $\mathcal{C}$

The set of constraints $\mathcal{C}$ defines some rules that plans have to adhere to. Some of these follow from the preconditions and effects of actions; others are more specific domain knowledge that rule out certain valid, but illogical plans. For example, in the logistics domain that was introduced before, the fact that an *unload* action has to be preceded by a *load* action follows from the preconditions and effects. However, it would be illogical to have these two actions one after the other without a *move* action in between, as the purpose of a *load* action is to bring the package to another location. Moreover, we can assume that planning problems in this domain never start with a package already loaded, so we can never satisfy an *unload* action from the initial state (this is further restriction of the first example we just gave).

The set of constraints $\mathcal{C}$ describes patterns that can be distinguished in valid, but more importantly, *actual* plans. This restricts the set of plans that we will count to plans that may actually occur. In each of those plans, we can identify at least one of the constraints in $\mathcal{C}$ that *justifies* each of the actions.

Definition 5. The set of all constraints is denoted by $\mathbb{C}$, and is defined as follows. Let $\mathbb{O}^{\mathbb{N}}$ be defined as the set of actions from $\mathbb{O}$, annotated with an identifier: $\mathbb{O}^{\mathbb{N}} = \{o^i \mid o \in \mathbb{O}, i \in \mathbb{N}^*\}$. The purpose of $\mathbb{O}^{\mathbb{N}}$ is to be able to distinguish different occurrences of the same action in a constraint, if necessary.

Then, we define *chaining constraints*, that describe what causal relationships may exist in a plan as:

1. For two (annotated) actions $o_1, o_2 \in \mathbb{O}^{\mathbb{N}}$,

   - $o_1 \rightarrow o_2$ is an atomic chaining constraint denoting that an occurrence of o_2 requires the occurrence of o_1 beforehand;
   - $o_1 \Rightarrow o_2$ is an atomic chaining constraint denoting that *each* occurrence of o_2 requires a *unique* occurrence of o_1 beforehand;[3]
   - $o_1 \times o_2$ is an atomic chaining constraint denoting that an occurrence of o_2 requires an occurrence of o_1 *immediately* preceding o_2; and
   - $\rightarrow o_1$ is an atomic chaining constraint denoting that o_1 may be present anywhere in the plan, without any (specific) occurrences of actions beforehand.

2. For actions $o_1, o_2 \in \mathbb{O}^{\mathbb{N}}$,

   - o_1 is an atomic *connective* (rather than constraint) requiring the presence of o_1;
   - $(o_1 \vee o_2)$ is an atomic connective requiring the presence of either of the actions o_1 or o_2 (or both); and
   - $(o_1 \parallel o_2)$ is an atomic connective requiring the presence of both actions o_1 and o_2 (either in parallel or one after the other).

   Let c_1, c_2 be connectives and $o \in \mathbb{O}^{\mathbb{N}}$, then $c_1 \vee c_2$, $c_1 \vee o$, $o \vee c$ and $c_1 \parallel c_2$, $c_1 \parallel o$, $o \parallel c_2$ are connectives too. For an (atomic) connective c, and $\rightsquigarrow \in \{\rightarrow, \Rightarrow, \times\}$, $c \rightsquigarrow o$ and $o \rightsquigarrow c$ are chaining constraints, as is $\rightarrow c$.

3. For two chaining constraints $c_1, c_2 \in \mathbb{C}$,

   - (c_1) is a chaining constraint;
   - $c_1 \wedge c_2$ is chaining constraint that requires both c_1 and c_2 to be true; and

[3] Thus, an action in the plan can satisfy the left-hand side of at most one $\Rightarrow$-constraint, but multiple $\rightarrow$-constraints.

- $c_1 \vee c_2$ is a chaining constraint that requires either c_1 or c_2 to be true (or both).

4. For a chaining constraint $c \in \mathbb{C}$, and an action $o \in \mathbb{O}^{\mathbb{N}}$,

   - $c \rightarrow o$, $o \rightarrow c$, $c \Rightarrow o$, $o \Rightarrow c$, $c \times o$ and $o \times c$ are also chaining constraints.

Secondly, we define *mutex constraints*, denoting pairs of actions that cannot occur together.

1. For actions $o_1, o_2 \in \mathbb{O}^{\mathbb{N}}$, $o_1 \between o_2$ is an atomic mutex constraint that denotes that o_1 and o_2 are mutually exclusive. Notice that $o_1 \between o_1$ means that an action is mutually exclusive with itself, i.e. only one execution of that action can be performed at any given time.

2. For two atomic mutex constraints $c_1, c_2 \in \mathbb{C}$, $c_1 \wedge c_2$ is a mutex constraint that requires both c_1 and c_2 to be true.

3. For actions $o_i \in \mathbb{O}^{\mathbb{N}}, i = 1..n$, $o_1 \between \cdots \between o_n$ is a mutex constraint denoting $\bigwedge_{i=1..n, j=1..n, i \neq j} o_i \between o_j$.

Example 6. The constraints we listed at the start of this section for the logistics domain can be expressed as follows.

- $(load^1 \rightarrow move^1 \rightarrow unload^1) \wedge (load^1 \Rightarrow unload^1)$. A *load* action precedes the corresponding *unload* action, and a *move* action is to be present in between (notice how a move action can support multiple $load \Rightarrow unload$ pairs); and
- $\rightarrow (load^1 \vee move^1)$. Only *load* and *move* actions can follow from the initial state (unlike *unload* actions);
- $move^1 \times move^2$. A *move* action may follow another *move* action; and
- $load \between load \between move \between move \between unload \between unload$. All actions are mutually exclusive, with each other and themselves.

Notice that the $move^1 \times move^2$-constraint is redundant, as the $\rightarrow move^1$-constraint would allow for a *move*-action to be satisfied from the initial state after an earlier one. This is

a highly unlikely case, however, as a single *move* suffices. It therefore warrants a special condition, so that we can assign it its own probability in the next section.

For brevity, we will concentrate on chaining constraints of the form $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$, where $\rightsquigarrow_i \in \{\rightarrow, \Rightarrow, \times\}$ in the definitions in the remainder of this section. All definitions are equally valid for chaining constraints of the form $c = \!\!\rightarrow o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$, however.

We can rewrite each chaining constraint into a DNF-like form, i.e. a disjunction of conjunctions, where each of the conjuncts is a chain of actions, of the form $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$, where $\forall i = 1..n \cdot o_i \in \mathbb{O}^{\mathbb{N}}, \forall j = 1..n-1 \cdot \rightsquigarrow_j \in \{\rightarrow, \Rightarrow, \times\}$ using the following rules. Let $\rightsquigarrow \in \{\rightarrow, \Rightarrow, \times\}$, then

1. If $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$, then $c \rightsquigarrow o$ is defined as $o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n \rightsquigarrow o$. Similarly for $o \rightsquigarrow c$.

2. If $c = c_1 \star c_2$, where c_1, c_2 are chaining constraints and $\star \in \{\wedge, \vee\}$, we can distribute the operator $\star$ as follows: $c \rightsquigarrow o$ is defined as $(c_1 \rightsquigarrow o) \star (c_2 \rightsquigarrow o)$. Similarly for $o \rightsquigarrow c$.

3. If $c = c_1 \vee c_2$, where c_1 and c_2 are connectives, $c \rightsquigarrow o$ is defined as $(c_1 \rightsquigarrow o) \vee (c_2 \rightsquigarrow o)$. Similarly for $o \rightsquigarrow c$ and $\rightarrow c$.
   If $c = c_1 \parallel c_2$, $c \rightsquigarrow o$ is defined as $(c_1 \rightsquigarrow o) \wedge (c_2 \rightsquigarrow o)$. Again, similarly for $o \rightsquigarrow c$ and $\rightarrow c$.

Each of the disjucts can then be construed as one of the constraints in the set $\mathcal{C}$. Thus, we assume $\mathcal{C}$ is normalised. We say that the actions $o_i, 1 \leq i \leq n$ are *part of* a chaining constraint $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$ and denote this by $o_i \in c$. Similarly for mutex constraints $c = o_1 \between \cdots \between o_n$.

Example 7. (Continued from Example 6.) The normalised form of the constraints in the

previous example is:

$$\begin{aligned}\mathcal{C} \;=\; \{&(load^1 \rightarrow move^1 \rightarrow unload^1) \wedge (load^1 \Rightarrow unload^1),\\ &\rightarrow load^1,\\ &\rightarrow move^1,\\ &move^1 \times move^2,\\ &load \between load \between move \between move \between unload \between unload\}\end{aligned}$$

A plan Δ satisfies a set of constraints $\mathcal{C}$ if all actions present in the plan are *justified* by one of the constraints and none of the mutex constraints is violated. For this purpose, we introduce an annotation scheme, that associates with each action in a plan the corresponding constraint that justifies it, instantiated to actual actions in the plan.

Definition 8. Given a chaining constraint $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$ and a plan $\Delta = \langle O = \{o'_1, \ldots, o'_m\}, \prec, \tau\rangle$, the *instantiation* of c, denoted by $c\theta$, is obtained by replacing each action $o_i \in c$ with an action of the same type in the plan according to a substitution $\theta : \mathbb{O}^{\mathbb{N}} \rightarrow O$. A substitution is called *full* (resulting in a *full instantiation*), if $\theta(o)$ is defined for each $o \in c$. An instantiation $c\theta$ is called *partial up to* o_j, if we can identify an action $o_j, 1 \leq j \leq n$ such that $\theta(o_k)$ is defined for all $k = 1..j$ and undefined for all $k = j + 1..n$. An instantiation is called *invalid* if it is neither full nor partial and *valid* otherwise.

Given a valid instantiation $c\theta$ for a chaining constraint $c = o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$ and a plan $\Delta = \langle O = \{o'_1, \ldots, o'_m\}, \prec, \tau\rangle$, $c\theta$ is *satisfied* by Δ, denoted by $\Delta \models c\theta$, iff either *(i)* for each instantiated action $\theta(o_i), i \geq 2$, it holds that $\theta(o_{i-1}) \prec^* \theta(o_i)$ (i.e., the precedence relations specified by the chain c have to be honoured in the plan), or *(ii)* only $\theta(o_1)$ is defined, and c is of the form $\rightarrow o_1 \rightsquigarrow_1 \ldots \rightsquigarrow_{n-1} o_n$ (i.e., if only o_1 is instantiated, it has to be satisfied from the initial state). For a constraint $c = \bigwedge_{i=1..m} c_i$, where each c_i is a chaining constraint $c_i = o_{i,1} \rightsquigarrow_{i,1} \ldots \rightsquigarrow_{i,n_i-1} o_{i,n_i}$ to be satisfied, we require that all of its conjuncts are satisfied, but also that the requirements of the $\Rightarrow$-constraints are honoured.

Definition 9. Given a plan Δ and a valid instantiated constraint $c\theta = \bigwedge_{i=1..m} c_i\theta$, we say

that $c\theta$ is satisfied by Δ, denoted by $\Delta \models c\theta$, iff

1. for all $c_i\theta, 1 \leq i \leq m$, it holds that $\Delta \models c_i\theta$; and
2. for all pairs c_i, c_j of conjuncts of c,
   - for all actions $o_{i,k_i} \in c_i$ and $o_{j,k_j} \in c_j$, if $o_{i,k_i} = o_{j,k_j}$, then $\theta(o_{i,k_i}) = \theta(o_{j,k_j})$, i.e. the instantiations of the conjuncts are consistent; and
   - for all $\Rightarrow_{i,k_i}$ and $\Rightarrow_{j,k_j}$, $\theta(o_{i,k_i-1}) \neq \theta(o_{j,k_j-1})$, i.e. no action in the plan can be used to satisfy the left-hand side of two $\Rightarrow$-constraints simultaneously.

Given a set of constraints $\mathcal{C}$, it *justifies* a plan Δ, if we can establish consistent justifications for each of the actions in the plan and none of the mutex constraints are violated.

Definition 10. Given a set of constraints $\mathcal{C}$, it *justifies* a plan Δ, denoted by $\mathcal{C} \models \Delta$, iff

1. for each action $o \in \Delta$ we can find a partial instantiation of a chaining constraint $c\theta$ up to o', where $c \in \mathcal{C}$ and $\theta(o') = o$, that is satisfied by Δ. Such a partial instantiation is called a *justification for* o;
2. no action in the plan is used to satisfy the left-hand side of two $\Rightarrow$-constraints simultaneously (as per Definition 9);
3. none of the mutex constraints is violated, i.e. for each pair of actions $o_1, o_2 \in \Delta$, if $o_1 \between o_2 \in \mathcal{C}$, then either $o_1 \prec^* o_2$ or $o_2 \prec^* o_1$.

Example 11. (Continued from Example 7.) One of the partial plans in Figure 1, is the following: *load*, *load*, *load*, *move*, *unload*. The justifications of the *load* and *move* actions can be straight-forward instantiations of the $\rightarrow load^1$ and $\rightarrow move^1$ constraints, respectively. The justification of the final *unload* action is depicted in Figure 2. Here, the arrows represent the substitution θ. Since a strict ordering is implied by this plan, none of the mutex constraints is violated and the plan satisfies the constraints.

Notice how the justification does not have to be unique. In this case, the *move* action can also be justified by the same constraint as the *unload* action is. In that case, the constraint

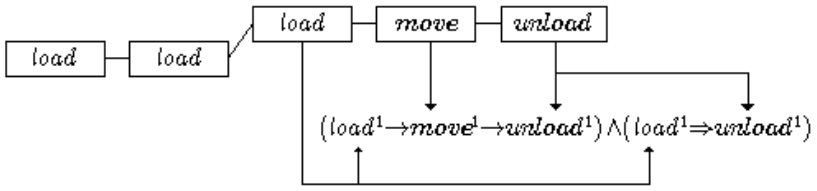

Figure 2: An example justification of the *unload* action of the plans of Figure 1.

would have to be partially instantiated, not substituting an action for the $unload^1$ in the constraint. (Of course, this would change nothing to the justification of the *unload* action.)

In case there are multiple justifications for a given action, we prefer the *largest* justification. A justification $c_1 = o_1 \rightsquigarrow \cdots \rightsquigarrow o_{n_1}$ is larger than a justification $c_2 = o_1 \rightsquigarrow \cdots \rightsquigarrow o_{n_2}$ if c_1 forms a longer chain, i.e. $n_1 > n_2$. This largest justification of an action o is denoted by $\gamma(o)$; $\Gamma(\Delta) = \bigcup_{o \in \Delta} \gamma(o)$ denotes the set of largest justifications for each action in a plan Δ.

In the next section, we show how we can generate plans using these constraints. Instead of finding a substitution that satisfies a constraint for a given plan, we can "grow" plans by expanding them with actions that would be justified by the plan. First, however, we introduce probabilities, so that we can establish the likelihood of the generated plans.

4.2 The Probabilities $\mathcal{P}$

The constraints show what kind of plans are valid. They do not show us, however, how likely the generated plans are. This is what the probabilities $\mathcal{P}$ will specify. Intuitively, $\mathcal{P}$ lists the probabilities of actions, given their justifications. By multiplying the probabilities for all actions in a plan, we get the total probability for a plan: $\mathcal{P}(\Delta)$. In general, we are often dealing with incomplete plans. Therefore, the probability $\mathcal{P}(\Delta)$ here should not be regarded as the probability of the plan in itself, but rather as the probability that *a plan has Δ as a sub plan.*

We start by defining the set of probabilities $\mathcal{P}$ that is part of the tuple $\langle \mathcal{C}, \mathcal{P} \rangle$. This is then extended to a probability function $\mathcal{P} : \mathbb{P} \rightarrow \mathbb{R}$ that associates a probability with a plan.

Definition 12. Formally, the set of probabilities $\mathcal{P}$ is a map $\mathbb{C} \rightarrow [0, 1\rangle$. Given a tuple

$\langle \mathcal{C}, \mathcal{P} \rangle$, $\mathcal{P}$ associates a probability $\mathcal{P}(c)$ to *each* of the (normalised) chaining constraints $c \in \mathcal{C}$. Additionally, for each chaining constraint c of the form $c = \rightarrow o_1 \rightsquigarrow_1 \cdots \rightsquigarrow_{n-1} o_n$ it assigns a probability to the constraint $\not\Diamond \rightarrow o_1 \rightsquigarrow_1 \cdots \rightsquigarrow_{n-1} o_n$, denoted by $\not\Diamond c$. The purpose of the latter probabilities is to distinguish between action occurrences *in* the plan and those at the very *beginning* of a plan.

The true probability of a constraint is usually unknown. However, we assume that we have access to previous plans, from which we can estimate the probability by the respective frequencies.

Example 13. (Continued from Example 7.) Given the set $\mathcal{C}$ of constraints for the logistics domain, we may observe the following probabilities $\mathcal{P}$:

$$\begin{aligned}
\mathcal{P} \quad = \quad & \big\{ \big[(load^1 \rightarrow move^1 \rightarrow unload^1) \wedge (load^1 \Rightarrow unload^1) \big] \mapsto 0.50, \\
& \big[\rightarrow load^1 \big] \mapsto 0.30, \\
& \big[\rightarrow move^1 \big] \mapsto 0.15, \\
& \big[move^1 \times move^2 \big] \mapsto 0.05, \\
& \big[\not\Diamond \rightarrow load^1 \big] \mapsto 0.55, \\
& \big[\not\Diamond \rightarrow move^1 \big] \mapsto 0.45 \big\}
\end{aligned}$$

The interpretation of a set $\mathcal{P}$ of probabilities is as follows: suppose we have a plan Δ with justifications $\Gamma(\Delta)$. Then, the probability of Δ equals the product of the probabilities of the individual justifications. However, for justifications $\gamma(o)$ of actions o at the beginning of a plan (i.e. actions $o \in \Delta$ for which holds $\nexists o' \in \Delta \cdot o' \prec^* o$), we have to use the probability of the constraint $\not\Diamond \gamma(o)$.

Definition 14. Let $\gamma^{\not\Diamond}(o)$ be defined as

$$\gamma^{\not\Diamond}(o) = \begin{cases} \not\Diamond \gamma(o) & \text{if } \nexists o' \in \Delta \cdot o' \prec^* o \\ \gamma(o) & \text{otherwise} \end{cases}$$

Given a tuple $\langle \mathcal{C}, \mathcal{P} \rangle$ and a plan Δ, such that $\mathcal{C} \models \Delta$, using a set of (largest) justifications $\Gamma(\Delta)$. Then, the probability of the plan, given by the function $\mathcal{P}(\Delta) : \Delta \to [0, 1\rangle$ equals

$$\mathcal{P}(\Delta) = \prod_{o \in \Delta} \mathcal{P}(\gamma^{\not\emptyset}(o))$$

Example 15. (Continued from Examples 11 and 13.) The *load*, *load*, *load*, *move*, *unload* plan discussed in Example 11 has an associated probability $0.55 \times 0.30 \times 0.30 \times 0.50 \times 0.50 = 0.012375$, assuming the figures given in Example 13. This amounts to an uncertainty of $H = -0.012375 \times \log_2(0.012375) = 0.078$.

This concludes our definition of the additional domain knowledge $\langle \mathcal{C}, \mathcal{P} \rangle$ that is required to compute the number of possible plans. In the next section, we will show how we can use this knowledge to determine privacy loss.

5 Counting Plans

In Section 3.2, we gave an introductory example of counting plans, cf. Figure 1. However, in order to count the plans, we don't have to actually generate the graph that we used to show this number. Instead, we will generate the plans and count them as we go along. Our algorithm is loosely based on that of the refinement planning approach [15, 16]. It is a recursive function, that expands a plan in each iteration, generating all successors and calling itself for each of those. The process stops when we have counted all plans for a given threshold value.

The algorithm itself is given in Algorithm 1. It takes as input a partial plan $\Delta = \langle \{o_0, o_1, \ldots, o_m, o_\infty\}, \prec, \tau \rangle$, consisting of actions o_0 and o_∞ to delimit the start and end of the plan, respectively, plus all the all the actions $o_1, \ldots, o_m$ that the opponent agent knows.[4] The start times τ are initialised to the start times of the actions, with $\tau(o_0) = 0$ and $\tau(o_\infty) = \infty$. The ordering constraints $\prec$ are initialised accordingly. Notice that if we are interested about the intrusion by a coalition of agents that share their knowledge,

[4] The base case, in which the opponent has no information at all, can be computed using a plan $\Delta = \langle \{o_0, o_\infty\}, \prec, \tau \rangle$.

Algorithm 1: $\textsc{count}(\Delta, \langle \mathcal{C}, \mathcal{P} \rangle, t, \sigma)$

Input: a partial plan $\Delta = \langle \{o_0, o_1, \ldots, o_m, o_\infty\}, \prec, \tau \rangle$;
domain constraints and probabilities $\langle \mathcal{C}, \mathcal{P} \rangle$;
a time step t; and
a threshold σ

Output: the number of possible plans that Δ can be expanded to, given $\langle \mathcal{C}, \mathcal{P} \rangle$

begin
  $\mathsf{count} \leftarrow 0$
  $\mathsf{extensions} \leftarrow \emptyset$
  $\mathsf{plans} \leftarrow \emptyset$
  $\mathsf{fringe} \leftarrow \operatorname{argmin}_{o \in \Delta \cdot \tau(o) \geq t}(\tau(o))$
  `// if Δ is justified by C, count it;`
  `// if it cannot be justified, stop`
  if $\mathcal{C} \models \Delta$ **then**
    $\mathsf{count} \leftarrow \mathsf{count} + \mathcal{P}(\Delta) \log_2(\mathcal{P}(\Delta))$
  else if $\mathcal{C} \not\models \langle \{o \mid o \in \Delta \cdot o \prec^* \mathsf{fringe}\}, \prec, \tau \rangle$ **then**
    return 0
  `// determine the applicable actions`
  forall $o \in \mathbb{O}$
    if $\rightarrow o \in \mathcal{C}$ **or**
      $\exists o' \in \Delta \cdot \tau(o') \geq t \wedge o \rightsquigarrow^* o' \in \mathcal{C}$ **or**
      $\exists o'' \in \Delta \cdot \tau(o'') < t \wedge \gamma(o'')\theta \circ o \models \Delta$ **then**
      $\mathsf{extensions} \leftarrow \mathsf{extensions} \cup \{o\}$
  `// expand the plan with applicable actions taking into`
  `// account possible parallelism`
  forall $O \in 2^{\mathsf{extensions}}$
    if $O \models \mathcal{C}$ **then**
      $\tau' \leftarrow \textsc{computeStartTimes}(\{o_1, \ldots, o_m\} \cup O, \prec \cup \{o \prec \mathsf{fringe} \mid o \in O\}, \tau, \mathcal{C})$
      if $\tau' \neq \bot$ **then**
        $\Delta' \leftarrow \{\langle \{o_1, \ldots, o_m\} \cup O, \prec \cup \{o \prec \mathsf{fringe} \mid o \in O\}, \tau' \rangle\}$
        if $\mathcal{P}(\Delta') \geq \sigma$ **then**
          $\mathsf{plans} \leftarrow \mathsf{plans} \cup \{\Delta'\}$
  `// count the plans recursively`
  $t(\Delta') \leftarrow \begin{cases} \tau(\mathsf{fringe}) & \text{if } \Delta \neq \Delta' \\ \tau(\mathsf{fringe}) + 1 & \text{otherwise} \end{cases}$
  $\mathsf{count} \leftarrow \mathsf{count} + \sum_{\Delta' \in \mathsf{plans}} \textsc{count}(\Delta', \langle \mathcal{C}, \mathcal{P} \rangle, t(\Delta'))$
  return count
end

we can simply add all actions known by the coalition. Furthermore, we take as input the domain knowledge $\langle \mathcal{C}, \mathcal{P} \rangle$, a time step t (initially 0), that keeps track of which part of the possibilities we have already tried, and a threshold σ that determines the minimum probability that we are interested in. The algorithm starts by initialising some variables in lines 2-5. The variable fringe is initialised to the first action that starts at time t or greater. In the following, we will only consider adding actions as new justifications for *later* actions, and expand upon partial justifications for *earlier* actions.

Next, lines 6-9, we count the plan Δ if it is justified by the constraints (which means that it is a valid possibility that has to be taken into account), or if there are any actions before time t that are not justified. If there are any unjustified actions before time t, it means that we can no longer justify them, as we only consider justifications after time t in the next steps, as explained above. In that case, we have reached a dead end.

Then, lines 10-14, we determine which actions are applicable and collect those in the set extensions. An action o is applicable if either *(i)* the constraints $\mathcal{C}$ allow for the action to be satisfied from the initial state, i.e. $\rightarrow o \in \mathcal{C}$ (this includes chaining constraints of the form $\rightarrow o \rightsquigarrow \cdots$), *(ii)* o is part of a chaining constraint that can provide a justification for a later action o' in the plan (i.e. actions starting after time t) or *(iii)* there exists an earlier action o'' whose partial justification $\gamma(o'')\theta$ can be extended by o, resulting in a satisfiable instantiation. A partial justification $c\theta$ up to o_j of a chaining constraint $c = o_1 \rightsquigarrow \ldots \rightsquigarrow o_n, n > j$ can be extended by an action o if o has the same action type as o_{j+1}. We denote the extension of a constraint c by an action o as $c \circ o$.

The next step is to consider all combinations of the actions that are applicable, lines 15-21. First we see whether the particular combination we are considering is valid. A mutex constraint $c = o_1 \between \cdots \between o_n$ is satisfied by a set $O \subseteq \mathbb{O}$ of actions, denoted by $O \models c$ if $|\{o_1, \ldots, o_n\} \cap O| \leq 1$, i.e. the set $\{o_1, \ldots, o_n\}$ does not contain mutex actions. A conjunction of mutex constraints $c = \bigwedge_{i=1..n} c_i$ is satisfied by a set of actions O if this property holds for each of the conjuncts: $\forall c_i \cdot O \models c_i$. For a set of constraints $\mathcal{C}$, we say that O satisfies $\mathcal{C}$, denoted by $O \models \mathcal{C}$, if O satisfies each of the mutex constraints in $\mathcal{C}$, ignoring the chaining constraints. For sets O not violating any mutex constraints (including

the empty set $\emptyset$ and all singleton sets of actions $o \in O$), we add the actions of O to the plan, and compute earliest start times τ' for all actions using a function COMPUTESTARTTIMES. For the actions $\{o_1, \ldots, o_m\}$, these are not changed, i.e. $\tau'(o_i) = \tau(o_i)$. For the new actions $o \in O$, $\tau'(o)$ equals the earliest possible start time of o. It is possible that mutex constraints in $\mathcal{C}$ and the ordering constraints $\prec$ prevent some start times to be computed, as this would violate some constraints. In that case, $\tau' = \bot$ and we ignore the particular combination of actions O. If τ' is valid, however, we use it to construct a new plan Δ' and add it to the list of new plans plans, if its probability is greater than or equal to the threshold value σ, i.e. $\mathcal{P}(\Delta') \geq \sigma$.

Finally, we recursively count all plans $\Delta' \in \mathsf{plans}$, increasing the time t to $\tau(\mathsf{fringe})$, as this is the time that we are now considering, or to $\tau(\mathsf{fringe}) + 1$ in the case that we did not add any actions (i.e. $\Delta = \Delta'$), and we want to consider the next action.

Example 16. We apply the algorithm to the problem given in Section 3.2. We start with the initial partial plan that consists of a *load* action at time 3, and an *unload* action at time 10. As indicated the parameter t is initialised to 0, and σ is set to an arbitrary value, low enough to expand the first few plans. The initial plan does not satisfy the constraints (as the *unload* action cannot be justified), so it is not counted. The *load* action is the first action with a start time greater than t, so is selected to be the fringe. Given the constraints given in Example 7 there are three possibilities: *(i)* we include a *load* action (either directly from the $\rightarrow$ *load* constraint, or because it starts the *load* $\rightarrow$ *move* $\rightarrow$ *unload* constraint that provides a justification for the *unload* action at time 10), *(ii)* a *move* action (due to the $\rightarrow$ *move* constraint), or *(iii)* we do nothing. In the first two cases, we increase t to $\tau(load) = 3$; in the last case to $\tau(load) + 1 = 4$.

In the latter case, we next select the original *unload* action as the fringe. Again, we can add *load* and *move* actions (by the same reasoning as before). In the second case, we can justify all actions in the plan and we count it as a possibility, in the former case we cannot. The third case, where we decide *not* to add any actions, leads to a dead end, as we cannot possibly justify the *unload* action.

The process we describe here is depicted in Figure 3. This also shows how, when we

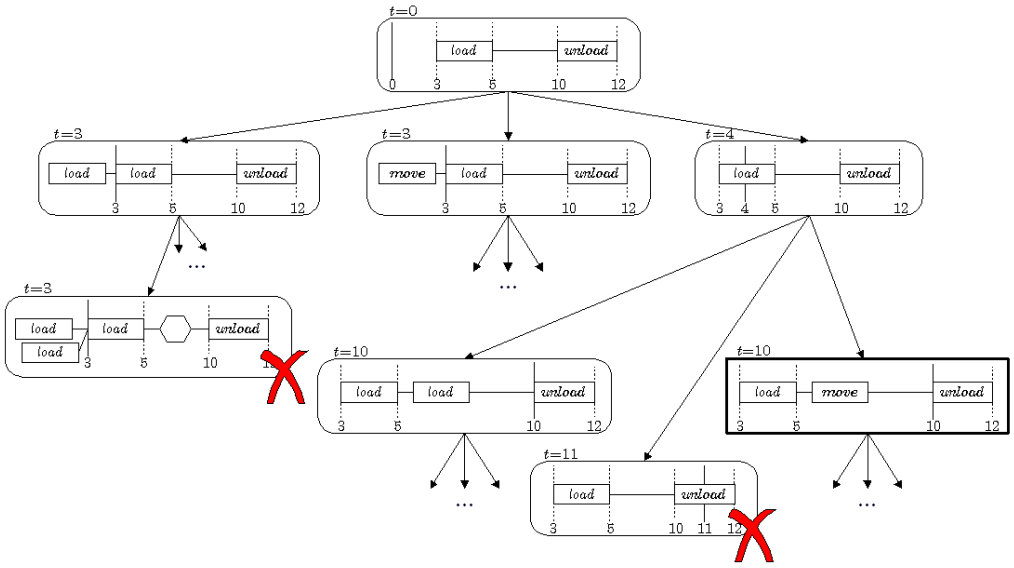

Figure 3: Part of the process of counting plans. Each (rounded) box contains a partial plan. Dashed lines show the start and finish times of the initial two actions; a solid line denotes the current value of t (which is also given at the top of each box). The bold, square box is being counted. The crossed-off boxes cannot be further expanded.

decide to add a *load* action, successive tries to add additional actions before the *load* action at time 3 end in failure as COMPUTESTARTTIMES will not be able to return a set of start times that does not violate any mutex constraints.

Remark. Notice that during the construction of plans, the constraints that "fire" (as a result of which actions are included in the plan) can be stored. This prevents justifications having to be computed during later iterations of the algorithm.

The counting of possible solutions has its own complexity class: #P [29]. In our particular case, the complexity is exponential in the number of actions. (Notice that considering the sets $O \in 2^{\mathsf{extensions}}$ instead of just the actions $o \in \mathsf{extensions}$ in line 15 is merely to cut down on the number of recursions. It does not constitute a higher degree of complexity.) In general the number of action types is far less than the number of action instantiations (that take into account all possible permutations of parameters). Hence, while exponential, it would only add a relatively minor overhead when planning.

Alice and Bob (whom we met in the introduction) *are wary of finding themselves in the*

same situation again as they did with Eve. They can use this algorithm to improve their interaction modules and prevent this to some degree as follows. Whenever the interaction modules detect an empty ride, it will compute the amount of privacy lost by announcing this empty ride. This can be computed by comparing the current amount of privacy that is lost H_{before} *(by running the* COUNT *algorithm on a partial plan that reflects the current knowledge that the other party has) by the amount* H_{after} *that would be lost when the opponent has the additional information (using a partial plan that reflects this). The cost associated with the empty ride and the privacy value are then compared to a threshold value for each agent, taking into account the information that was already shared with that agent. (Alice and Bob might be willing to incur a higher degree of privacy loss to each other than to newcomers, so the threshold values may differ from agent to agent.) If the privacy loss is less than the threshold value for some of the partners, the empty ride is announced to those partners. Upon receiving an announcement, the interaction modules decide whether any offers can be made at all. If so, then these offers are again compared to a threshold value (which may be different from the initial value) and any offers deemed rational are then put forward. In this way, privacy loss is kept to a minimum, and is only incurred if the cost incurred without sharing the information makes it worth it. This may lead to less optimal solutions, as some empty rides and offers are not announced. Such is the cost of privacy.*

6 Related Work

Multiagent Planning

As already stated in the introduction, research on multiagent planning methods has been driven by privacy concerns of the autonomous actors involved. In this lies the most important difference with the field of *distributed planning*, which is concerned with planning in a distributed environment. One of the earliest attempts at solving this problem is that of Corkill [4]. His interest stemmed from "*situations where sensory devices, processing capability, and devices to be controlled have wide spatial distributions.*" This clearly shows the difference in perspective: the focus is on a single system, that is distributed in nature. The

same view is present in an early overview article on the area [9]. Several approaches have emerged from the distributed planning research. The first is the use of *social laws*. Such laws limit the behaviour of the planners involved such that coordination problems hardly arise. Shoham and Tennenholz [24] and Briggs [2] are proponents of this approach. This requires, however, that such a separation of planners is possible. If the different planners have shared resources, for example, this is not possible. *Market theory* provides a way to resolve this. These methods rely on negotiation or auctions to coordinate the use of shared resources between agents. Zlotkin and Rosenschein [43] and Walsh and Wellman [37] are representative of research in this direction.

Methods that revolve around the plans that are built themselves have also been developed. *Plan merging* techniques assume that planners compute base plans separately, after which these plans are analysed to detect and resolve conflicts, and exploit possible positive interactions. Georgeff [12], Yang [39] (in particular Chapter 7) and Tsamardinos et al. [28] all take this approach. Yang et al. [40] and Foulser et al. [10] combine this with social laws to ensure efficient merging. Several methods have been designed based on the concept of exchanging information on (parts of) the plan. The *Partial Global Planning* approach [8] is based on the premise that agents inform other agents of parts of their plans that they think are of use to the other agent, or may lead to conflicts. The agents then build a "partial global plan" that collects the received information. From this, they can distill positive interactions that can be exploited and negative ones that have to be repaired.

The work done in multiagent planning largely follows the same lines. Privacy, however, takes a more prominent role in these works. Privacy features either explicitly, or is implicitly present as *self-interested* agents are concerned. Distributed planning techniques are all adapted to this change in situation: plan merging (for example by De Weerdt [5]), market theory (such as proposed by Sandholm and Lesser [21]) and plan exchanges (as exemplified by the Generalised Partial Global Planning approach [6] and the work by Von Martial [35]). Specialised methods have been developed as well. For example, Van der Krogt and De Weerdt [33] propose to use plan repair techniques to merge sub goals that are being exchanged. It is important to note that none of these systems explicitly mentions a measure

of privacy, although some claim to be built around the principle of privacy.

Multiagent Systems and Privacy

There are several other sub fields of multiagent systems that have looked at privacy issues. The most relevant of these for planning are the fields of of Distributed Constraint Optimisation (DCOP) and Distributed Constraint Satisfaction (DisCSPs). Recent work proposes metrics for analysis of privacy loss in such systems. Most of the work in this area focuses on distributed meeting scheduling. In this type of problems, a number of agents has to schedule a number of meetings. Each meeting requires a certain set of agents to be present, and each agent has preferences or costs attached to time slots and locations. Silaghi and Faltings [25] use a measure of privacy to drive their algorithm. Each agent has certain costs associated with the revelation of whether some tuple of values is feasible. During the exchange of messages, agents have to pay this cost if some tuple is fully determined by the other agents. Negotiations are terminated if the cost of revealing a certain tuple is greater than the potential reward for collaborating. A similar model is the basis of the work by Silaghi and Mitra [26]. However, the privacy metric here is the size of the smallest coalition necessary to deduce an agent's costs for certain tuples. Wallace and Freuder [36] consider a measure of privacy loss that is very close to ours. Their work, like ours, is based on information entropy. However, the application of information theory is more straight-forward as they consider the uncertainty of each of the variables in the constraint satisfaction problem, rather than having to apply it to an additional data structure (i.e. the plan) as we do. Recent work by Maheswaran et al. [17] proposes a general quantitative framework to analyse privacy loss in DCOP/DisCSP. The three earlier approaches can be seen as specific instances of this framework.

Beyond DCOP and DisCSP, research on privacy is undertaken in the agent community at large. This includes work on cryptographic techniques, secure auctions and randomisation (see e.g. work by Brandt [1], Van Otterloo [34] and Naor [18]). Of particular interest to planning is the work on randomisation (e.g. Paruchi et al. [19] and Van Otterloo [34]). These approaches assume that actions and behaviours can be observed. By choosing ac-

tions in a randomised fashion (e.g. using policies with a high entropy) agents can try to provide minimal information on their preferences, while still attaining their goals. Yu and Cysneiros [41] discuss the privacy aspect of the design of a multiagent system in relation to competing design aspects such as cost, performance, usability, etc. A systematic framework is proposed to support system analysts and designers in the trade-offs involved.

7 Conclusions

Although privacy is an issue that is often mentioned in work on multiagent planning (as well as multiagent approaches to different problems), heretofore this notion was neither made explicit, nor analysed. The present work shows how Shannon's Information Theory can be applied to planning to derive meaningful definitions of concepts such as uncertainty, information and privacy loss in terms of the action types that an agent's plan consists of. This definition of privacy loss allows an agent to establish how much privacy it has lost by giving out certain information. Proactively, it can be used to establish how much *would* be lost, if certain options during negotiation were acted upon. It is less useful as a tool to evaluate the worth of the information one has just learned about an opponent's plan; it requires detailed knowledge about the agent's domain.

A number of extensions to this work seem obvious. Firstly, it seems reasonable to consider more than just the type of action. Our model is limited in the sense that we do not distinguish between knowing that a *move* action takes place, and knowing the precise locations. There may be domains where such a distinction is vital and requires a more rigorous assessment then our current model allows for. We conjecture that an extension of the domain constraints that allows for parameters of actions, rather than just action types, could possibly solve this issue.

Secondly, we are interested in obtaining approximations for the number of possible plans in a given situation. Especially when we start reasoning over possible action parameters, this can become quite important, as the search space multiplies by the number of possible parameters. Solution counting algorithms for CSPs (such as presented by Pesant [20]) may

be employed for this purpose, as several efficient translations of planning into CSP exist.

Most importantly, however, this work provides the means to evaluate multiagent planning systems on their privacy impact and allows new multiagent planning systems to be designed. Whereas before, privacy was mentioned as a driving force but not explicitly taken into account, the existence of a metric for privacy loss can direct research to new algorithms that are optimised for privacy. This would also entail research into the relation between privacy loss and other factors such as optimality and search efficiency. Also, it would need to factor in trust, to establish threshold values for the different agents.

Acknowledgements

Roman van der Krogt is supported by an Irish Research Council for Science, Engineering and Technology (IRCSET) Postdoctoral Fellowship.

References

[1] F. Brandt. Fully private auctions in a constant number of rounds. In *Proc. of the* 7^{th} *Annual Conf. on Financial Cryptography (FC-03)*, pages 223–238, 2003.

[2] Will Briggs. *Modularity and Communication in Multi-Agent Planning.* PhD thesis, University of Texas at Arlington, 1996.

[3] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[4] Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, pages 168–175, San Mateo, CA, August 1979. Morgan Kaufmann Publishers.

[5] Mathijs M. de Weerdt. *Plan Merging in Multi-Agent Systems.* PhD thesis, Delft Technical University, Delft, The Netherlands, 2003.

[6] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, June 1992.

[7] Keith S. Decker and Jinjiang Li. Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3(2):113–157, 2000.

[8] Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 875–883, San Mateo, CA, 1987. Morgan Kaufmann Publishers.

[9] V.R.; Corkill D.D. Durfee, E.H.; Lesser. Trends in cooperative distributed problem solving. *Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[10] D.E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence Journal*, 57(2–3):143–182, 1992.

[11] M. Fox and D. Long. The automatic inference of state invariants in tim. *Journal of AI Research*, 9:367–421, 1998.

[12] Michael P. Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pages 125–129, Menlo Park, CA, August 1983. AAAI Press.

[13] N. Gupta and D. S. Nau. Complexity results for blocks-world planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, Menlo Park, CA, 1991. AAAI Press.

[14] Mark Iwen and Amol Dattatraya Mali. Distributed graphplan. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-02)*, pages 138–145, 2002.

[15] Subbarao Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, 6(1):16–25, 1995.

[16] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.

[17] Rajiv T. Maheswaran, Jonathan P. Pearce, Emma Bowring, Pradeep Varakantham, and Milind Tambe. Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. *Autonomous Agents and Multi-Agent Systems*, 13(1):27–60, 2006.

[18] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic Commerce*, pages 129–139, 1999.

[19] P. Paruchi, M. Tambe, D. Dine, S. Kraus, and F. Ordonez. Safety in multiagent systems via policy randomization. In *AAMAS workshop on Safety and Security in Multiagent Systems*, 2005.

[20] Gilles Pesant. Counting solutions of csps: A structural approach. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 260–265, 2005.

[21] Tuomas W. Sandholm and Victor R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[22] Thomas D. Schneider. Information theory primer, v2.60. `http://www.lecb.ncifcrf.gov/~toms/paper/primer`, 2007.

[23] C.E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423,623–656, 1948.

[24] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1–2):231–252, 1995.

[25] M.C. Silaghi and B. Faltings. A comparison of distributed constraint satisfaction approaches with respect to privacy. In *Proceedings of the Third workshop on Distributed Constraints Reasoning (DCR-02)*, pages 147–155, 2002.

[26] M.C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *Proc. of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-04)*, pages 531–535, 2004.

[27] M. Tribus. *Thermostatics and thermodynamics.* D. van Nostrand Company, Inc., Princeton, NJ, 1961.

[28] I. Tsamardinos, M. E. Pollack, and J. F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pages 264–272, Menlo Park, CA, April 2000. AAAI Press.

[29] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[30] Menkes van den Briel, Romeo Sanchez Nigenda, Minh Binh Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 562–569, 2004.

[31] Roman van der Krogt. Privacy in multiagent planning: A classical definition with illustration. In *Proc. AAMAS '07 Workshop on Coordinating Agents' Plans and Schedules*, 2007.

[32] Roman van der Krogt. Privacy loss in classical multiagent planning. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-07)*, 2007.

[33] Roman van der Krogt and Mathijs de Weerdt. Coordination through plan repair. In *MICAI 2005: Advances in Artificial Intelligence*, volume 3789 of *Lecture Notes on Artificial Intelligence*, pages 264–274. Springer, 2005.

[34] S. van Otterloo. The value of privacy: optimal strategies for privacy minded agents. In *Proc. of the Fourth Int. Conf. on Autonomous Agents and Multi- Agent Systems (AAMAS-05)*, pages 1015–1022, 2005.

[35] Frank von Martial. *Coordinating Plans of Autonomous Agents*, volume 610 of *Lecture Notes on Artificial Intelligence*. Springer Verlag, Berlin, 1992.

[36] Richard J. Wallace and Eugene C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161:209–227, 2005.

[37] William E. Walsh and Michael P. Wellman. A market protocol for decentralized task allocation and scheduling with hierarchical dependencies. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 325–332, 1999.

[38] Daniel S. Weld. An introduction to least-commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[39] Qiang Yang. *Intelligent planning: a decomposition and abstraction based approach*. Springer-Verlag, London, UK, 1997.

[40] Qiang Yang, Dana S. Nau, and James Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(4):648–676, November 1992.

[41] E. Yu and L. Cysneiros. Designing for privacy and other competing requirements. In *Proceedings of the 2nd Symposium on Requirements Engineering for Information Security (SREIS-02)*, 2002.

[42] Jian Feng Zhang, Xuan Thang Nguyen, and Ryszard Kowalczyk. Graph-based multi-agent replanning algorithm. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, 2007.

[43] Gilad Zlotkin and Jeffrey S. Rosenschein. Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5:163–238, 1996.

Author's Bio

Roman van der Krogt received his Ph.D. from Delft University of Technology, on plan repair for multiagent systems. After a short period as a post-doctoral researcher at that university,

he moved to the Cork Constraint Computation Centre (4C), where he is currently employed as a research scientist. During his time at 4C, he has worked on diverse topics such as scheduling for manufacturing (with Bausch & Lomb, Alcatel-Lucent and Intel); multiagent planning; combining AI planning with constraint-based reasoning; and combining optimisation and simulation. He was awarded an IRCSET Fellowship for his work on multiagent planning.