Measuring Plan Coverage and Overlap for Agent Reasoning

John Thangarajah
RMIT University
Melbourne, Australia
john.thangarajah@rmit.edu.au

Sebastian Sardina
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Lin Padgham
RMIT University
Melbourne, Australia
lin.padgham@rmit.edu.au

ABSTRACT

In Belief Desire Intention (BDI) agent systems it is usual for goals to have a number of plans that are possible ways of achieving the goal, applicable in different situations, usually captured by a *context condition*. In Agent Oriented Software Engineering it has been suggested that a designer should be conscious of whether a goal has *complete coverage*, that is, is there some plan that is applicable for every situation. Similarly a designer should be conscious of *overlap*, that is, for a given goal, are there situations where more than one plan could be applicable for achieving that goal. In this paper we further develop these notions in two ways, and then describe how they can be used both in agent reasoning and agent system development. Firstly we replace the boolean value for basic coverage and overlap with numerical measures, and explain how these may be calculated. Secondly we describe a measure that combines these basic measures, with the characteristics of the coverage/overlap in the goal-plan tree below a given goal. We then describe how these domain independent measures can be used for both plan selection and intention selection, as well as for guidance in agent system development.

Categories and Subject Descriptors

I Computing Methodologies [**I.2 Artificial Intelligence**]: I.2.11 Distributed Artificial Intelligence—*Intelligent Agents*

General Terms

Algorithms, Measurement, Design

Keywords

Agent reasoning, intention selection, coverage, overlap, goals, plans

1. INTRODUCTION

In this paper we explore and refine the notions of *coverage* and *overlap* as used in Agent Oriented Software Engineering [8] and describe how they can be used for domain independent agent reasoning. The primary motivation for this work was the search for general purpose characteristics that could sensibly be used to guide intention selection in a BDI (Belief, Desire, Intention) agent system. Typically an agent may well be pursuing multiple goals, and at any point in time it must decide which intention it will progress in the next step. Importantly, the approach we take allows all complex computations to be done at compile time, thus retaining the important soft real-time aspect of agent systems.

Existing agent platforms such as JACK^{TM}[1] offer a choice between *round robin* which does a fixed number of steps on each intention in turn, or *FIFO* which basically uses a queue, finishing one intention and then moving to the next. Some systems such as JAM [6] also allow a priority or utility on goals and/or plans which can be used to order the intention queue. AgentSpeak(XL) [1], AgentSpeak(RT) [13] and TAEMS [5] all use time related information to prioritise the scheduling of intentions, while [15] explores dynamic changes to the priority depending on temporal information. However many standard agent programming languages do not include temporal information - neither when a goal should be achieved by, nor how long a plan or goal can be expected to take. Utility information is also typically unavailable, and requiring a programmer to provide this can be problematic. To our knowledge there is no work which provides any non-temporal general purpose heuristic for intention selection, which does not require user provided priorities or utilities. Agent programming language definitions such as AgentSpeak [9] or CANPLAN [10] typically define only how to step individual intentions and remain agnostic about how to select the particular intention to be progressed.

The intuition behind this piece of work is that (all else being equal) if we have a number of intentions which we are able to progress at any time point, we will prefer to progress the one which we believe has fewest possible successful executions. That is, the one most vulnerable to becoming unable to be successfully executed due to decisions taken in executing other intentions. Preferring such intentions ensures that choices made in pursuing an alternative intention do not eliminate the relatively fewer available ways to achieve such an intention. In order to realise this intuition we use the concept of coverage. In the Agent Oriented Software Engineering context, the Prometheus methodology [8] encourages developers to specify whether a goal has full coverage or not by considering whether for any situation, there can be no applicable plan. In this work we specify coverage as a fraction of the space of all possible models – that portion of the state space for which there is some applicable plan. We calculate this fraction using model counting to ascertain firstly the total number of models in the domain of concern; and secondly the number of models in which a single plan and a set of plans are applicable in using the context conditions of the relevant plans.

As we will discuss in section 3, this *Basic Coverage* must be further refined to give a *Coverage Measure* based on the hierarchy below the goal as well as the immediate relevant plans. As plans are selected, and paths followed, it may be the case that additional

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

[1]`http://aosgrp.com/products/jack/`

constraints are introduced in sub-plans which in fact means that the applicability space indicated by the context of a plan is reduced further. For example (see figure 2), consider a goal to Travel, with 3 plans: WalkPlan, TramPlan and FlyPlan. Walk has the context condition *distance is short and weather is fine*; Tram has the context condition *distance is short and weather is not fine*; Fly has the context condition *distance is long*. Now assume the TramPlan has a sub-goal GetTimetable which has 3 possible plans: one with context that it is *a weekday and not a public holiday*, one with context that it is *a weekend but not Christmas or Good Friday*, and another that it is *a public holiday but not Christmas or Good Friday*. Here we see that once plans are decomposed there is no decomposition possible for the case of short distance, not-fine weather and Christmas day. Thus the apparent full coverage at the top level goal is compromised. Our Coverage Measure modifies the Basic Coverage to account for such compromises in the tree below.

In section 4, we define a measure of overlap to capture the intuition that in the case where coverage is identical, we are more likely to succeed if a larger part of the state space has alternative plans available in the event that the chosen plan fails (perhaps due to stochastic causes, or maybe because of environmental changes outside of the agent's control).

In section 5, we discuss how these measures can be used for several reasoning tasks, including the intention selection which was the original motivation for the work. The more refined approach to coverage can also enable analysis of agent software to identify and alert the developer to places where the apparent Basic Coverage is substantially reduced once the Coverage Measure which takes into account the hierarchy below is considered. Plan selection is another task which can utilise these measures.

Identification and careful specification of domain independent characteristics can provide the basis for additional power in the next generation of agent systems. Coverage and overlap are important such characteristics and this work provides the foundations for understanding both how to compute them and how they can be used.

2. MEASURING BASIC COVERAGE

We assume the basic standard plan rules typical for agents in the BDI paradigm where $G : \psi \leftarrow P$, meaning that *plan P is a reasonable plan for achieving goal G when (context) condition ψ is believed true*. Though different BDI languages offer different constructs for crafting plans, most allow for sequences of domain actions that are meant to be directly executed in the world (e.g., lifting an aircraft's flaps), and the posting of (intermediate) *sub-goals* $!G'$ (e.g., obtain landing permission) to be resolved. Sub-goals posted during the execution of a plan are resolved via the plan library. When a plan is selected for realising the achievement of a particular goal, it is placed into the *intention base*[2] for execution. The execution of a BDI system can be seen then as a *context sensitive sub-goal expansion*, allowing agents to "act as they go" by making *plan choices* at each level of abstraction with respect to the current situation.

There are then two key decisions an intelligent BDI agent is required to make on an ongoing basis. The first is, which intention from the intention base to progress at each execution cycle. The second is, given a pending goal to be addressed (either completely new or arising from an existing intention), which plan among the available ones in the library to select for execution.

In deciding which plan to select to address a given goal, a BDI system relies on the *context condition* of plans. The context condition ψ of a plan $G : \psi \leftarrow P$ encodes the domain knowledge of when procedure P is an adequate approach to address G. Context conditions are generally formulae in some logical language. For simplicity, at this point, we consider a BDI agent system that is programmed relative to some finite propositional language $\mathcal{P}$.

Given a goal G, let $Pl(G)$ be the set $\{P_1, \dots, P_n\}$ of alternative plans for achieving G, the so-called *relevant* plans for G. The first thing we are interested in is to know, and compute the coverage of each relevant plan (and even of the goal G itself), that is, in how many world situations an agent will be able to use a plan (or resolve a goal). To make this notion concrete, we recast the coverage problem as that of *model counting* (or #SAT) problem [4], that is, the problem of computing the number of models for a given (propositional) formula – the number of distinct truth assignments to variables for which the formula evaluates to true. As standard, for a propositional formula φ, we will use $\#\varphi$ to denote the model count of φ. The model counting problem generalizes SAT and is the canonical $\#P$-complete problem.

Coverage:
So, given a plan-rule $G\!:\!\psi \leftarrow P$, we define

$$\mathcal{A}(P) = \#\psi$$

to be the number of models in which P is applicable (when no ambiguity arises, we use the plan-body P to refer to the whole plan-rule and ψ_P to refer to the corresponding context condition). For example, for plan P_1 in Figure 1, $\mathcal{A}(P)$ denotes the area $a{+}e$. When $\mathcal{P}$ is the language in which context conditions are written, we use $S_T = 2^{\mathcal{P}}$ to denote the set of all possible worlds (i.e., models), in the domain of concern. Hence, a single plan as above has a *Basic Coverage*:

$$C(P) = \frac{\mathcal{A}(P)}{S_T}$$

Note that $0 \leq C(.) \leq 1$.

The Basic Coverage can be generalized to a set of plans S in a straightforward manner as follows (recall ψ_X refers to the context-condition of plan X):

$$\mathcal{A}(S) = \#(\bigvee_{P \in S} \psi_P).$$

Considering Figure 1, $\mathcal{A}(\{P_1, P_2, P_3\})$ denotes the areas $a + c + d + e + f + g + h$. As before, the Basic Coverage of a set of plans S is defined as:

$$C(S) = \frac{\mathcal{A}(S)}{S_T}$$

Overlap:
Besides the coverage, we are also interested in overlap, which in terms of Agent Oriented Software Engineering is when there are two or more plans applicable in the same situation to achieve a particular goal. For example, in Figure 1 the region 'e' is the state space covered by both plan P1 and P4. The greater the overlap, the greater the chance of another plan being applicable in the event that one fails, thus providing flexibility and robustness to the whole system. The *Basic Overlap* of a set of plans can be easily defined as:

$$O(\{P_1, \dots, P_n\}) = \#(\phi_1 \wedge \dots \wedge \phi_n)$$

It is not difficult to see that all above definitions for coverage and overlap can be computed using model counting solvers. Because the reasoning we are interested in will be done offline and not at BDI execution time, one could make use of exact counting

[2]we describe the structure of the intentions further in section 5.

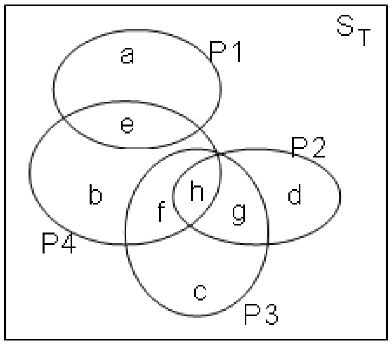

Figure 1: Illustration of state coverage and overlap of 4 alternate plans for achieving a goal.

algorithms, either DPLL-style exhaustive search or those based on "knowledge compilation" [4]. In addition, we observe that, unlike in many SAT/#SAT applications, context conditions in a BDI application are not of a combinatoric nature, and hence we expect exact model counting algorithms to perform well enough for offline preprocessing. Nonetheless, if time is an issue, one could always rely on available approximate counting techniques that provide fast estimates. Some of those techniques come with no guarantees (e.g., ApproxCount [14]), while some methods provide lower or upper bounds with a correctness guarantee, often in a probabilistic or statistical sense (e.g., SampleCount [3]).

We close this section by noting that what we have done so far is to define the coverage and overlap of BDI plans as a model counting problem at the *basic* level. That is, we have not considered the fact that BDI plans can, and most often will, have sub-goals. This calls for a more adequate measure that considers the plans relevant for those sub-goals and that is defined in terms of the whole goal-plan hierarchy implicit in every BDI plan library. This is the subject of the next section.

3. COVERAGE MEASURE OF GOAL-PLAN HIERARCHIES

In this section, we describe the algorithms for calculating a measure of coverage for goals (and plans) considering the goal-plan hierarchies. By goal-plan hierarchy we refer to the implicit goal-plan structure that is present in the plan library of BDI systems – goals are achieved by a set of alternate plans, and each plan contains sub-goals or actions,[3] where each sub-goal in turn is handled by a set of plans. Figure 2 illustrates an example of a goal-plan tree structure based on our motivating example from Section 1. We note we restrict our attention to plan libraries having no cycles.

We want to calculate a *Coverage Measure* for goals which modifies the Basic Coverage defined in the previous section based on the goal-plan hierarchy beneath each plan of the goal. As mentioned previously, an apparently high Basic Coverage may be compromised by lower coverage in the underlying tree. We take this into account when determining the Coverage Measure of a goal as follows (we use the diagram illustrated in Figure 1).

Exclusive Coverage and Exclusive Overlap.

Firstly we define two notions to allow us to speak about the number of models in each of the types of region a–h in Figure 1. We define the notion of *Exclusive Coverage* ($EC(G, P)$) of a plan, P, with respect to a goal, G, to capture the number of models in which P is exclusively applicable (i.e. ψ_P is exclusively true), as defined as in Equation 1. For example, regions 'a', 'b', 'c' and 'd' in Figure 1.

We define *Exclusive Overlap*($EO(G, S)$) in Equation 2 to capture the number of models in the overlapping area of all plans in some group S with respect to a goal G. Note that the exclusive overlap for a group of plans refers to the number of models in the region that is exclusively covered by *all* of the plans in that group only and no other plan. For example, for the group {P2,P3} the exclusive overlap region is 'g' and for the group {P2,P3,P4} it is 'h'.

Recall from the previous section that $\mathcal{A}(P)$ is the number of models in which the plan P is applicable (similarly, $\mathcal{A}(\{P_1, \ldots, P_n\})$ for a set of plans), and that S_T is the total number of models in the domain of concern.

$$EC(G, P)^4 = \frac{\mathcal{A}(Pl(G)) - \mathcal{A}(Pl(G) \setminus \{P\})}{S_T} \tag{1}$$

$$EO(G, S)^5 = \frac{\mathcal{A}(Pl(G))) - \mathcal{A}(Pl(G) \setminus S) - \sum_{P \in S} EC(G, P)}{S_T} \tag{2}$$

Note that if a plan has no overlap with the other plans, then its basic and exclusive coverage measures coincide.

Coverage Measure for Goals and Plans.

We now wish to define a *Coverage Measure* for a goal by summing these separate regions, appropriately discounted with regard to the Coverage Measure of the underlying tree.

The exclusive coverage areas are discounted by the Coverage Measure of the relevant plan (which captures the Coverage Measure of its sub-goals). The exclusive overlap areas (regions 'e','f','g' and 'h') are discounted by the Coverage Measure of the plan with the highest Coverage Measure of that group. This is because when there is an overlap between plans all of them are applicable for that space and the agent would always choose the plan with the highest Coverage Measure (i.e. highest chance of success). The Coverage Measure of a goal G, denoted $C_M(G)$, is therefore the addition of the Coverage Measures of the exclusive coverage regions of each plan of G and the Coverage Measures of the exclusive overlap regions of each grouping of plans, with the appropriate discount factors described above. This is defined as follows:

$$\begin{aligned} C_M(G) = & \\ & \sum_{P \in Pl(G)} EC(G, P) \times C_M(P) \; + \\ & \sum_{S \in \mathcal{P}_{\geq 2} Pl(G)} EO(G, S) \times \max(\{C_M(P) \mid P \in S\}). \end{aligned} \tag{3}$$

The Coverage Measure of a plan P, denoted $C_M(P)$ and shown in Equation 4, is defined as the product of the Coverage Measures of its sub-goals, where $Sg(P)$ is the set of sub-goals within the body of plan P. We take the product since for a plan to succeed all the sub-goals must be achieved, and we assume that the success (with respect to coverage) of each sub-goal is independent of each other's success. Using probability theory, the probability of two independent events occurring together (in parallel or in sequence) is the

[3]Messages to other agents are regarded as actions for the purpose of this paper.

[4]In terms of model counting, as presented in Section 2, $EC(G, P)$ can be expressed as $\#(\psi_P \wedge \bigwedge_{P' \in Pl(G)\setminus\{P\}} \neg\psi_{P'})$.

[5]In terms of model counting $EO(G, S)$ can be expressed as $\#(\bigwedge_{P \in \Pi} \psi_P \wedge \bigwedge_{P' \in Pl(G)\setminus\Pi} \neg\psi_{P'})$.

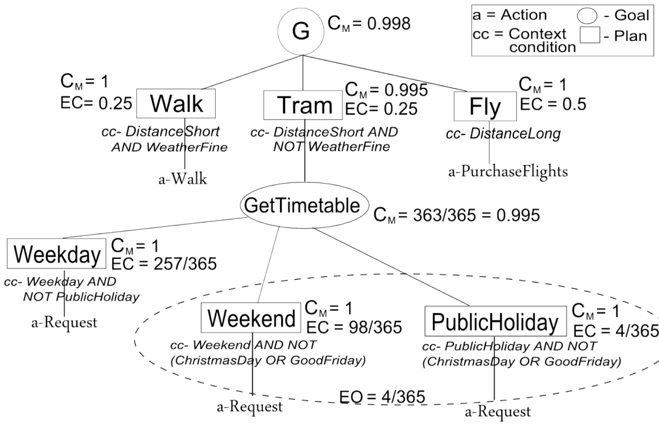

Figure 2: Example coverage calculation for a goal. [7]

product of the individual probabilities of each event. A plan with no sub-goals (that is, a leaf plan in a goal-plan tree with only actions) has a Coverage Measure of 1.

$$C_M(P) = \begin{cases} 1 & \text{if } Sg(P) = \{\} \\ \displaystyle\prod_{G \in Sg(P)} C_M(G) & \text{otherwise} \end{cases} \tag{4}$$

In the above Coverage Measure for a plan, except for the case where the sub-goals have a Coverage Measure of 1, the more sub-goals a plan has the lower the Coverage Measure of that plan. As stated above, this is due to the fact that the more the agent has to achieve the lesser the chance of success. This highlights the importance that as plans are modularized into sub-goals[6], it is important to consider the individual coverage of each of the sub-goals striving to achieve full coverage whenever possible.

Figure 2 illustrates the above calculations applied to the travel goal example introduced in Section 1. As mentioned then, if we only considered the coverage of the immediate plans of the top level goal, the goal would have full coverage (0.25 + 0.25 + 0.5 = 1). However, due to the incomplete coverage at the lower levels (there is no 'Tram' plan decomposition for ChristmasDay or GoodFriday) this value is reduced when Coverage Measures are calculated.

4. OVERLAP MEASURE OF GOAL-PLAN HIERARCHIES

We now consider how an *Overlap Measure* should be calculated, using a similar notion of influence from the underlying goal-plan tree, as with our Coverage Measure. We recall that overlap is useful in the case of plan failure, so an alternative plan can be tried in the event that one fails. Hence, overlap can be seen as a measure related to likely success of the goal with respect to failure recovery.

Recall from Section 2 that the overlap of a goal is the overlap between the plans of the goal. That is, the models in which more than one of the plans are applicable in. We recall also that Exclusive Overlap of a set of plans relative to a particular goal ($EO(G, S)$) is defined in Equation 2.

In order now to calculate our Overlap Measure for a goal, taking account of the underlying tree, we will:

1. Firstly, sum the Exclusive Overlap count of the individual regions in all combinations of plans in $Pl(G)$, relative to $\mathcal{A}(Pl(G))$[8], multiplied by the average of the Coverage Measure of the plans involved. We discount using the Coverage Measure, as the overlap is only of value to the extent that there is coverage, and we consider the average Coverage Measure of the group of plans and not the maximum Coverage Measure, since overlap is beneficial in the event of plan failure hence the success of all the plans needs to be considered.

2. Secondly, in order to capture the amount of overlap in the tree below each plan of the goal, we add the sum of the Overlap Measures of the plans of the goal ($Pl(G)$). The Overlap Measure of each plan (Equation 6) is determined by the sum of the Overlap Measures of the sub-goals of the plan (if any).

This is defined as follows:

$$O_M(G) = \sum_{S \in \mathcal{P}_{\geq 2} Pl(G)} \frac{EO(G,S)}{S_T} \times AVG\Big(\{C_M(P) \mid P \in S\}\Big) + \sum_{P \in Pl(G)} O_M(P) \tag{5}$$

Equation 6, defines the Overlap Measure of a plan. Unlike in coverage, where the coverage measure of sub-goals discount the coverage measure of the plan, in overlap we are concerned with the total overlapping spaces in all the sub-goals (that is, the tree beneath the plan) to provide a measure related to success in the event of failure.

$$O_M(P) = \sum_{G \in Sg(P)} O_M(G) \tag{6}$$

We note that whereas C_M is always between 0 and 1, the Overlap Measure is ≥ 0.

In the above Overlap Measure we do not distinguish between the number of plans that overlap a particular region of the state space (for example, in Figure 1 region 'e' is overlapped by 2 plans, and region 'h' is overlapped by 3 plans). Although having more plans overlapping the same space may seem better in terms of recovering from multiple failures, this depends on how much failure is expected and it is not clear that it makes sense to consider such a level of detail. However, if desired or considered important for a particular application, Equation 5 can be modified to weight the Overlap Measure of each exclusive overlap region proportionally to the number of plans in the overlap, by replacing term $EO(G, S)$ with $|S| \times EO(G, S)$.

We note that a property of our definition of the Overlap Measure is that, under the assumption of complete coverage for each goal, it does not matter how the overlap is distributed within the tree; nor is the Overlap Measure affected by the form of the goal-plan tree (i.e. depth or breadth). For example, in Figure 3 we assume that the plans P3 and P4 overlap 30% of the relevant space for G2, but have no sub-goals. In the tree under G1 this same overall amount of overlap is distributed with 10% overlap between P1 and P2, and

[6]This is often the case when developing agent systems.

[7]Note that this is not a complete example and is a simplified version for illustration and clarity of the Coverage Measure calculations. Similarly, we have made assumptions about the number of public holidays in the year and the overlap of public holidays with weekends and weekdays.

[8]Note that we are concerned with a measure relative to the models covered by (the plans of) the goal, not relative to S_T as in coverage.

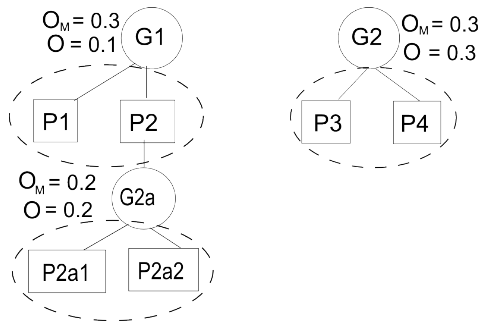

Figure 3: Example of possible distribution of overlap.

20% overlap between P2a1 and P2a2. Applying our formulae[9] we see that $O_M(G1)$ is equal to $O_M(G2)$.

If we assume constant, but less than full coverage then the Overlap Measure will be affected by the shape of the tree, and the distribution of the overlap within that tree, in a similar way to how the Coverage Measure is affected. With trees such as are shown in Figure 3 an overlap of say 40% in the tree of G2 will give a greater Overlap Measure than the same size overlap at G2a, when the Overlap Measure is propagated up to G1.

5. COVERAGE & OVERLAP FOR AGENT REASONING

In this section we discuss the usage of the Coverage and Overlap Measures that we have introduced in the previous section in agent reasoning. In particular, how they may be used for plan selection, intention selection and in agent design and development.

In order to perform some of the above reasoning, the goal-plan structures for each top level goal and the Basic Coverage counts (as defined in Section 2) can be constructed at compile time. (For details on constructing goal-plan trees with annotations see [11].)

5.1 Plan Selection

The most straightforward use of the Coverage and Overlap Measures at execution time would be to select between a set of applicable plans, that is, when there is more than one plan applicable in the current state to achieve a goal. Intuitively, we would prefer to choose the plan with the highest Coverage Measure, and if Coverage Measures were equal, then the one with a higher Overlap Measure. We capture this below.

Let $App(G)$ be set of applicable plans of G ($App(G) \subseteq Pl(G)$) and $Pref(P)$ be the preference of plan P. The $Pref$ partial ordering of plans within an applicable plan set is then defined by the following rules.

1. $\forall_{P, P' \in App(G)} \; C_M(P) > C_M(P') \implies Pref(P) > Pref(P');$

2. $\forall_{P, P' \in App(G)} \; C_M(P) = C_M(P') \wedge O_M(P) > O_M(P') \implies Pref(P) > Pref(P').$

These can readily be incorporated into a plan selection rule of an agent language such as that of CAN [10].

[9]We note that the Coverage Measure of all the plans will always be 1, because leaf plans are 1, and under the assumption of full coverage at each goal, there is no discounting as one goes up the tree.

5.2 Intention Selection

Intention selection is the issue of, if an agent has a number of intentions (instantiated plan structures for achieving high level goals) that are active, how should these be interleaved, and in particular which one should be progressed at the next step. Programming languages such as AgentSpeak [9] typically define when an intention is in a state that it can be progressed, along with how to progress it, but do not define how to select between multiple intentions. As one intention is progressed and its goals realised, it is of course possible that things are changed in such a way that other intentions are unable to be successfully realised.

Current implementations of BDI agents typically use one of several defaults in progressing intentions. One method ("FIFO") is to simply place intentions in a queue, and execute each in turn - though moving to the next if one becomes idle for some reason. Another approach is what is known as "round robin" where each intention in the queue is progressed a fixed number of steps before moving onto the next. An additional option is to (somehow) assign a priority or utility to intentions and order the queue according to that "priority". However, using priorities or utilities typically requires substantial information to be provided by the developer, which is generally onerous for anything more than a simple priority on high level goals. To our knowledge there is no principled mechanism for determining this priority, that does not rely on either temporal information, or programmer provided utilities.

We explain below how our Coverage Measure can be used to select which intention to work on next, when the current intention either finishes or becomes *unprogressable*.

When a goal is adopted as an intention, an instance of its goal-plan tree is created and placed into the set of intentions (Γ) that the agent wishes to accomplish. We refer to this instantiated goal-plan tree as the *execution-tree* ($ExTree$) of that goal. As an intention (that is, the adopted goal) is progressed[10] the nodes of its execution-tree are annotated as follows:

- When a plan is selected.
- When a plan completes. That is, when all its sub-goals and actions complete.
- When a goal completes. That is, when at least one of its plans completes.
- When a plan fails. That is, for an abstract plan when one of its actions fail, and for a concrete plan when one of its sub-goals fail.[11]

Figure 4 shows an example of a goal-plan tree (a) and a corresponding execution-tree (b) that is partially executed with the above annotations.

We define the function $next(I)$, where $I = ExTree(G)$, to return the next step in the execution-tree of the adopted goal G. For example, in Figure 4(b) the $next(I)$ function would return the action a_4.

The intention structure Γ is then a set of execution-trees of the corresponding top level goals, *each execution-tree representing an intention*. An intention is said to be *progressable* if the next step of the execution is either an action or a sub-goal for which there is at least one applicable plan in the current state.

[10]or executed.

[11]In the interest of space we do not go into a definition of the various reasons a goal could fail. For a fully usable language this requires language constructs which are outside the scope of this paper, such as tests on beliefs, which in turn require updates of beliefs, etc.

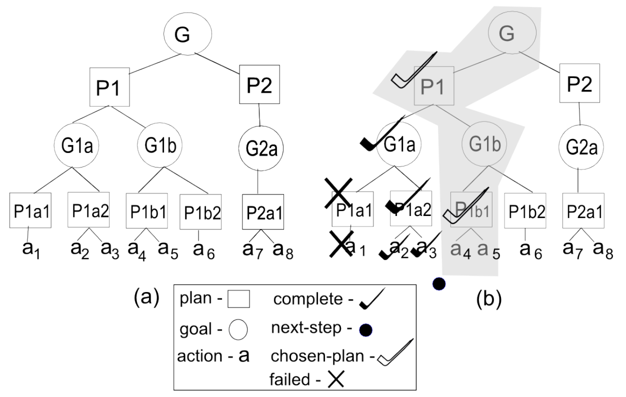

Figure 4: Illustration of goal-plan tree and execution tree.

When an intention I is *progressed* this involves processing the *next*(I) in the execution-tree. If that item is an action, it is simply executed. If it is a goal, then a plan is selected from the applicable plan set for that goal and the *next*(I) becomes the first sub-goal or action of that plan. It is here that when more than one plan is applicable the coverage based plan selection mechanism described earlier in this section may be used.

An intention is said to be *finished* when the root goal of the execution-tree completes as described above, or fails.[12] An intention is said to be *unprogressable* if it is in a waiting state. This can be due to a wait for a message response, or some action executing externally that needs to complete before proceeding. It may also include such things as an intention being blocked [10] or suspended for some reason [12].

Once an agent has selected an intention for execution we assume it will continue to be progressed until it is either finished, or becomes unprogressable. When this happens, a new intention must be chosen to start executing. It is here that we will use our Coverage Measures.

The intention selection question is then, when the current intention finishes or becomes unprogressable, which progressable intention should be made current - or which progressable intention has the highest priority. Our intuition is that we will prefer to prioritise things with least coverage, as these are the ones that have fewer possibilities for success. The Coverage Measure for an intention that has not yet started to execute is simply the Coverage Measure of the goal, relevant to that intention, as described in Section 3. However, we also need a Coverage Measure for a partially executed intention, that is an execution-tree that has been progressed, which we obtain by building on our previous definitions as follows:

1. The Coverage Measure of a goal, if a plan has been selected, is the Basic Coverage of the chosen plan ($C(P)$), multiplied by the Coverage Measure of that plan ($C_M(P)$).

$$C_M(G) = \Big(C(P) \bullet C_M(P)\Big) \qquad \text{if } \exists P : \text{chosen-plan}(G, P) \quad (7)$$

2. The Coverage Measure of a (possibly partially executed) plan is the product of the Coverage Measures of the goals still to be achieved of that plan.

$$C_M(P) = \begin{cases} 1 & \text{if } Sg(P) = \{\} \\ \prod_{g \in Sg(P) \wedge \text{not complete(g)}} C_M(g) & \text{otherwise.} \end{cases} \quad (8)$$

This Coverage Measure for partially executed intentions (Equation 7) captures our strong preference, once we have made a plan selection, to succeed that plan without failure or backtracking. Thus we no longer consider the coverage of alternative plans in measuring the coverage of a goal, where a plan selection has been made. It also captures the fact (in Equation 8) that coverage of sub-goals which have already succeeded is irrelevant. For example, in Figure 4(b), the shaded region indicates the part of the goal-plan which is considered when calculating the Coverage Measure of the partially executed intention.

The following rules now provide a priority ordering on progressable intentions which can be applied as needed, but in particular for choosing a new current intention when an intention finishes or becomes unprogressable.

Recall Γ is the set of intentions (execution-trees) and that the next step of an intention (progressed in the corresponding execution-tree) is either an action or sub-goal. So, let

- $Act(next(I))$ be true when the next step of intention I is an action;
- $Sg(next(I))$ be true when the next step of intention I is a sub-goal;
- $Current(I)$ be true when I is the current intention being progressed;
- $G(I)$ be the top level goal of the intention I, and $Pr(I)$ be the priority of intention I.

The relative priority ordering of intentions are established by the following rules applied in the oder specified below:

1. $\forall_{I, I' \in \Gamma}\, Current(I) \implies Pr(I) > Pr(I')$.
2. $\forall_{I, I' \in \Gamma}\, Act(next(I)) \wedge Sg(next(I')) \implies Pr(I) > Pr(I')$.
3. $\forall_{I, I' \in \Gamma}\, C_M(G(I)) > C_M(G(I')) \implies Pr(I) > Pr(I')$.
4. $\forall_{I, I' \in \Gamma}\, O_M(G(I)) > O_M(G(I')) \implies Pr(I) > Pr(I')$.

Our first priority is then to maintain focus as long as is possible while the second is to execute any actions that are pending, as it makes no sense to keep decomposing plans without executing the actions in as timely manner as possible. The third priority then captures the intuition that if one has an intention that has relatively fewer spaces/models in which it can be progressed through to completion, then we prefer to progress it when we have the opportunity, as opposed to an intention which has a higher Coverage Measure, representing a larger number of models incorporating successful completion. Finally, if other things are equal, we prioritise doing first the one that has a smaller Overlap Measure - i.e. the one with fewer options for recovery.

5.3 Agent Design and Development

When specifying events (goals) within an agent design, using the Prometheus methodology [8] and the supporting Prometheus Design Tool (PDT) [7] developers specify (amongst other things) the goal-plan trees for each agent. During this process they are

[12]As with goal failure, due to space limitations and the focus of this paper, we do not go into a precise definition of failure of an intention.

prompted to consider the coverage and overlap properties of the goal. It is suggested developers note if there is overlap, on what basis one of the overlapping plans will be chosen, and if there is not full coverage, which are the situations where there will be no applicable plan. This is because both lack of full coverage, and unintended overlap are common causes of bugs in implementations of BDI agents systems. In this section we identify areas in which the the coverage and Overlap Measures we have defined may be beneficial.

Coverage Measures to identify potential flaws in the design: With the new measures defined in this paper it is now possible to alert developers to cases where lack of coverage elsewhere in the goal-plan tree compromises full (or a partial high level) coverage at a top level goal. Areas where there is a significant difference between the Basic Coverage of a goal, and the Coverage Measure would be candidates for further investigation.

Overlap as a measure of robustness of goals: Overlap, as discussed, is related to the potential to recover from failure during execution, selecting an alternative plan to try to achieve some failed sub-goal. Once overlap figures are obtained, at compile time, it becomes possible to report which goals have a relatively low Basic Overlap and/or Overlap Measure. Alternative overlapping plans are one way of making a system robust to stochastic failure. While an important high level goal may have a high Basic Overlap to support such failure recovery, a low Overlap Measure may indicate potential for increasing robustness in the tree below.

Overlap for debugging: Overlap measures can also be useful in potential debugging. As noted previously, the reason for prompting developers to consider overlap is partly because unintended overlap is a common cause of error. If a system is failing at some top level goal, a non-zero Overlap Measure may indicate that the tree below that goal is a potential place to examine for error.

5.3.1 *Abstract Coverage Measures*

As mentioned above during the *detailed design* phase of developing agent systems, the developer specifies the goal-plan tree and for each goal indicate whether full coverage and overlap is expected. However, currently these attributes are not used for any automated reasoning during design, although they are used for testing and debugging. The coverage and overlap measures as we have defined them in this paper require at least a precise specification of context conditions, along with the domain and range of all variables used, which is not necessarily available at design time.

However, the Coverage Measure that we have defined can be abstracted to give some useful information at design time, using only boolean values (`True, False` as currently provided by the developer) for initial measures, and three values (`True, False, Uncertain`) for calculated measures. These measures can then indicate places where full Basic Coverage is compromised by the tree below and can call the designer's attention to possible areas for further examination prior to any implementation. We describe this abstraction below.

In order to specify the rules for this more Abstract Coverage Measure ($C_M^A(.)$) we first define an ordering over the three values we will use:

`True > Uncertain > False`

The rules are then as follows:

1. The Abstract Basic Coverage of a goal is `True`, for full coverage or `False` otherwise.

$$C^A(G) = \begin{cases} \texttt{True} & \text{if there is full coverage indicated} \\ \texttt{False} & \text{otherwise} \end{cases} \tag{9}$$

2. The Abstract Coverage Measure of an plan is the minimum of the Coverage Measure of its sub-goals or `True` if the plan has no sub-goals.

$$C_M^A(P) = \begin{cases} \texttt{True} & \text{if } Sg(P) = \{\} \\ \text{MIN}\left\{ C_M^A(g) \mid g \in Sg(P_A) \right\} & \text{otherwise} \end{cases} \tag{10}$$

3. The Coverage Measure of a goal is as follows (here, `T=True`, `F=False`, and `U=Uncertain`):

$$C_M^A(G) = \begin{cases} \texttt{T} & \text{if } \forall_{P \in Pl(G)}\; C_M^A(P) = \texttt{T} \wedge C^A(G) = \texttt{T} \\ \texttt{F} & \text{if } \forall_{P \in Pl(G)}\; C_M^A(P) = \texttt{F} \vee C^A(G) = \texttt{F} \\ \texttt{U} & \text{otherwise} \end{cases} \tag{11}$$

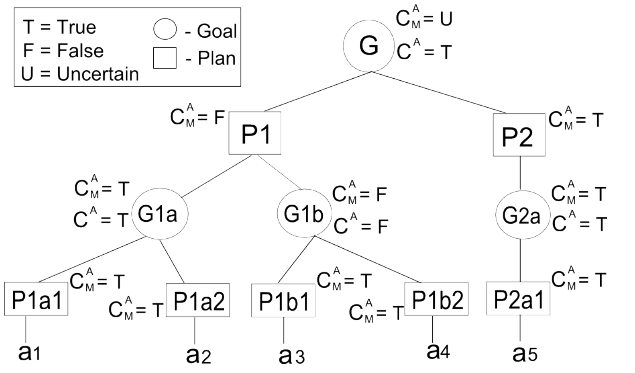

Figure 5: Example Abstract Coverage Measures for design

Figure 5 illustrates the propagation of these simplified Coverage Measures in a goal-plan tree. The `Uncertain` Coverage Measure value thus identifies the case where there is some path(s) with full coverage through the entire tree, but ensuring full plan coverage is dependent on plan selections. For example, in Figure 5 the path containing plan P2 has full coverage.

6. CONCLUSION

BDI Agent programming languages provide a powerful platform for developing complex applications. They support the use of domain specific information, which makes them very suitable for real and complex applications. However, their value also lies in the generic reasoning that is incorporated into the execution engine, independently from the domain based program. The key standard features on which much of the success of the paradigm is based, are hierarchical plan selection based on context conditions and persistent goals with failure recovery. Additional generic mechanism that can be incorporated into the execution engine to make these systems smarter, without requiring application specific coding, are of interest to the agent development community. In trying to identify such opportunities it is also important to be cognisant of the need to avoid overloading the developer with requirements to provide details which are not readily available.

In this work we have taken the concepts of coverage and overlap that have been used in Agent Oriented Software Engineering and refined these to support smarter agent systems. Importantly we do not require any additional information from the developer or the domain, beyond what is required in typical BDI agent development. Also importantly, all the complex calculation can be done offline at compile time, leaving only simple computational update processes during execution.

The basis of our approach is the process to calculate, using model counting, a numerical measure of the extent to which the set of plans for a goal cover the relevant state space. We recognise however that apparently high coverage at the immediate level can be compromised by lack of coverage in the sub-goals below. Consequently we define a measure which takes account of this factor, and discounts the immediate coverage based on the characteristics of the underlying goal-plan tree. We apply a similar approach to measuring overlap.

Having defined these measures we then show how they can be used for both plan selection and intention selection at execution time. In addition, we indicate how the measures can be used to identify potential Software Engineering issues. Based on the quantitative approach developed we then abstract back to a qualitative approach suitable for use during design, prior to full details being available to calculate numerical Coverage Measures. This provides better information than what is currently provided in agent design methodologies.

In this work we use an idealised and simplified agent programming language, and in particular we do not account for any decisions coded within plan bodies. Consequently it is possible that actual coverage is less than what is calculated with our method. However, we do not consider this a substantial disadvantage, as it is possible to replace test and action steps with new sub-goals whose plans have the test condition and action precondition, respectively, as context conditions. This would allow the coverage measure to detect the otherwise hidden constraints. Under certain assumptions, also, constraints in plan bodies could be automatically regressed to plans' context conditions (e.g., see [2]), though this is an orthogonal problem and is out of the scope of this work.

We also acknowledge that we have not yet implemented the reasoning described, based on these measures, and so do not yet have experience of their value in practice. Nevertheless, based on many years practical experience, and work with industry partners, we are convinced that these measures provide valuable information which, either alone, or in combination with additional aspects, can further improve the behaviour of autonomous intelligent agents.

Acknowledgments

We acknowledge the support of the Australian Research Council under Discovery grant DP1094627.

7. REFERENCES

[1] R. H. Bordini, A. L. C. Bazzan, R. de Oliveira Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1294–1302. ACM Press, 2002.

[2] L. P. de Silva, S. Sardina, and L. Padgham. First principles planning in BDI systems. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, volume 2, pages 1001–1008, Budapest, Hungary, May 2009. IFAAMAS.

[3] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2293–2299, 2007.

[4] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009.

[5] B. Horling, V. Lesser, R. Vincent, and T. Wagner. The Soft Real-Time Agent Control Architecture. *Autonomous Agents and Multi-Agent Systems*, 12(1):35–92, 2006.

[6] M. J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Annual Conference on Autonomous Agents (AGENTS)*, pages 236–243, New York, NY, USA, 1999. ACM Press.

[7] L. Padgham, J. Thangarajah, and M. Winikoff. Prometheus design tool. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1882–1883, 2008.

[8] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. Wiley Series in Agent Technology. John Wiley and Sons, NY, USA, 2004.

[9] A. S. Rao. Agentspeak(L): BDI agents speak out in a logical computable language. In W. V. Velde and J. W. Perram, editors, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World. (Agents Breaking Away)*, volume 1038 of *Lecture Notes in Computer Science (LNCS)*, pages 42–55. Springer, 1996.

[10] S. Sardina and L. Padgham. A BDI agent programming language with failure recovery, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011.

[11] J. Thangarajah. *Managing the Concurrent Execution of Goals in Intelligent Agents*. PhD thesis, RMIT University, Melbourne, Australia, 2004.

[12] J. Thangarajah, J. Harland, D. Moreley, and N. Yorke-Smith. Suspending and resuming tasks in BDI agents. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 405–412, Estoril, Portugal, May 2008.

[13] K. Vikhorev, N. Alechina, and B. Logan. Agent programming with priorities and deadlines. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 397–404, 2011.

[14] W. Wei and B. Selman. A new approach to model counting. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2005.

[15] H. Zhang, S. Y. Huang, and Y. Chang. An agent's activities are controlled by his priorities. In *Proceedings of the 2nd KES International conference on Agent and multi-agent systems: technologies and applications*, KES-AMSTA'08, pages 723–732, Berlin, Heidelberg, 2008. Springer-Verlag.