Improving Plan Execution Robustness through Capability Aware Maintenance of Plans by BDI Agents

Alan White[1], Austin Tate[2], Michael Rovatsos[3]

Artificial Intelligence Applications Institute
Centre for Intelligent Systems and their Applications
School of Informatics, University of Edinburgh, UK
[1]a.g.white@sms.ed.ac.uk, [2]a.tate@ed.ac.uk, [3]mrovatso@inf.ed.ac.uk

Keywords: Multiagent Teamwork, Belief-Desire-Intention, Planning, Capability, Robustness

Abstract. In a realistic environment, intentions of Belief-Desire-Intention (BDI) agents may be threatened by exogenous change. Subsequent activity failure may incur debilitative consequences that hinder both recovery and subsequent goal achievement. CAMP-BDI (Capability Aware, Maintaining Plans) embodies BDI agents with capability knowledge, allowing anticipation of threats to activity success and stimulating the proactive, preventative modification of intended plans. We describe resultant agent-level algorithms and supporting architecture, including extension to provide decentralized, distributed maintenance through structured messaging. Our results show superior goal achievement to a reactive equivalent in a stochastic environment, increasing with the likelihood of debilitative failure effects. We suggest CAMP-BDI offers a valuable approach towards robustness, particularly in tandem with reactive recovery methods.

1 Introduction

The Belief-Desire-Intention (BDI) approach defines intelligent agent behaviour through rational goal and plan selection. BDI has been widely employed by intelligent agents, including within realistic, stochastic and dynamic domains such as emergency response. In these environments, exogenous change during execution may contradict the assumptions made when forming intentions, risking activity failure and potential *debilitative* consequences. Current BDI architectures typically employ reactive approaches, such as replanning or plan repair, to handle activity failure; *Jason* agents (Bordini and Hübner [2006]), for example, enact predefined recovery plans upon goal failure. However, failure associated debilitation may increase the cost of, or even stymie, recovery from post-failure states. Continuous planning mitigates uncertainty by postponing planning decisions, but this shorter-term viewpoint risks inadvertent long term failure – such as where necessary resources are not reserved to protect against contention.

This paper describes the CAMP-BDI (Capability Aware, Maintaining Plans) approach, which embodies BDI agents with the ability to introspectively reason over their intended plans. This reasoning enables proactive plan repair to be performed where divergences between actual and anticipated world state threaten failure – allowing response to exogenous change whilst supporting long term plan formation and resource reservation. We present the following contributions:

- An algorithm for anticipatory plan repair behaviour, referred to as performance of *maintenance*
- Extension of local behaviour to encompass decentralized maintenance of distributed intentions

– A supporting architecture providing the capability, dependency, and obligation knowledge used to perform introspective reasoning and guide maintenance changes
– A policy mechanism allowing runtime tailoring of maintenance behaviour

An experimental implementation of CAMP-BDI was evaluated against a reactive replanning system, within a logistics environment subject to unpredictable exogenous changes. We gathered results over multiple experimental runs, for varying probabilities of failure-associated debilitations. CAMP-BDI was observed to offer superior goal achievement over reactive replanning, with superior efficiency (in planning calls per goal achieved) at higher likelihoods of post-failure debilitation – reflecting an increased difficulty of recovery and activity in post-failure world states.

2 Motivating Example

Our motivating example is a logistics domain, where the goals of the Multiagent System (MAS) are to deliver cargo to requesting locations in a stochastic, dynamic, continuous and non-deterministic environment. Uncertainty arises from agent health state, weather conditions (rainstorms may flood roads or cause landslips), or emergence of 'danger zones' rendering given locations unsafe. This extends from a single-agent approach, where a *Truck* agent transports an item of cargo, to employ multiple supporting agents – e.g. *Bulldozers* to clear blocked roads, *APCs* (military Armoured Personnel Carriers) to secure dangerous locations, and with logical agents acting as organizational controllers or brokers, within a distributed agent team.

Activity failure risks negative consequences that can hinder both recovery and future goal achievement. For example, Figure 1 depicts a *Truck* agent travelling a planned route from location A to *M*, when road $F \rightarrow M$ is rendered unusable by flooding – threatening *Truck*'s intended activity, *move*(F, M). This risks both failure of that activity *and* debilitation; i.e. *Truck* may be rendered unusable by both recovery plans and future intentions if stuck on $F \rightarrow M$ or damaged as a result. Even if found, recovery plans may be costly due to necessary backtracking or remedial actions – it may be cheaper to identify an alternate route immediately upon $F \rightarrow M$ becoming known as flooded, rather than pursuing the plan until failure at M. Finally, failure may not be solely deterministic – some states (such as *partial* flooding on $F \rightarrow M$) may not be significant enough in effect to *ensure* failure, yet still impact whether execution succeeds.

Our system aims to improve robustness – measured as goal achievement by the overall multi-agent system – in environments subject to exogenous change, where failure risks lasting debilitation, resource contention prohibits reliance on short term continual planning, and domain complexity renders probabilistic planning methods (such as MDP policy formation) intractable. We argue this requires *proactive* behaviour – to anticipate threats to activities and pre-emptively modify plans to avoid failure. It also requires extension to cover the distributed context of Multi-agent teams – i.e. when *Truck* is at risk of failing to meet an obligation, dependent agents should anticipate and compensate accordingly. Finally, we recognize deterministic state models only approximate realistic environments and account for where states can increase the risk of failure, but not significantly enough to be represented in deterministic operator preconditions.

3 Architecture Components

CAMP-BDI agents employ the following meta-knowledge components for introspective reasoning. These represent a subset of agent *Beliefs*, although semantics will be implementation-specific; we simply state that our CAMP-BDI algorithms require this information. To distinguish between a selected desire and the approach to achieve it, we define an intention i as combining a goal and associated plan: $i = \{i_{goal}, i_{plan}\}$ (as expressed in Simari and Parsons [2006]).

3.1 Capabilities

Capabilities define meta-knowledge regarding the activities performable and goals achievable by agents, including those delegated through formation of dependency contracts. This model is used to represent meta-knowledge information for both those activities directly performed by an agent and – as a necessary field provided within dependency contracts – those delegated to others; this allows the same reasoning algorithms to be applied for both of these types of activity.

An activity a is defined as equivalent to a *task* in a Hierarchical Task Network (HTN), in that it can be viewed as composite (i.e. decomposable) or atomic. An a is modelled as a deterministic state transition $F(a, S) = S'$; successful execution of a in state S achieves a successor state S'. We assume S maps to equivalent agent beliefs B; i.e. successful execution of a in S sees B updated to reflect S'. A given activity may correspond to an atomic action (primitive activity), or subgoal (composite activity) – the latter requires decomposition into, and execution of, a subplan. A plan p is an ordered sequence of activities $p\{a_1, \dots , a_n\}$, scheduled and executed to achieve some goal; a plan is primitive if every $a \in p$ is primitive. If *continual* planning is employed, p may contain composite activities, whose refinement into subplans is deferred until execution.

A capability $c(a)$, denoting the holding agents ability to perform a, has the following fields;

$$c(a) = \langle s, g(a), pre(a), \mathit{eff}(a), \mathit{conf}(a, B_a)\rangle$$

- s: signature s with name n and x parameters: $s = n(t_1, ..., t_x)$. A specific capability instance c in a MAS can be uniquely identified by combining s and the (identifier of the) agent holding $c(a)$. The parameters t are used to ground the abstract terms in $g(a)$, $pre(a)$ and $\mathit{eff}(a)$.
- $g(a)$: defines the goal state (S_G) – the set of atoms achieved through successful execution of a; i.e. the state transition of a in S is given by $F(a, S) = S'$, where $F_G \subseteq S'$. S_G will initially be an abstract specification, to be ground using parameters of s.
- $pre(a)$: preconditions (belief atoms) defining where a can be achieved - specifically, use of $c(a)$ is not *guaranteed to fail* if $pre(a) \subseteq B_a$, again ground through the parameters for s.
- $\mathit{eff}(a)$: the complete set of post-effects of using $c(a)$ – i.e. $\mathit{eff}(a) = g(a) \cup side\mathit{effects}(c(a))$, effectively defining S' for the activity transition function $F(a, S) = S'$. This can be used to distinguish between the (goal achievement) purpose of a and side-effects; i.e. *fly* and *drive* achieve the same $g(a)$ (to arrive at some location), with different total effects $\mathit{eff}(a)$.
- $\mathit{conf} : a \times B_a \to [0 : 1]$; a *confidence* function which (quantitatively) estimates the quality (in this context, indicating likelihood of success) of performing a through use of $c(a)$ in the execution context indicated by beliefs B_a, as a scalar value.

3.1.1 Capability typology

We define the *type* of a capability using two overlapping categories – *granularity* and *locality* – denoting whether $c(a)$ corresponds to the ability to achieve $g(a)$ through performing a primitive or composite a, and whether actual performance of a entails delegation (i.e. is external).

*3.1.1.1 **Primitive capabilities***

Primitive capabilities state the holding agent can achieve $g(a)$ (with the given preconditions, effects and confidence) through an atomic a; equivalent to Singh *et al.* [2010]'s concept of *know-how* regarding basic activities. This type represents the *effectors* an agent can use to directly change the world's state. Primitive capabilities require explicit specification at implementation time, similar to operators within a classical planning domains. In order to be executed, plans – whether single agent or distributed – must eventually resolve to performance of activities which correspond to primitive capabilities. In the distributed case, this may potentially be over multiple decomposing delegation relationships. As a consequence, all composite and external capability knowledge ultimately resolves to, and derives from, some subset of primitive capabilities.

3.1.1.2 ***Composite capabilities***

Composite capabilities represent knowledge of some plan(s) for achieving $g(a)$, where a is a composite (non-atomic, divisible) activity – either an i_{goal} (i.e. the root activity in a task decomposition hierarchy) or a subgoal within a plan. BDI agents are typically implemented as employing a library of plan recipes. Composite capabilities hold a $1 : n$ relationship with the plans known to an agent for achieving that $g(a)$. Each plan in the plan library will map to exactly one representative composite capability, with each composite capability mapping to $n \geq 1$ plans for achieving $g(a)$. This supports reasoning about abstract activities within a plan – including if continual planning is used, where the actual refinement decomposition of a will be delayed until closer to a's execution (where greater certainty is expected regarding a's execution context).

Composite capabilities can be formed automatically using the contents of an agent plan library; with $g(a)$ corresponding to some goal, and $pre(a)$ being the disjunction of selection conditions for all plans to meet that goal – i.e. defining states where the agent knows a selectable plan (we assume that if a plan p's selection conditions hold, so will preconditions for every $a \in p$). As the precise semantics of *how* $g(a)$ is to be performed – i.e. which exact plan would be selected, with what specific activities and effects – can vary with the specific execution context, we define $\mathit{eff}(a)$ as the state corresponding to $g(a)$. Where a is ground and the execution context B_a known, $\mathit{eff}(a)$ can be calculated by identifying which plan would be selected; for multiple options, it is assumed the plan with greatest estimated (as in 3.1.2.3) confidence would be used.

3.1.1.3 ***External capabilities***

CAMP-BDI agents are expected to advertise capabilities where they can accept obligations from others (with any authority constraints reflected by selective advertisement, and updated to reflect changes in circumstance such as confidence loss); recipients use the received information to form a corresponding *external* capability set, representing where an a can be performed through delegation. Both primitive and composite capabilities can be advertised, although knowledge of the plans represented by the latter will be restricted to the advertiser. As obligants identify and perform any plan selection within their internal reasoning, external capabilities are always modelled as being primitive – i.e. indivisible from the (potential) dependant's perspective.

3.1.2 The Confidence function

The *qualification* problem (McCarthy [1958]) recognises that, when specifying deterministic preconditions, it is impossible to represent every state which can *potentially* threaten successful execution as this would over-constrain operators to unusability. As a result, certain risk-increasing states will likely be deemed not significant enough to represent outright within preconditions – operator preconditions can be considered as defining the conditions under which failure is *not guaranteed*, rather than those where success *is*. The confidence function ($conf_a(a, B_a)$) allows identification of whether exogenous change impacting B_a has increased the risk of failure, even if preconditions still hold for a – i.e. $conf(driveAlong(F, M), B)$ is lower if $slippery(F, M) \in B$ than if $dry(F, M) \in B$. A numerical value allows semantic-independent comparison between different internal and/or external capabilities sharing the same s, whilst allowing flexible granularity during implementation (e.g. to use enumerated values $yes = 1$, $maybe = 0.5$, $not = 0$ rather than requiring precise, continuous, estimation for the probability of success).

Estimation depends on both the capability type and a itself. If a is unground, confidence indicates the *general* ability of that agent to achieve $g(a)$ in B_a – an *abstract* estimation. If a is ground, additional semantic information can be used for *specific* estimation. *Primitive* capabilities will use a predefined calculation for both abstract and specific estimation – such as considering past execution results (Singh *et al.* [2010] use such an approach for learning plan selection conditions, in order to support their reuse) as well as B_a. Implementation of the confidence function for a primitive capability requires domain and specific analysis, and relies upon the domain knowledge and the analytical or modelling abilities of a human designer.

3.1.2.1 ***Primitive Capability Confidence Estimation***

It is difficult to generalize the difficulty of primitive confidence estimation, as calculation will depend upon the specifics of the capability holding agent (i.e. that performs the activity) and the properties of the environment. We suggest implementation use information gathered during analysis of which states impact success and to what significance – as this will likely be gathered regardless, to specify operator preconditions for planning domains or define plans and selection conditions for the agent's plan library. We also do not require exact *probabilistic* estimation, but instead an indicative value; allowing flexibility over the level of granularity where more precise estimation is infeasible or impractical. Maintenance policies (in 3.3) can also modified in order to help compensate for consistent over or under-estimation.

3.1.2.2 ***External Capability Confidence Estimation***

External capabilities use a fixed, abstract confidence value, as received in the relevant capability advertisement. Agents are unlikely to share the semantic knowledge for *specific* confidence estimation within capability advertisements, as the recipients may lack the modelling or sensory ability required to interpret – particularly as MAS approaches are often motivated by distribution of knowledge and sensory capacity within an environment. However, *specific* estimates are provided for delegated activities in the external capability field of contracts (in 3.2).

3.1.2.3 ***Composite Capability Confidence Estimation***

Composite capabilities represent (knowledge of) a set of plans for a given goal ($P_{capability}$). Estimation of composite capability confidence utilizes the estimated confidence of the individual plans in $P_{capability}$ – i.e. assessing the quality of plans the agent can employ for that a. Estimation of plan confidence is also employed when performing maintenance, to determine whether to accept generated plans (described in 4.3). This makes plan confidence estimation relevant both in anticipating threats (i.e. to estimate confidence for subgoals/composite activities through their potential refinement subplans), and also generally to support our process of addressing identified threats through forming and inserting maintenance plans (described in the following section). We first discuss the methods for plan confidence estimation, before discussing *how* confidence for each $p \in P_{capability}$ is used to form an overall confidence estimate for the capability.

Plan confidence may be calculated in various ways. One method, similar to TÆMs' *q_min* metric (Lesser *et al.* [2004a]), is to use the minimum confidence of a constituent activity – given below, with *conf* expanded to consider a plan p as the first argument, where B_p is the execution context of the first activity in p. B_a will be updated with a_n's (capability-defined) effects following confidence estimation of a_n, to estimate an execution context for the following a_{n+1}:

$$conf_{min}(p, B_p) = min_{a \in p}\, conf(a, B_a)$$

This form of estimation can be used if every $a \in p$ must have 'acceptable' confidence for p to be considered acceptable. Such a constraint can guard against future maintenance, as the resultant value indicates if *any* activity in the plan is below the relevant confidence threshold (see 3.3) – although this does not prevent subsequent exogenous change from requiring maintenance regardless. However, this may over-constrain plan acceptance during maintenance by defining plan confidence by the worst constituent activity – i.e. a plan containing many low-confidence activities would be given the same confidence as one with a single low confidence activity.

An alternative is to average the confidence of *all* plan activities. A weighting function (given as $w_i \to \mathbb{Z}_{\geq 0}$) can scale the significance of an $a_i \in p$ in contributing to p's confidence, based upon relative precedence ($1 \leq i \leq n$ denotes the location of a in a plan p of n activities):

$$conf_{avg}(p, B_p) = \left(\frac{\sum\limits_{i=1}^{n} conf(a_i, B_{a_i}) \times w_i}{\sum\limits_{i=1}^{n} w_i} \right)$$

For example, w_i may be greater for lower values of i, reflecting more imminent execution; e.g. $w_i = (n-i)/i$ to determine the weight of a_i. This would reflect the likelihood of the environment changing over time, and consequently increasing uncertainty regarding the execution context – and confidence – for latter activities. Additionally, in some environments it may be difficult to always generate plans where *every* activity has confidence above the defined threshold. An averaged value allows *incremental* improvement in i_{plan} confidence, by allowing insertion of maintenance plans that still offer an *overall* confidence improvement over a threatened activity.

When calculating composite capability confidence, we assume the highest confidence plan is always selected for a goal. Composite capability confidence is therefore the highest of a selectable plan (i.e. where preconditions hold – if none are selectable, 0 is returned), where a_{goal} is the activity being performed using the composite capability and $B_{a_{goal}}$ it's execution context:

$$conf(a_{goal},\ B_{a_{goal}}) = max_{\substack{p \in P_{capability} \\ pre(p) \subset B_{a_{goal}}}}\ conf(p,\ B_{a_{goal}})$$

This equates to formation and traversal of an AND-OR tree (similar to goal-plan trees in Thangarajah *et al.* [2003]) representing all potential plan and subplan execution 'paths' to decompose and achieve a_{goal}. The return value derives from visiting every leaf activity ($O(n)$ worst case complexity, for n leaf nodes), and originates from either a primitive or external capability confidence value (i.e. as estimated for a leaf node activity). Cyclical loops are assumed not to occur, due to the decompositional nature of plans; this property is also prevents infinite loops in agent activity itself. Use of advertised confidence for external capabilities – rather than requesting that potential dependants calculate a value locally – restricts the semantic knowledge requirements to the advertising agent.

There is considerable scope for domain specific optimization of confidence calculation for both primitive and composite types; the specific implementation of plan confidence evaluation (whether based on minima, averaging, or some other approach) is likely to be influenced by the environment's properties and the agent's planning implementation. In the $conf_{min}$ case, α-β pruning can to improve common-case complexity if estimation is being performed to determine whether confidence is below a set threshold value (for example, a policy maintenance threshold). Finally, composite capabilities representing runtime planning abilities require custom implementation of confidence estimation, similar to as for primitive capabilities.

3.2 Obligation and Dependency Contracts

Dependency *contracts* are assumed to be formed as far in advance of execution as possible, to protect against contention from others for both agent capabilities (i.e. to form dependencies upon) and environmental resources. CAMP-BDI agents are aware of their obligation and dependency contracts; these state the mutual beliefs established between dependants and obligants regarding a delegated activity. CAMP-BDI requires the following fields to be represented and established as part of contract formation;

- An **activity**, which the obligant(s) agree to perform upon the dependant's request. We refer to the dependant's i_{plan} containing that activity as the *dependent intention*.
- **Causal link** states; states the dependant intends to establish, representing the effects of preceding activities in the dependent plan, prior to execution of the delegated activity.

– An **external capability**, used by obligant(s) to convey the (anticipated) post-effects and confidence for the activity – the latter estimating the execution context using the causal link states. If there is more than one obligant, the individual obligant capabilities will be merged:
  - *Confidence* is set as the *minimum* individual obligant confidence
  - *Preconditions* are formed as the conjunction of all obligant preconditions
  - *Effects* are set as the union of all obligant post-effects

– A **maintenance policy**, used to guide maintenance behaviour (see 3.3).

3.3 Maintenance Policies

A maintenance policy[1] defines specific fields, applied to a defined set of agents and/or capabilities, which influence maintenance behaviour:

– **Threshold**: the minimum confidence (quality) value for an activity; runtime modification of this value also allows compensation for over-sensitive confidence estimation

– **Priority**: guides relative prioritisation within maintenance behaviour, when multiple activities in an i_{plan} are identified as being under threat of failure

These fields allow the additional computational costs incurred by proactive maintenance to be balanced against the benefits of avoiding failure; lower threshold values can be associated with those activities carrying more severe failure penalties, increasing the effective frequency under which an agent will attempt to identify and perform confidence-raising modifications to i_{plan}. Similarly, if an activity has little or no significant failure consequences (and reactive failure handling is provided), the associated maintenance threshold can be set to a very low or zero value – 'turning off' maintenance in favour of reactive recovery.

Maintenance policies are intended to be dynamically modifiable; both field values and the sets of agents and/or capabilities the policy applies to can be altered during runtime. This allows a degree of tailoring for the implementation-time defined maintenance behaviour; agent-capabilities associated with activities with greater (in either probability or severity) failure consequences can be given lower thresholds and higher priorities. In this context, maintenance policies provide an additional layer of maintenance behaviour definition, aiding genericization of the underlying maintenance algorithm and even potential reuse of agents in differing environments.

Contract maintenance policies merge dependant and obligant policies; these are respectively matched to the dependent's i_{goal} (i.e. where i_{plan} contains the delegated activity) and that associated with the obligant and delegated activity (i.e. obligant's i_{goal}). To restrict changes to a minimal subset of the overall distributed plan, the merged maintenance policy uses the most constrained field values (lowest threshold and highest priority), ensuring that obligants will have met their conditions for maintenance (i.e. will have performed any confidence improving modifications they are capable of) before informing dependants of confidence changes.

4 The CAMP-BDI algorithm

CAMP-BDI agents extend the generic BDI reasoning cycle (Rao and Georgeff [1995]) through the addition of contract formation and maintenance steps (Algorithm 1). The *maintain* function is invoked in three contexts. Firstly,(**1**) after an intention i is selected; secondly, (**2**) after a dependency is updated as indicated by receipt of *obligationMaintained* messages from the relevant obligant(s); finally (**3**) to maintain mutual beliefs by maintaining preformed or cached plans for existing obligations when the agent is otherwise idle (has no intentions).

The *formAndUpdateContracts* function forms new, and updates existing, dependency and obligation contracts (the latter sending *obligationMaintained* messages). This function executes after *maintain*, to account for changes in either the relevant i_{plan} or associated dependencies

Algorithm 1: The CAMP-BDI reasoning cycle; changes from the generic algorithm given by Rao and Georgeff [1995] are highlighted by **bold** text

```
initializeState();
while agent is alive do
    $D \leftarrow$ optionGenerator($eventQueue$, $I$, $B$);
    $i \leftarrow$ deliberate($D$, $I$, $B$);
    /* (1) Maintenance of currently selected intention $i$ */
    if $i \neq \emptyset$ & i not waiting on a dependency to complete then
        $i \leftarrow$ updateIntentions($D$, $I$, $B$);
        $B_i \leftarrow$ estimated execution context of $i$;
        maintain($i$, $B_i$);
        formAndUpdateContracts($i$);
        execute();
    /* (2) Maintenance of intentions in response to
       dependency changes received from obligant */
    for each obligationMaintained message $\in$ eventQueue do
        $i_{dependency} \leftarrow$ the associated dependant intention;
        $B_{dependency} \leftarrow$ estimated execution context of $i_{dependency}$;
        maintain($i_{dependency}$, $B_{dependency}$);
        formAndUpdateContracts($i_{dependency}$);
    /* (3) Maintenance of obligants held by this agent, if no
       intentions were selected */
    if $i = \emptyset$ then
        for each obligation contract $\in$ agent's Obligations do
            $i_{ob_{goal}} \leftarrow$ activity defined in $obligation$;
            $i_{ob_{plan}} \leftarrow$ cached plan for $obligation$ (to achieve $i_{ob_{goal}}$);
            $i_{ob} \leftarrow i_{ob_{goal}}$, $i_{ob_{plan}}$;
            $B_{ob} \leftarrow$ execution context estimated using (causal links in
            $obligation \cup B$);
            maintain($i_{ob}$, $B_{ob}$);
            formAndUpdateContracts($i_{ob}$);
    getNewExternalEvents();
    $I \leftarrow$dropSuccessfulAttitudes();
    $I \leftarrow$dropImpossibleAttitudes();
    $I \leftarrow$postIntentionStatus();
```

(i.e. propagating changes received within *obligationMaintained* messages). The *obligationMaintained* message itself contains a contract (i.e. as given in 3.2) as a body, with that contract updated to account for the changes to the i_{plan} resulting from maintenance.

If there are multiple possible intentions ($|I| > 1$), the agent only attempts to maintain the specific $i \in I$ which was selected for execution. We view intention selection as goal driven behaviour, such that subsequent changes to i_{plan} by maintenance would not invalidate the original choice to select i. This avoids the cost of maintaining every potential intention *prior* to selection – especially as any changes made by maintenance to (the i_{plan} of) *unselected* intentions could be rendered futile or even unnecessary by the subsequent execution of the *selected* i.

Algorithm 2 defines the *maintain* function. Given an intention i, *maintain* first identifies any threats to activities within i_{plan}, and – if any are found – will subsequently (attempt to) modify

i_{plan} appropriately in mitigation. We use a two step process – firstly forming an ordered agenda (described in 3) of *maintenance tasks* (see 4.1), each representing a threatened activity, before iterating through that agenda until either a task is successfully handled *or* the entire agenda has been iterated through. The function *handleMaintenanceTask* (described in 4.3) attempts to modify i_{plan} to address the issue represented by a given maintenance task, returning true if successful (i.e. i_{plan} is modified).

The algorithm terminates after the first successfully handled maintenance task, as modifications may invalidate other maintenance tasks in the agenda. An alternative approach would be to iteratively diagnose and handle tasks, until either an empty agenda is formed *or* handling fails – but this is likely to result in significantly higher computational cost, and has less certainty of termination due to the potential for incremental growth of the agenda if tasks result from each maintenance plan change. Agenda formation and handling are separated to allow the former to prioritize the set of threatened activities; this decoupling also facilitates future investigation into alternate approaches for either threat diagnosis or handling.

Algorithm 2: The *maintain* function

```
Data: $i$ – An intention; a plan $i_{plan}$ to meet some goal $i_{goal}$
    $B_i$ – The estimated execution context of the first activity in $i_{plan}$
$handled \leftarrow$ false;
$agenda \leftarrow$ new empty Agenda;
$agenda \leftarrow$ the agenda returned by formAgenda($i_{goal}, i_{plan}, B_i, agenda$);
while $\neg handled$ & $\neg agenda.isEmpty()$ do
    $handled \leftarrow$ handleMaintenanceTask($agenda$.removeTop());
Update Dependency contracts;
if $i$ is an Obligation then
    Update contract and send to the dependant in an $obligationMaintained$ message;
```

In our motivating example (Fig. 1), *formAgenda* would anticipate – using the associated capability's *pre* field – that flooding of $F \rightarrow M$ will violate preconditions of *move*(F, M) when it is executed, and subsequently inserts a corresponding maintenance task into the agenda. The subsequent *handleMaintenanceTask* call for the generated maintenance task would seek to modify i_{plan} such that i_{goal} can be achieved; i.e. avoiding use of *move*(F, M) or (if capable) removing the flooded state of $F \rightarrow M$.

4.1 Maintenance tasks

A maintenance task mt defines a threatened activity a, a *type* of threat (*preconditions* or *effects*, serving to also indicate the desired handling method), estimated execution context B_a for a, estimated confidence $conf_a$ of a given B_a, and the maintenance policy mp_a associated with a;

$$mt = \langle a, type, B_a, conf_a, mp_a \rangle$$

Capability knowledge facilitates introspective reasoning for maintenance task generation. Activities are mapped to – in precedence order – internal capabilities, contract-contained external capabilities, and finally advertised external capabilities; this assumes activities are only delegated where necessary and that agents adopt the least complex (fewest activities) approach for performing any activity. If an activity can be met by several external capabilities, that with highest general confidence is selected; mirroring the most likely criteria for obligant selection. Maintenance tasks are ordered in the agenda first by their (mp_a defined) priority, and then by the precedence of a

(i.e. to prioritize activities with more imminent execution) in i_{plan}.

Preconditions maintenance tasks are generated where a's preconditions do not hold in B_a ($pre(a) \nsubseteq B_a$), and indicate maintenance should attempt to restore precondition states. Successful handling of this type generates a plan to achieve the required preconditions (i.e. ensure that $pre(a) \subseteq B_a$), then inserts that plan such that it will execute immediately before $mt.a$ does. As this insertion will see expansion of i_{plan}, the *preconditions* type is only generated if it is desirable to *preserve* a – due to a achieving a goal state, or to avoid (the costs of) cancelling a pre-existing dependency contract for a. These restrictions aim avoid scenarios where the iterative handling of preconditions tasks (over multiple reasoning cycles) causes the iterative expansion of i_{plan} – with consequent loss of optimality for achieving i_{goal} due to focusing upon preservation of that individual activity ahead of considering the overall *plan* requirements for achieving i_{goal}.

Effects maintenance tasks are generated if either $pre(a) \nsubseteq B_a$ and a does *not* require preservation, or $conf_a < mp_a.threshold$ (a is of unacceptable quality, i.e. considered to be at an unacceptable risk of failure). This type indicates that a should be replaced by a maintenance plan achieving the same stateas $\mathit{eff}(a)$, but with greater confidence. We also employ this type for violated preconditions where a is not to be preserved; this allows potentially greater reconsideration of i_{plan}, including whether a should be preserved at all, avoiding the potential for iterative expansion associated with handling the preconditions type.

4.2 Agenda Formation

Agenda formation (algorithm 3) employs recursion (**2**) to support hierarchical plan structures (where composite activities are decomposed into subplans), iterating through each leaf activity (in the case of continual planning, including any composite activities which have not yet been refined) in execution order. The *getCapability* function associates each activity with it's appropriate representative capability. We assume that CAMP-BDI agents will hold capability meta-knowledge covering every a they can execute; i.e. that agents must have the *know-how* to perform or decompose (as appropriate) any i_{goal} they can adopt, or any a lying within an i_{plan} adopted as an intention.

Capability knowledge is used to identify threats to leaf activities (**1**) and form representative maintenance tasks to be inserted into the agenda. At the end of each iteration, B_a is updated with activity effects to estimate the execution context for the subsequent activity; this is discarded by the *maintain* algorithm (top level called), but preserves the estimated post-execution context when performing recursive calls.

A *consolidate* function (**3**) merges multiple maintenance tasks for activities within the same subplan into a single effects maintenance task – where $mt.a$ is the composite activity being refined by that subplan. This helps avoid recurrent costs of re-diagnosing and handling each individual threatened activity over multiple reasoning cycles, by instead defining a maintenance task that – when handled – entails reformation of an entire (but minimal) subset of i_{plan} containing multiple threatened activities, effectively handling those threats in a single *maintain* call.

4.3 Handling Maintenance Tasks

Handling a maintenance task (mt) requires modification of the i_{plan} containing $mt.a$, by the formation and insertion of a *maintenance plan* into i_{plan}. This is performed by the *handleMaintenanceTask* function (Algorithm 4), which calls the *handlePreconditionsTask* ((**1**) – detailed in Algorithm 5) and *handleEffectsTask* ((**3**) – detailed in Algorithm 6) functions to respectively handle the *preconditions* and *effects* types of maintenance task. As part of maintenance plan generation, capability knowledge will be used to both form an appropriate planning problem and specify an

Algorithm 3: The *formAgenda* function

```
Data: $g$ – a goal met, or composite activity performed, by $p$
      $p$ – plan of $n$ activities $\{a_1, ..., a_n\}$ to perform $g$
      $agenda$ – priority ordered list of maintenance tasks; empty in initial (top-level) call
      $B_a$ – estimated execution context of $a_0$ in $p$
Result: $agenda$ updated with maintenance tasks for $p$
      $B_a$ updated with post-effects of $p$ (used by recursion)
$B_{start} leftarrow$ copy of $B_a$ (for execution context estimation);
for each activity $a \in p$ do
    if $a$ is abstract then
        return $agenda, B_a$;
    $c_a \leftarrow$ getCapability($a$);
    /* (1) Generate maintenance tasks for leaf activities */
    if $c_a$ primitive $\|$ ($c_a$ composite & ($a$ is not decomposed into a subplan)) then
        if maintenance task $mt$ found for leaf activity $a$ then
            Add $mt$ to $agenda$;
            Update $B_a$ with $c_a$.eff($a$);
    /* (2) Recursion for decompositional subplans */
    else if $c_a$ composite & ($a$ is decomposed into a subplan) then
        $p_a \leftarrow$ subplan decomposing $a$;
        $agenda, B_a \leftarrow$ formAgenda($a, p_a$, $agenda$, $B_a$);
    /* (3) Consolidate multiple tasks into one */
    $agenda \leftarrow$ consolidate($g$, $agenda$, $B_{start}$);
return $agenda, B_a$;
```

operator set reflecting the capabilities of the maintaining agent (i.e. defining which activities it can be perform). In the case where a *preconditions* type cannot be handled, an equivalent *effects* maintenance task with the same field values is generated and handled instead (**(2)**) – this allows for the replacement of $mt.a$ rather than face definitive failure from violated preconditions, and effectively relaxes the maintenance planning problem by removing the necessity of preserving $mt.a$ (i.e. as required by a *preconditions* task). For example, if *Truck* cannot restore preconditions for *move*(F, M), it will attempt (through effects maintenance) to find an alternate method to achieve the required goal state $at(M)$.

4.3.1 Performing Preconditions Maintenance

Preconditions maintenance (Algorithm 5) attempts to generate a plan re-establishing preconditions of $mt.a$, to be inserted prior to $mt.a$ (similar to *prefix* plan repair in Komenda *et al.* [2014]). Generated maintenance plans are only accepted for insertion if their estimated confidence is above that defined by $mt.mp_a.threshold$. This attempts to prevent requiring the subsequent maintenance of the i_{plan} as a result of accepting and inserting a suboptimal confidence plan. However, this restriction is *not* applied if $mt.a$ is the next activity in i_{plan} to execute – instead, we deem adoption of any *greater* confidence plan as being preferable over definite failure (this assumes it is easier to prevent than recover from failure, as per our motivation).

Fig. 2 illustrates a preconditions maintenance scenario based upon the motivating example shown in Fig. 1. Here *Truck* holds plans to travel to location M – as the road $D \rightarrow F$ is blocked, a preconditions task is generated for the threatened future *move*(D, F) activity (in this example, we assume *Truck* wishes to preserve this activity). Fig. 3 illustrates the outcome of successful

Algorithm 4: The *handleTask* function

```
Data: $mt$ – A maintenance task
      $i$ – The intention requiring maintenance; $i = \{i_{goal}, i_{plan}\}$
Result: boolean – true if $i_{plan}$ is modified and $mt$ addressed.
$handled$ ←false;
if $mt.type$ =preconditions then
    // (1) Handle preconditions type of $mt$
    $handled$ ←handlePreconditionsTask($mt, i$);
    if $\neg handled$ then
        // (2) Create equivalent effects task for $mt.a$
        $mt$ ←new MaintenanceTask(effects,$mt.a$, $mt.B_a$, $mt.conf_a$);
    else
        return $handled$;
// (3) Handle effects type of $mt$
return handleEffectsTask($mt, i$);
```

Algorithm 5: The *handlePreconditionsTask* function

```
Data: $task$ – a maintenance task
Result: true if a plan was found and inserted
$i_{mt}$ ← plan containing $task.a$;
$c_a$ ← getCapability($task.a$);
Define planning problem $prob_a$, with initial state = $task.B_{mt}$ and goal = $c_a$.pre($task.a$);
if acceptable plan $plan_a$ solving $prob_a$ found then
    Insert $plan_a$ into $i_{mt}$ as predecessor of $task.a$, and return true;
return false;
```

maintenance; a maintenance plan to restore those preconditions – using a *Bulldozer* to re-open $D \rightarrow F$ and remove the associated blocked state – is inserted to form a prefix to $mt.a$.

4.3.2 Performing Effects Maintenance

Effects maintenance attempts to substitute a subset of the plan containing $mt.a$, with a new (sub)plan that will achieve identical effects to the replaced subset. For example, in the plan given by Fig. 2 preconditions may hold – yet have *slippery* road conditions reduce confidence in travel over $D \rightarrow F$ to an unacceptable level – leading to generation of an effects maintenance task for *move*(D, F). Successful handling will modify a (minimal) subset containing $mt.a$ of the i_{plan} for *deliverCargo*, restoring the overall confidence in achieving that goal to an acceptable level.

Our algorithm (algorithm 6) adopts a similar approach to Hierarchical Task Network (HTN) plan repair – upwards recursion is used to re-refine composite activities (subgoals or the root i_{goal}), terminating when either an acceptable confidence (greater than $mt.mp_a.threshold$) maintenance plan is found and inserted, or i_{goal} is reached without success (i.e. has attempted and failed to reform the entire i_{plan}, through iteration at **(3)** in the algorithm). We trade-off the potential cost of multiple planning calls at goal/subgoal levels against the stability loss and computational costs associated with performing complete replanning (Fox *et al.* [2006]).

Dependency cancellation may carry costs from communication; there is also a risk that external capabilities may no longer be available for forming *new* dependencies upon following changes in circumstance (despite having a pre-existing and now-cancelled, dependency contract). This can

Algorithm 6: The *handleEffectsTask* function

```
Data: $mt$ – a maintenance task
Result: true if a plan was found and inserted into the $i_{plan}$ containing $mt.a$
$a \leftarrow mt.a$;
$i_{mt} \leftarrow$ intended plan containing $mt.a$;
if $i_{mt}$ is a hierarchical plan then
    $p_{mt} \leftarrow$ subplan of $i_{mt}$ containing $a$;
else
    $p_{mt} \leftarrow i_{mt}$;
$B_{mt} \leftarrow mt.B_a$;
/* (1) Attempt replacement of $mt.a$ only                          */
if $a$ not last in $i_{mt}$ $\|$ $a$ has subsequent dependencies then
    $c_a \leftarrow$ getCapability($a$);
    Define planning problem $prob_a$, with initial state = $B_{mt}$ and goal = $c_a$.effects($a$);
    if acceptable plan $plan_a$ found for $prob_a$ then
        Replace $a$ in $p_{mt}$ with $plan_a$;
        return true;
/* (2) Attempt replacement of $mt.a$ and it's suffix in $p_mt$     */
if $a$ not first in $i_{mt}$ $\|$ $a$ has preceding dependencies then
    $a \leftarrow$ goal achieved by $p_{mt}$;
    $c_a \leftarrow$ getCapability($a$);
    Define planning problem $prob_a$, with initial state = $B_{mt}$ and goal = $c_a$.effects($a$);
    if acceptable plan $plan_a$ found for $prob_a$ then
        Replace the suffix of $p_{mt}$ from $a$ inclusive with $plan_a$;
        return true;
/* (3) Iterates through increasingly abstract plan levels */
while $a \neq$ root goal of $i_{mt}$ do
    $a \leftarrow$ goal activity for $p_{mt}$;
    $B_{mt} \leftarrow$ estimated execution context of $a$;
    $c_a \leftarrow$ getCapability($a$);
    Define planning problem $prob_a$, with initial state = $B_{mt}$ and goal = $c_a$.effects($a$);
    if acceptable plan $plan_a$ found for $prob_a$ then
        // (4) Use $plan_a$ to re-decompose/re-refine $a$
        Replace $p_{mt}$ with $plan_a$;
        return true;
return false;
```

stymie maintenance planning if use of a particular – now unavailable – external capability is necessary to achieve i_{goal}.

To account for these dependency associated issues, our algorithm attempts two additional restricted scope planning operations at the lowest (i.e. most specific subplan) level of iteration (the subplan containing $mt.a$). Firstly, if dependency contracts exist for $mt.a$ or it's successors in that subplan, the algorithm attempts to generate a maintenance plan that can be inserted to directly replace $mt.a$ (**1**); retaining successive activities and associated dependency contracts (Fig 4).

Secondly, if dependencies precede $mt.a$, the algorithm also attempts *suffix* plan repair (similar to repeated lazy repair in Komenda *et al.* [2014]); the generated maintenance plan replaces

$mt.a$ and it's successors in that subplan, but preserve preceding activities (**2**). Fig. 5 depicts an example, where the inserted maintenance plan achieves the goal state defined by the parent activity (i.e. shares the same goal as the subplan being modified with a new suffix). These two more constrained cases attempt to reduce disruption to a *distributed* plan performing team, at the cost of (potentially) requiring extra planner calls.

The algorithm in the worst case iterates and attempts to plan at all levels of a hierarchical i_{mt}, including at the initial p_{mt} level twice (once for a failed preconditions maintenance task, and once for replacement of $mt.a$ only), equivalent to $O((n+2)p)$ complexity (where n is the number of plan levels, and p the cost of planning). This, however, may still entail significant *actual* computational cost from performance of multiple planning operations.

5 Distributed Behaviour

Multiagent Systems (MAS) use co-operative teams of agents to achieve goals unattainable by individuals; failure in one agent's activity can reciprocally impact other team members and threaten failure of the distributed plan. Our approach assumes hierarchical agent teams arise from delegation to, and decomposition into plans by, obligants. We define a decentralized approach for performing distributed maintenance, as centralized approaches are often infeasible for realistic domains due to the distribution of knowledge and capabilities in these environments. We apply the agent level maintenance algorithms to the distributed context, using structured communication to drive adoption of maintenance *responsibility* at increasingly abstract levels of the team hierarchy (Fig. 8).

The supporting architecture (Section 3) is critical in supporting distributed maintenance. Dependency and obligation contracts provide *specific* capability information for delegated activities – the external capabilities field makes this information available for use by dependants, but also offsets semantic *knowledge* requirements to the obligant(s) which actually *provide* the contents of that field. As internal and external capabilities share the same representative model, our maintenance reasoning algorithms can reason over delegated activities using the same logic as for those performed through internal capabilities.

Upon completing *maintain* for an i_{plan}, where the associated i_{goal} corresponds to an obligation, the (waiting, quiescent) dependent agent is messaged. This communicates changes in the i_{plan} (which may stem from exogenous change and/or successful maintenance) through an *obligationMaintained* message – signifying the obligant has made any confidence-raising changes it is capable of (by communicating the correspondingly updated contract within the message body), through performing local-level maintenance. This allows the dependant to immediately maintain their *own* dependant intention upon receipt (if necessary), with the knowledge that the obligant has performed any possible change at it's more specific local level. Obligants maintain intentions both when actively performing (as an intention) an obligation, or if currently not pursuing *any* intention (Algorithm 1) – the latter allowing otherwise idle agents to maintain (contractual) mutual beliefs regarding the future execution of dependencies.

Dependants adopt maintenance responsibility if and when obligants cannot maintain confidence (including ensuring preconditions hold) in their subpart of a distributed plan; this restricts changes in a distributed plan to the 'lowest' (most specific) agent level. Responsibility will gradually move up a team hierarchy until an agent maintains an intention with an outcome which is acceptable to both itself and, if applicable, the direct dependant of that intention's i_{goal} (i.e. such that the dependant does not require to maintain *their* dependant intention on the basis of low confidence or unmet preconditions in the updated contract's external capability field).

The resultant behaviour (shown in Fig. 8) can be described as follows;
1. Agents *C* and *D* call *maintain* within their local reasoning cycle(s).

2. *C* and *D* individually perform post-maintenance messaging; each sends a *obligationMaintained* message to *B* that includes contracts updated to account for any maintenance changes.
3. *B* calls it's *maintain* method upon receipt of *obligationMaintained* messages from all obligants. Information in the messaged, updated contracts is used to update the contract held by *B* for that dependency, which will subsequently be updated and sent to *A* after *maintain* completes.
4. *B* sends *A* post-maintenance messaging, again using *obligationMaintained* messages.
5. *A* calls *maintain* upon receipt of *B*'s message; as *A* is not an obligation, no further messaging is required.

Contracts help synchronize this behaviour by defining a common maintenance policy for delegated activities, which is employed by both the obligant(s) and dependant. Shared confidence threshold triggers – combined with the sequencing of *maintain* calls and the communication of contract updates within *obligationMaintained* messages – ensure that, for a dependent to diagnose and attempt to handle an effects maintenance task, the obligant must have first done the same. Although the above example indicates a linear approach to dependency formation, indirect 'self dependencies' can emerge – for example, in the above, *D* may form a dependency upon some other capability of *A* in the course of performing it's own intention.

As an example, Fig 6 depicts a simple distributed intention where *Truck1* holds an obligation to perform a delivery task for *LogisticsHQ*, but has been damaged – reducing confidence in the intended movement. *Truck1* attempts effects maintenance but, being unable to self-repair, cannot restore confidence to an acceptable level. *Logistics HQ* detects loss of confidence for the corresponding activity within it's own (dependent) i_{plan} following receipt of an *obligationMaintained* message from *Truck1* (conveying the change in confidence), and adopts maintenance responsibility. Successful maintenance sees *Logistic HQ*'s i_{plan} employ a new, undamaged, obligant – *Truck2* – able to perform delegated activities with superior confidence (Fig 7) to *Truck1*. The resultant behaviour is equivalent to maintenance of a *local* i_{plan}, but occurs across the hierarchical team executing a distributed plan (where obligants effectively refine delegated tasks).

Our overall design aims to replicate re-refinement based HTN plan repair, over a distributed plan where the delegation of activities (and their performance by obligants through the formation and execution of corresponding plans) is analogous to a task refinement. Agents adopt responsibility for maintenance when executing their own planned activities (as in Algorithm 1), and in response to the outcome of obligant maintenance. In the latter case, the dependant can use contractual information to judge whether obligant maintenance outcome is acceptable (using it's *own* policy-defined standards) and will modify the dependant i_{plan} if not.

6 Evaluation

We compared a MAS of CAMP-BDI agents against a system using a reactive approach, in our previously described motivating logistics domain. In the reactive system, agents attempted replanning in response to activity failure; this was similar in concept to the approach of determinization-based probabilistic planners like *FF-Replan* (Yoon *et al.* [2007]), allowing us to argue it is an appropriate representation for the handling of unexpected outcomes[2]. Finally, a system with *no* failure mitigation strategy provided a 'worst-case' baseline for performance.

The MAS was evaluated in terms of overall goal achievement (successful deliveries), number of activities executed per goal achieved (efficiency), and the average planning calls per achieved goal (indicating computational cost). Possible exogenous changes in the environment included landslips (blocking roads), locations becoming dangerous, or rainfall gradually flooding roads (with an interim *slippery* state being associated with an increased probability of failure for activities using that road). A variety of heterogeneous agent types existed, with delegation and co-ordination used to address negative world states:

- **Truck** agents could load, unload, and transport cargo objects
- **APC** (Armoured Personnel Carrier) agents could render dangerous locations safe
- **Hazmat** agents could decontaminate roads made toxic by spills of hazardous cargo
- **Bulldozer** agents could unblock roads closed by landslips.

All experiments were performed on a system with an Intel i5-3750k processor (3.5Ghz) and 16GB RAM, running Java v1.8.0_31. The *Jason* agent framework (Bordini and Hübner [2006]) was extended to provide our experimental systems; both to support contract formation during distributed intention execution, and to integrate *LPG-td* (Gerevini and Serina) as the runtime planner for both CAMP-BDI and Replanning agents. A classical planner offered both greater flexibility than an HTN or plan library approach, and faster performance than probabilistic methods – making it a suitable analogue to a real-world implementation.

Experiments were performed for ten runs of each approach, with each run lasting for the generation (and success or failure in meeting of) 100 cargo delivery goals and employing the same procedurally generated geography. Three types of post-failure debilitation could occur; cargo could be *destroyed*, cargo could be *spilled* (rendering roads toxic unless decontaminated), or agents could become damaged (with graded degrees and associated confidence loss). In the latter case, agents would 'heal' over time if idle. We evaluated performance for $n = 0.2$, 0.4, 0.6 and 0.8; corresponding to 20, 40, 60 and 80$\%$ chances of the above debilitation. These probabilities were applied individually for each debilitation type, with cargo damage/spillage debilitation only possible if a failed activity involved loading, unloading or moving whilst carrying cargo. Results for ten experimental runs, performed under fixed simulation seeds, were averaged.

Our results show CAMP-BDI enjoyed significant advantage in goal success rate over Replanning, increasing with the likelihood of debilitative failure consequences (Fig. 9); CAMP-BDI maintained around 95% goal achievement for all consequence probability ranges, whilst Replanning dropped from achieving 61.9% of goals at $n = 0.2$, to 26.6% at $n = 0.8$. The Worst-Case system was comparatively consistent, but universally poor; achieving 19.5% to 16% of goals between $n = 0.2$ to $n = 0.6$, before dropping to 8.6% at $n = 0.8$ as failure began to have nearly certain consequential effects. The impact of debilitation was likely reduced at the lower values of n in the Worst-Case system as goals would fail immediately upon the first failure, *regardless* of debilitation; meaning goal failure was less likely be associated with irrecoverable debilitated states at lower n values. Replanning agents, in contracts, would typically only fail in a goal after repeated activity failure – with the individual risks of debilitation accumulating to increase the overall risk of at least *one* debilitation during pursuit of an i_{goal} – and reactive replanning. The Replanning approach was also capable of recovering to achieve the goal despite debilitation in certain cases (such as non-fatal agent damage).

We attribute the increasing performance gap between CAMP-BDI and Replanning to the increasing frequency of debilitation, particularly as damage to agents impacted subsequent activities – with subsequent failures having a compounding debilitative effect. The proactive approach of CAMP-BDI carried two advantages. Firstly, a focus on avoiding failure also avoided (planning and acting in) a less optimal post-failure state. Secondly, when agents *were* damaged, the resultant confidence loss saw maintaining dependants seek alternative, higher-confidence (and consequently undamaged) obligants – reducing the workload on agents with non-optimal health.

Whilst our experimental environment had agents gradually recover health through idleness, we can imagine a similar scenario where repair activities existed; i.e. CAMP-BDI could be used to stimulate damaged agents to proactively repair as soon as activities were deemed at risk, whilst any reactive approach would only address damage after a failure – an event which might in itself exacerbate that damage. Pro-active repair would, however, require capability modelling to link agent health state to greater confidence (potentially with respect to maintenance policy defined thresholds) in activity preconditions.

Finally, the worst-case system saw a less pronounced drop in performance over increasing

n values, due to immediate failure – as opposed to the potential for repeated failures, and accumulated consequences, associated with reactive replanning. This is reflected by the sharp drop in goal achievement as post-failure debilitation became virtually inevitable at $n = 0.8$, mirroring our observations of goal achievement in Fig. 9.

A similar pattern can be observed in Fig. 10, where CAMP-BDI maintained a relatively consistent activity success rates (99.78% to 99.70%) over $n = 0.2$ to $n = 0.8$, compared with decreasing performance of the Replanning (90.90% to 86.66%) and Worst-Case (89.59% to 83.39%) systems. We would naturally expect CAMP-BDI to have a greater activity success due to it's pre-emptive, failure avoidance focus, and these results support the assertion that our proactive approach was effective in preventing activity failure. Activity success rates remained generally high, even in the worst case scenario, as failures generally occurred only after multiple successful activities. The decrease in success rates in Replanning and Worst-Case systems can be seen as indicative of the increasing influence of post-failure debilitation.

One obvious concern with a proactive approach is *cost*, given CAMP-BDI's use of planning. Toyama and Hager [1997] note reactive approaches hold an advantage in only expending their costs following definitive, rather than potential, failure. Indeed, our results show CAMP-BDI performed significantly more planning calls at lower consequence probabilities (Fig. 12); 9.91 calls per goal, compared to 5.62 for Replanning at $n = 0.2$. As the probability of post-failure debilitation increased, reactive replanning became significantly *less* efficient; an average 19.51 planning calls were required for each goal achieved at $n = 0.8$, compared to 11.03 for CAMP-BDI. This reflects the increasing likelihood of debilitation stymieing *reactive* recovery – suggesting maintenance costs can be balanced against those incurred by failure. It may also still be preferable to employ even a higher cost proactive approach, if failure risks sufficiently severe consequences – such as scenarios where delivery goals concern the transport of nuclear waste or essential medical supplies.

A possible optimization for CAMP-BDI is the consideration of temporal thresholds during maintenance – e.g. using maintenance policies or initial configurations to define how many activities in the future an agent should consider during maintenance. This would allow the balancing of maintenance costs for more temporally distant tasks against the value of earlier anticipation and prevention of failure. Exact thresholds would likely be domain specific – depending on elements such as average plan length or the probability of exogenous change between initially anticipating a threat to an activity, and the actual time until it's (expected) execution. It would also be necessary to consider the accuracy of our algorithm in predicting future execution contexts, and how increasing rates of exogenous change might impact the accuracy of context estimation for more distant activities.

We also examined the average number of activities required *per goal achieved* (Fig. 11). Here, CAMP-BDI remained generally consistent, requiring an average of 15.69 to 16.88 activities (from $n = 0.2$ to $n = 0.8$) per goal achieved. Replanning shown a gradual increase in cost, from 21.47 to 29.31 activities per goal. This reflected both the increased difficulty of the environment (greater failure consequences entailed more frequent confidence loss, failure and consequent replanning), and decreasing goal achievement – the number of activities executed *in total* actually decreased from an average 1322.9 at $n = 0.2$ to 700.3 at $n = 0.8$ per experimental run, in contrast to CAMP-BDI's relatively consistent average of 1504.6 to 1564.2 activities for the same n values.

The Worst-Case system showed more variable behaviour, with a noticeable *decrease* in average activities per goal from 28.85 at $n = 0.2$ to 24.93 at $n = 0.4$, before rising again to a maximum of 39.04 at $n = 0.8$. This may be partly a symptom of the extremely low goal success rate in this system, combined with variation in the point in execution at which intention failure occurred. Although the results for $n = 0.6$ show a comparatively modest increase, we note that one run of the Worst-Case system was discounted after failing to achieve *any* goals – meaning an average could not be determined for that particular run, and indicating our actual average could be far higher (although not observed in our experimental runs, the same scenario could also feasibly occur in the more difficult $n = 0.8$). In this context of extremely low goal achievement, the Worst Case value may not necessarily indicate the cost to achieve goals, but how early or late failure typically occurred during i_{plan} execution – the latter being perhaps more random than in the Replanning and CAMP-BDI cases, which would at least *attempt* to respond to actual or anticipated failure.

Our overall results show CAMP-BDI offers a clear advantage when activity failure carries a risk of debilitative failure. Although proactive approaches may risk excess cost from false-positive identification of (potential) failure, these results indicate this cost can be effectively mitigated where post-failure debilitation stymies reactive failure recovery or the achievement of subsequent goals. However, it is infeasible to expect that a proactive system will be able to detect and avoid *every* failure in a realistic environment – particularly if failure-causing exogenous change may occur *during* activity execution.

Some form of reactive failure recovery strategy is always likely to be required to compensate for random, 'unpreventable' failure cases. We suggest CAMP-BDI can complement reactive approaches, by being used to target those failures which *are* preventable and may be difficult to recover from. Our maintenance policy concept also allows for a degree of optimisation; such as by reducing confidence thresholds for activities with lower risks of post-failure debilitation and allowing their failure to instead be handled reactively This would remove the iterative costs of maintenance, under the basis that for such activity types is should be possible to easily recover from any failure.

7 Related Work

CAMP-BDI draws from a variety of existing work; our capability model captures concepts of *know-how-to-perform*, *can-perform* and *know-how-to-achieve* defined by Morgenstern [1986] as well as similar concepts of activity knowledge expressed in Singh [1999]. Plan confidence estimation adopts an approach akin to (a subset of) TÆMS quality metrics (Lesser *et al.* [2004b]), such as *q_min*; future work may investigate alternate methods. He and Ioerger [2003] also suggest a quantitative estimation approach, but employed for producing maximally efficient schedules. Other work has also examined using capability knowledge within BDI reasoning process; Sabatucci *et al.* [2013] suggests use of capabilities representing plans and viability conditions to evaluate whether desires are achievable during intention selection. Waters *et al.* [2014] suggest an intention selection approach prioritizing the most constrained options by favouring those with least *coverage* (Thangarajah *et al.* [2012]). This differs from CAMP-BDI as they seek to maximize overall intention throughput, whilst our approach aims to ensure existing intended goal are achieved. Our capability knowledge model *could* facilitate similar reasoning during desire and intention selection, although our work focuses upon robustness of *selected* intentions.

Plan Execution Monitoring (PEM) approaches, such as SIPE (Wilkins [1983]), and plan repair approaches, such as O-Plan (Drabble *et al.* [1997]), share conceptual similarities with CAMP-BDI as both respond to divergence from expected states through replanning or plan repair. Typically, the latter is favoured in *distributed* plans due to offering greater plan stability and, as a consequence, reduced messaging costs (for communicating plan changes to others). CAMP-BDI differs by focusing upon BDI agent reasoning, and extension of the resultant local agent-level behaviour to perform distributed maintenance; our confidence estimation approach also varies from PEM approaches, which typically assess deterministic (pre)conditions to identify divergences from expected and actual states. We can also view our approach as similar to usage of synthesized *protection monitors* in CPEF (Myers [1999]), which also acts to detect violation of states required by the current plan.

Braubach *et al.* [2005] define two goal types for driving agent proactivity; *achievement* of a state, or *maintenance* of it for a defined period or whilst set conditions hold. Duff *et al.* [2006] further distinguish reactive and proactive types of maintenance goal; the former requires re-establishment of the state once violated, the latter constrains goal and plan adoption to prevent it's violation. The reactive case will stimulate adoption of achievement goals to re-establish violated (maintained) states; CAMP-BDI could be used to maintain the resultant intentions. Precondition maintenance in CAMP-BDI is similar in outcome to inferring proactive maintenance goals, corresponding to activity precondition states and active until that activity begins execution. Effects maintenance can be viewed as somewhat similar, in that loss of high-confidence associated states will trigger plan modification; although our approach does not necessarily entail *re-establishment* of states if maintenance planning can identify acceptable alternative activities. We assume any mechanisms used for identifying plans for intentions, and/or to find maintenance plans, would recognise and respect maintenance goals.

Hindriks and Van Riemsdijk [2007] suggests an approach which, similarly to CAMP-BDI, employs a (limited) lookahead – in their case with regard to respecting proactive maintenance goals. A goal-plan tree is used to anticipate future effects of plans, to avoid intention of plans that would violate maintenance goals. Plans in this approach are pre-defined and immutable; anticipated violation is suggested as best addressed by goal relaxation to allow alternative plan options. This differs from our view of plans as modifiable, and may not be a viable approach in domains where goals *cannot* be relaxed (i.e. due to *safety responsibilities* towards preserving or ensuring certain states, as defined by Wooldridge *et al.* [2000]). Duff *et al.* [2006] suggest a predictive approach, again using a goal-plan tree to filter goal adoption based upon effects of potentially usable plans. CAMP-BDI varies by more explicitly considering exogenous change, rather than effects from goal/plan adoption, as a source of violation. Our approach also focuses upon ensuring *existing* intentions avoid failure after exogenous change – proactive maintenance goals are typically employed more as constraints upon the *formation* and adoption of desires or intentions (although this does also influence any subgoal refinement within continual approaches).

Continual planning handles uncertainty by deferring planning decisions (desJardins *et al.* [2000]) – including decomposing certain abstract activities only upon execution. CAMP-BDI supports this approach through composite capabilities; these allow determination of whether undecomposed (sub)goals can be met, by representing the set of available plans and providing an estimation of what level of confidence should be expected from the plans represented by that capability. Where planning incorporates sensing – representing knowledge requirements through preconditions, and information attainment through effects – these can be represented within capabilities.

Markov Decision Processes (MDPs) offer an approach for acting within stochastic domains; they use state transition probabilities and a reward function to generate a *policy*, which defines the optimal activity to perform in each possible state. Partially Observable MDPs (POMDPs) remove total knowledge assumptions through a probability map of state observations, which is used to infer actual states and define a solveable MDP. Whilst MDP approaches theoretically offer optimal behaviour, complexity issues render them intractable as state space increases (and particularly when extending to realistic environments), with attempts to address this issue focusing on abstracting the state space at the cost of overall optimality (Boutilier and Dearden [1994]). There is also a risk that the transition probability information required to define and solve an MDP problem is unavailable, or impractical to learn under reasonable time constraints.

The BDI model can be viewed as a more efficient alternative to MDP approaches; Schut *et al.* [2002] have shown BDI agents can handle domains which are intractable for MDPs, and with approximate performance (depending on time costs of runtime planning) to MDPs. Work has also sought to reconcile BDI and MDP approaches; Simari and Parsons [2006] identify similarities and suggest possible mapping between policies and plans. Pereira *et al.* [2008] extend that work by defining an algorithm to form deterministic plans (for libraries) from POMDP policies – although this assumes the latter can be formed offline, which may not hold given the known intractability issues with POMDPs. MDP specification of a domain can also be non-intuitive, restricting it's usability; Meneguzzi *et al.* [2011] suggest a method to map more intelligible HTN domains onto MDPs, where they define transition probabilities based upon the presence of individual states within operator preconditions, rather than through probabilities in the *environment*. In defining CAMP-BDI, we assume realistic domains will necessitate use of deterministic plans and operator specifications, due to the intractability issues associated with MDPs. Our confidence estimation does resemble the information provided by MDP transition functions, but we only require a scalar *estimate* of quality (with flexible granularity) rather than an *exact* probability.

8 Conclusion

This paper contributes the CAMP-BDI approach for distributed plan execution robustness. We defined a pre-emptive plan modification (*maintenance*) algorithm, including provision capability and contract meta-knowledge, and tailored through provision of maintenance policies. This local behaviour was extended to the distributed case through a structured messaging approach. Although we do not argue that CAMP-BDI, or any proactive approach, can entirely replace reactive methods – that all failures can be avoided – it can offer a valuable complementary approach.

Our approach does entail an analytical cost in learning and modelling capability knowledge, which must be balanced against the likelihood and severity of post-failure debilitation. We argue the domain analysis requirements for capability modelling are justifiable, as specification of plans or planning domains – particularly operator preconditions – would require similar knowledge of which states impacting activity success *and* with what significance, regardless. Specification costs are also mitigated by our confidence model not requiring *exact* probability estimation, but only an appropriate indicative value – with granularity being limited to whatever level is feasible for the domain. Capability knowledge may also have applications in other robustness approaches, or for desire and intention selection – helping justify analysis and specification costs.

In future, we intend to examine methods to minimize planning costs and further extend the use of maintenance policies. Potential expansions can include defining maintenance policy fields that specify proactive maintenance goals, allow relaxations during maintenance planning, provide conditional rules denoting where maintenance is intractable (allowing handling to be deferred to dependants instead of performing futile planning), or to better focus upon activities whose failure is associated with greater risks of debilitation. Another possible extension is to allow definition of conditional response rules for maintenance planning – i.e. to define specific known maintenance plan recipes for given conditions, allowing circumvention of the cost of runtime planning. We may also investigate the possible use of heterogeneous planning approaches, as our algorithms do not prescribe a specific plan formation method. For example, more specific agents could adopt HTN or plan library solutions to improve reactive speed, with upper level (more abstract, logical organizer or broker types) employing more flexible classical planners when confidence loss could not be addressed by those lower in the team hierarchy. A final area for potential optimization is confidence estimation and agenda formation, although the most effective approaches are likely to be domain-specific.

Acknowledgements

This work was funded with support from EADS Innovation Works. Alan White would like to extend additional thanks to Dr. Stephen Potter for his invaluable help and advice. The authors and project partners are authorized to reproduce and distribute reprints and online copies for their purposes, notwithstanding any copyright annotation hereon.

Notes

[1]Maintenance policies are not related to the MDP policy concept, but derive from those employed in experiments such as the Coalition Agents Experiment (Allsopp *et al.* [2002]). There, policies provide runtime-modifiable extension of planning constraints; as with CAMP-BDI, they allow a degree of dynamic modification to agent and system behaviour.

[2]*FF-Replan* determinizes a probabilistic domain to take advantage of historical optimizations in classical planning. Differences in actual and anticipated (by classical operator) outcomes are handled through reactive replanning. Our reactive system effectively used single-outcome determinization where success was the most probable outcome if preconditions held. Failure triggered re-planning, mirroring *FF-Replan*'s eponymous response to unexpected outcomes.

References

D. N. Allsopp, P. Beautement, J. M. Bradshaw, E. H. Durfee, M. Kirton, C. A. Knoblock, N. Suri, A. Tate, and C. W. Thompson. Coalition Agents Experiment: Multiagent Cooperation in International Coalitions. *IEEE Intelligent Systems*, 17(3):26–35, 2002.

R.H. Bordini and J.F. Hübner. BDI Agent Programming in AgentSpeak Using Jason. In F. Toni and P. Torroni, editors, *Computational Logic in Multi-Agent Systems*, volume 3900 of *Lecture Notes in Computer Science*, pages 143–164. Springer Berlin Heidelberg, 2006.

C. Boutilier and R. Dearden. Using Abstractions for Decision Theoretic Planning with Time Constraints. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1016–1022. San Francisco, CA: Morgan Kaufmann, 1994.

L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In R.H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3346 of *Lecture Notes in Computer Science*, pages 44–65. Springer Berlin Heidelberg, 2005.

M.E. desJardins, E.H. Durfee, C.L. Ortiz Jnr., and M.J. Wolverton. A Survey of Research in Distributed, Continual Planning, 2000.

B. Drabble, J. Dalton, and A. Tate. Repairing Plans On-the-fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, 1997.

S. Duff, J. Harland, and J. Thangarajah. On Proactivity and Maintenance Goals. In *AAMAS-06*, pages 1033–1040, 2006.

M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *In Proc. ICAPS*, pages 212–221. AAAI Press, 2006.

A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs. In *In Proc. of 6th Int. Conf. on AI Planning Systems (AIPS'02*. AAAI Press.

L. He and T.R. Ioerger. A Quantitative Model of Capabilities in Multi-Agent Systems. In *Proceedings of the International Conference on Artificial Intelligence, IC-AI '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 2*, pages 730–736, 2003.

K.V. Hindriks and M.B Van Riemsdijk. Satisfying maintenance goals. In *IN PROC. OF DALT'07*. Springer, 2007.

A. Komenda, P. Novák, and M. Pechoucek. Domain-independent multi-agent plan repair. *J. Network and Computer Applications*, 37:76–88, 2014.

V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Nagendra Prasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TÆMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, 2004.

J. McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanisation of Thought Processes*, pages 77–84, 1958.

F. Meneguzzi, Y. Tang, K. Sycara, and S. Parsons. An approach to generate MDPs using HTN representations. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities*, Barcelona, Spain, 2011.

L. Morgenstern. A First Order Theory of Planning, Knowledge, and Action. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, TARK '86, pages 99–114, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

K. L. Myers. Cpef: A continuous planning and execution framework. *AI Magazine*, 20(4), 1999.

D.R. Pereira, L.V. Gonçalves, G.P. Dimuro, and A.C.R. Costa. Constructing BDI plans from optimal POMDP policies, with an application to AgentSpeak programming. In *Proc. of Conf. Latinoamerica de Informática, CLEI*, volume 8, pages 240–249, 2008.

A.S. Rao and M.P. Georgeff. BDI Agents: From Theory to Practice. In *In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

L. Sabatucci, M. Cossentino, C. Lodato, S. Lopes, and V. Seidita. A Possible Approach for Implementing Self-Awareness in JASON. In *EUMAS'13*, pages 68–81, 2013.

M. Schut, M. Wooldridge, and S. Parsons. On Partially Observable MDPs and BDI Models. In *Selected Papers from the UKMAS Workshop on Foundations and Applications of Multi-Agent Systems*, pages 243–260, London, UK, UK, 2002. Springer-Verlag.

G.I. Simari and S. Parsons. On the Relationship Between MDPs and the BDI Architecture. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1041–1048, New York, NY, USA, 2006. ACM.

D. Singh, S. Sardinia, L. Padgham, and S. Airiau. Learning Context Conditions for BDI Plan Selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 325–332, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

M.P. Singh. Know-How. In M. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*, volume 14 of *Applied Logic Series*, pages 105–132. Springer Netherlands, 1999.

J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference Between Goals in Intelligent Agents. In *IJCAI-03*, pages 721–726, 2003.

J. Thangarajah, S. Sardina, and L. Padgham. Measuring Plan Coverage and Overlap for Agent Reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1049–1056, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

K. Toyama and G. Hager. If at First You Don't Succeed... In *Proc. AAAI*, pages 3–9, Providence, RI, 1997.

M. Waters, L. Padgham, and S. Sardina. Evaluating Coverage Based Intention Selection. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 957–964, Paris, France, May 2014. IFAAMAS. Nominated for Jodi Best Student Paper award.

D. E. Wilkins. Representation in a Domain-Independent Planner. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*, pages 733–740, 1983.

M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

S. W. Yoon, A. Fern, and R. Givan. FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, page 352, 2007.

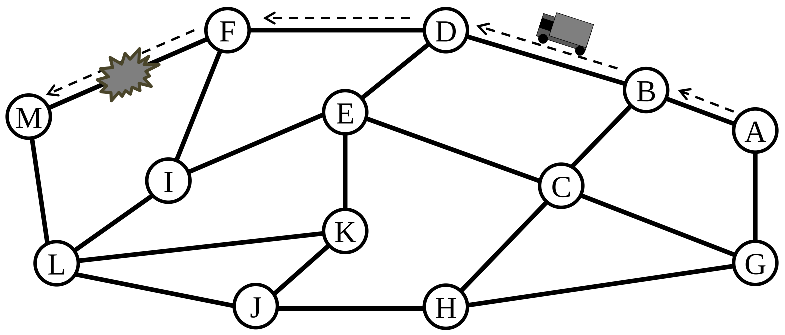


Fig. 1: Example of *Truck* executing a plan to travel from *A* to *M*.

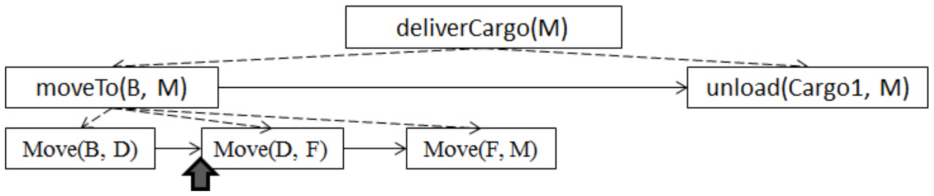


Fig. 2: Example where *Truck*'s plan to *deliverCargo* is threatened by violated preconditions of *Move*(D, F), indicated by the arrow, following closure of $D \rightarrow F$.

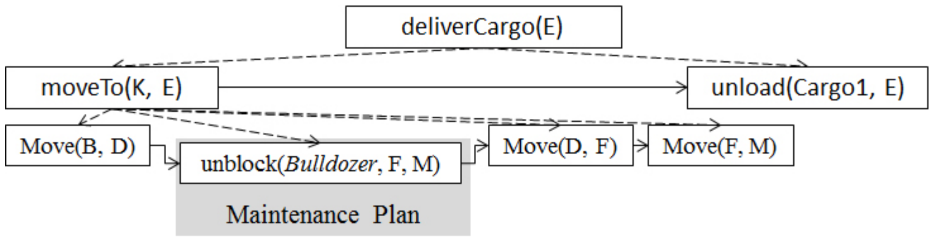


Fig. 3: Example insertion of a successfully identified maintenance plan, restoring the preconditions of the threatened *Move*(D, F) through *unblock* clearing $D \rightarrow F$.

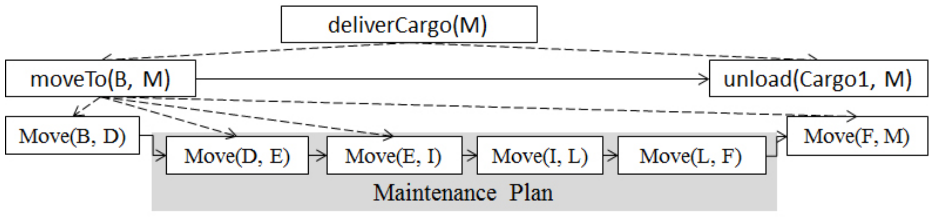


Fig. 4: Example insertion of a maintenance plan as a substitute for *Move*(D, F), which achieves the same goal state (being at location F) from the estimated execution context (including start location) of *Move*(D,F).

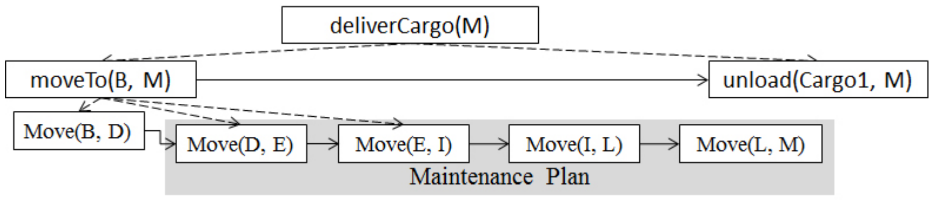


Fig. 5: Example insertion of a successfully identified maintenance plan in the suffix case; the initial context of the threatened *Move*(D, F) is employed as the initial state for planning, with the goal defined as that of the parent *MoveTo*(B, M) activity

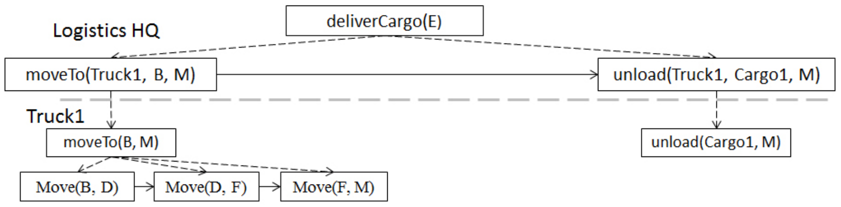


Fig. 6: Example of a distributed intention, where *Truck1* holds an obligation to perform the two activities *moveTo*(B, M) and *unload*$(Cargo1, M)$.

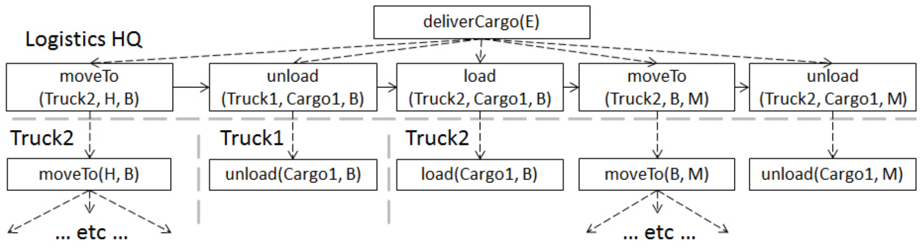


Fig. 7: Result of adoption of maintenance responsibility by *Logistics HQ* in response to low confidence in *Truck1*'s obligation (Fig 6). The new obligant *Truck2* originates at a different initial location and must first travel to B to retrieve *Cargo1* – which was previously carried by, and now must be unloaded by, *Truck1*.

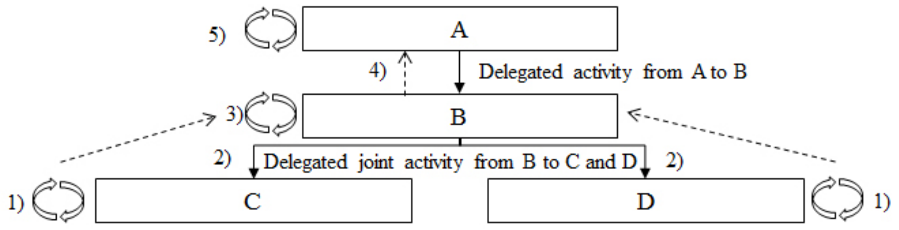
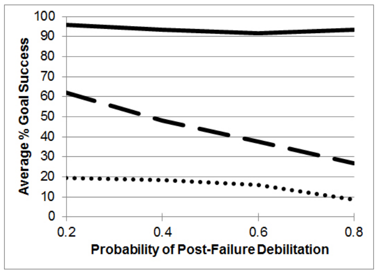
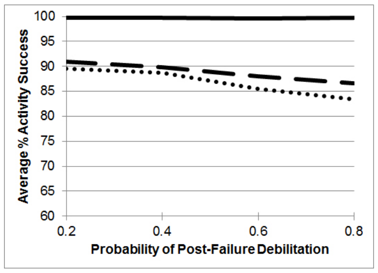

5)
A
4)
Delegated activity from A to B
3)
B
2)
Delegated joint activity from B to C and D
2)
1)
C
D
1)

Fig. 8: The adoption of responsibility process in a hierarchical team, where B is an obligant of A, and C and D are obligants for a joint activity in B's plan.



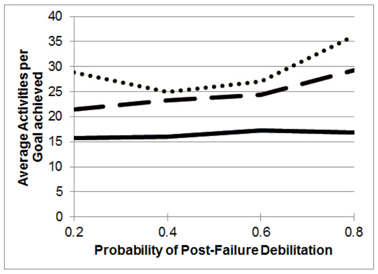
	CAMP-BDI	Replanning	Worst Case
0.8 Avg	93.3	26.6	8.6
(Std.dev.)	7.99	9.2	4.34
0.6 Avg	91.6	37.5	16
(Std.dev.)	7.57	8.95	10.02
0.4 Avg	93.3	47.9	18.33
(Std.dev.)	2.69	9.25	9.18
0.2 Avg	95.9	61.9	19.5
(Std.dev.)	2.62	4.13	13.37

Fig. 9: Average goal achievement rate (%) for 0.2 to 0.8 post-failure damage probability, with standard deviation. CAMP-BDI results are shown as solid lines, Replanning dashed, and Worst-Case as dotted.



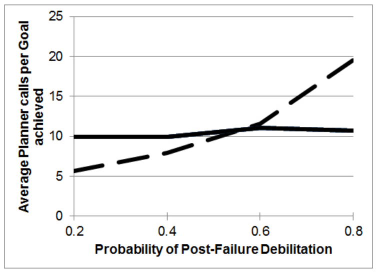
	CAMP-BDI	Replanning	Worst Case
0.8 Avg	93.3	86.66	83.39
(Std.dev.)	7.99	2.54	5.4
0.6 Avg	91.6	88.03	85.46
(Std.dev.)	7.57	1.62	5.29
0.4 Avg	93.3	89.83	88.72
(Std.dev.)	2.69	1.83	2.06
0.2 Avg	95.9	90.91	89.59
(Std.dev.)	2.62	1.25	3.6

Fig. 10: Average activity success (%), for 0.2 to 0.8 post-failure damage probability, with standard deviation. CAMP-BDI results are shown as solid lines, Replanning dashed, and Worst-Case as dotted.



	CAMP-BDI	Replanning	Worst Case
0.8 Avg	16.88	29.31	36.04
(Std.dev.)	1.66	9.19	11.35
0.6 Avg	17.23	24.41	26.98
(Std.dev.)	1.59	5.34	8.89
0.4 Avg	16	23.27	24.93
(Std.dev.)	0.81	3.58	8.61
0.2 Avg	15.7	21.47	28.85
(Std.dev.)	0.42	1.67	16.78

Fig. 11: Average Activities per Goal, for 0.2 to 0.8 post-failure damage probability, with standard deviation. CAMP-BDI results are shown as solid lines, Replanning dashed, and Worst-Case as dotted.



	CAMP-BDI	Replanning
0.8 Avg	10.7	19.51
(Std.dev.)	1.78	10.22
0.6 Avg	11.06	11.55
(Std.dev.)	1.57	4.27
0.4 Avg	9.93	7.88
(Std.dev.)	3.29	2.99
0.2 Avg	9.91	5.62
(Std.dev.)	1.37	1.45

Fig. 12: Average planner calls per goal achieved for 0.2 to 0.8 post-failure damage probability, with standard deviation. CAMP-BDI results are shown as solid lines and Replanning as dashed.