Intention Interleaving Via Classical Replanning

Mengwei Xu, Kevin McAreavey, Kim Bauters, Weiru Liu
School of Computer Science, Electrical and Electronic Engineering, and Engineering Maths
University of Bristol, Bristol, UK
{mengwei.xu, kevin.mcareavey, kim.bauters, weiru.liu}@bristol.ac.uk

***Abstract*—The BDI architecture, where agents are modelled based on their belief, desires, and intentions, provides a practical approach to developing intelligent agents. One of the key features of BDI agents is that they are able to pursue multiple intentions in parallel, i.e. in an interleaved manner. Most of the previous works have enabled BDI agents to avoid negative interactions between intentions to ensure the correct execution. However, to avoid execution inefficiencies, BDI agents should also capitalise on positive interactions between intentions. In this paper, we provide a theoretical framework where first-principles planning (FPP) is employed to manage the intention interleaving in an automated fashion. Our FPP approach not only guarantees the achievability of intentions, but also discovers and exploits potential common sub-intentions to reduce the overall cost of intention execution. Our results show that our approach is both theoretically sound and practically feasible. The effectiveness evaluation in a manufacturing scenario shows that our approach can significantly reduce the total number of actions by merging common sub-intentions, while still accomplishing all intentions.**

***Index Terms*—BDI Agents, Intention Interleaving, Planning**

I. Introduction

Belief-Desire-Intention (BDI) [1] is one of the most popular agent development models and forms the basis of, among others, AgentSpeak [2], 3APL [3], CANPLAN [4], 2APL [5], and Jason [6]. In a BDI agent the (B)eliefs represent what the agent knows, the (D)esires what the agent wants to bring about, and the (I)ntentions those desires the agent has chosen to act upon. In all of the works listed above the desires and intentions are represented implicitly through a plan library. Each plan describes how an agent can react to an (external/internal) event under specific conditions, and the set of intentions are those plans that are currently being executed. Both beliefs and plan libraries are well-studied, but intentions – a crucial part of BDI agents – are one of the least studied areas in BDI theory [7].

For an agent to successfully handle its intentions, a variety of tasks need to be completed. These include issues such as intention refinement (to reduce a high-level intention into actionable steps), intention revision (which intentions to drop/replace *e.g.* [8]), and intention progression (which intention to progress next *e.g.* [9]). In this paper, we address the problem of intention interleaving, where we are interested in identifying and exploiting overlapping programs (e.g. common actions) between different intentions. A desirable property of any agent-based system is that the system is *reactive*; the agent can respond to new events even while already dealing with other events. To this end, intentions are executed in an interleaved manner. When an agent is pursuing multiple intentions in parallel, it is critical for the agent to avoid negative interference between intentions, i.e. conflict resolution [10], [11]. However, to avoid execution inefficiency, the agent also should capitalise on positive interactions between intentions. Opportunities for positive interactions between intentions enable the agent to reduce the effort (e.g. resources) it exerts to accomplish its intentions. In particular, positive interactions exist when intentions overlap with each other. In this case, the agent with the overlapping intentions can merge its intentions (effectively allowing one to skip some of its plan steps in its plan) to reduce the overall execution cost.

To illustrate the problem, consider a BDI implementation for a Mars Rover agent. The agent has a goal to transmit soil experiment results and a goal to transmit image collection results. The agent could perform the goals sequentially by *establishing the connection* with the Earth, sending soil experiment results, *breaking the connection*, then *establishing the connection* with the Earth, sending image collection results, and *breaking the connection*. Alternatively, it could *establish the connection* with the Earth, send both the soil experiment and image collection results, and *break the connection*. Clearly, the second approach manifests a more sensible and intelligent behaviour, and is accomplished by the agent being able to discover and exploit the commonality of different intentions. While, unlike conflict resolution, exploiting commonality of intentions is not necessary for the agent to perform its tasks correctly, it can be of vital importance in a resource-critical domain such as in the autonomous manufacturing sector [12].

Within the BDI community, few papers have discussed how to address these issues. One motive for this is that there has been a focus on a simple intention selection mechanism that favours highly efficient reasoning cycle above all else. Still, recently, a number of approaches on dealing with positive interactions between multiple intentions in parallel have been released. In works of [13], [14], the authors propose a way to detect and exploit positive intention interactions by reasoning about definite and potential pre-conditions and post-effects of plans and goals. However, this approach is limited to intention merging at the plan level to avoid duplicate plan executions, thus ignoring the merging of individual steps (e.g. actions) within plans. As a result, the approach needs to adopt a conservative strategy where the merging is allowed if and only if the definite and potential effects of one plan is completely subsumed by the others to preserve correct intention execution.

This leaves the agent with a brittle mechanism to detect potential overlapping intentions and attempting to schedule its actions to take advantage of them. Instead, we show that within

2375-0197/19/$31.00 ©2019 IEEE
DOI 10.1109/ICTAI.2019.00021

a BDI context, as a high-level modelling language, many of these intention issues can be resolved through planning in an automated fashion. We accomplish this by showing how intentions (particularly the complete intention execution traces, each of which leads to a successful execution of an intention) can be modelled as the search space of a PDDL problem description [15] (the de-facto standard planning language). Subsequently, planning is employed to identify a *conflict-free* (i.e. correct execution) and *maximal-merged* (i.e. minimal effort) execution trace. The approach we introduce is agnostic to the actual planner being used, thus implying our approach can be used with modern highly efficient online planners (e.g. [16]) to execute plans until it is necessary to replan.

In this paper, we propose a planning-based extension to BDI where the planning is used to exploiting overlapping intentions while resolving conflicts during interleaved execution of intentions. To this end, we (i) formalise the intention of a BDI agent as an AND/OR graph; (ii) define all potential and complete execution traces of a set of intentions; (iii) compute all potential overlapping programs among a set of intentions; (iv) present our planning-based approach; (v) provide the implementation; and, finally, (vi) present some effectiveness evaluation in manufacturing test beds of increasing sizes.

The remainder of the paper is organised as follows: in Section II, we recall preliminaries on BDI agents, AND/OR graphs, and first-principles planning (FPP); in Section III, we describe our FPP framework; in Section IV and Section V, we present implementation and evaluation; in Section VI, we discuss related work; and, in Section VII, we conclude.

II. PRELIMINARIES

In this section, we recall necessary preliminaries on BDI agents, AND/OR graphs, and first-principles planning (FPP). We rely on some standard mathematical notation: 2^S is the power set of S and $\mathbb{R}$ is the set of real numbers.

A. BDI Agent

A BDI agent is a tuple $\langle \mathcal{B}, \Pi, \Lambda \rangle$ with $\mathcal{B}$ a belief base, Π a plan library, and Λ an action library. The belief base $\mathcal{B}$ is a set of formulas encoding the current beliefs, with $\mathcal{B} \models \varphi$ denoting that the sentence φ is true according to belief base $\mathcal{B}$. The *plan library* is a collection of plans of the form $P = G : \varphi \leftarrow h_1; ...; h_n$. We say that G is the *head* or *goal*, φ is the *context*, and $h_1; ...; h_n$ is the body of the plan P. For ease of reference we also refer to these as *head*(P), *context*(P), and *body*(P), respectively. The body $h_1; ...; h_n$ is a sequence such that each h_i, $1 \leq i \leq n$, is either an action or a (sub)goal. For a given component h and a plan P we define a body order function $O(h, P) = i$ to retrieve the position of such component in the plan body. We say that a plan P is *relevant* (resp. *applicable*) for dealing with a goal G when $head(P) = G$ (resp. when $\mathcal{B} \models context(P)$). We also use $\mathcal{G}$ to denote the set of (sub)goals (i.e. the head) in the plan library. The *action library* is a set of actions, each of the form $a = \langle \psi, \phi^-, \phi^+ \rangle$ where ψ is a precondition, ϕ^- and ϕ^+ are delete and add sets of atom. Revising a belief base $\mathcal{B}$ with an effect $\mathit{eff} = \langle \phi^-, \phi^+ \rangle$, written as $\mathcal{B} \circ \mathit{eff}$, is defined as $(B \setminus \phi^-) \cup \phi^+$. For simplicity, we consider a BDI agent that is programmed relative to some finite propositional language and its plan library does not have recursive plans.

B. AND/OR Graphs

A directed graph is a tuple (N, E) where N is a set of nodes and $E \subseteq N \times N$ is a set of directed edges. A multigraph is a tuple (N, L, E') where L is a set of labels and $E' \subseteq N \times L \times N$ is a set of multiedges such that for each $l \in L$ we have that $(N, \{(n, n') \mid (n, l, n') \in E'\})$ is a graph. We say n' is a child of n, written as $n' \in child(n)$ iff $(n, l, n') \in E'$ for some $l \in L$. Given nodes $n_1, n_{m+1} \in N$ in a multigraph, then a sequence of nodes and labels $(n_1, l_1, \ldots, n_m, l_m, n_{m+1})$ is a path from n_1 to n_{m+1} iff each n_j is unique and $(n_j, l_j, n_{j+1}) \in E'$ for $j = 1, \ldots, m-1$. A multi-graph is acyclic if, for each $n \in N$, there exists no path from n to itself. A rooted multigraph is a tuple $(N, L, E', \bar{n})$ where (N, L, E') is a multigraph and $\bar{n} \in N$ is a root node such that for each $n' \in N \setminus \{\bar{n}\}$, there exists a path from $\bar{n}$ to n'. An AND/OR graph $(N_\vee \cup N_\wedge, L_\vee \cup L_\wedge, E_\vee \cup E_\wedge, \bar{n})$ is a rooted acyclic multigraph where $N_\vee$ (resp. $N_\wedge$) is a set of OR-nodes (resp. AND-nodes), $L_\vee$ (resp. $L_\wedge$) is a set of OR-labels (resp. AND-labels), $E_\vee \subseteq N_\vee \times L_\vee \times N_\wedge$ (resp. $E_\wedge \subseteq N_\wedge \times L_\wedge \times N_\vee$) is a set of OR-edges (resp. AND-edges), and $\bar{n} \in$ is a root node.

C. First-Principles Planning

A problem in first-principles planning (FPP), also known as classical planning, is defined as $\mathcal{S} = \langle S, s_0, S_G, O, f, r \rangle$ where S is a finite and discrete set of states, s_0 is the initial state, and S_G is the non-empty set of goal states. $O(s) \subseteq O$ represents the set of operators in O that are applicable in each state $s \in S$. $f(\alpha, s)$ is the transition function, i.e. the state which follows state s after applying operator $\alpha \in O(s)$. Finally, $r(\alpha, s)$ is the reward for applying operator α in state s. A solution to $\mathcal{S}$ is a sequence of applicable operators $\alpha_0; \ldots; \alpha_n$ that generates a state sequence $s_0; s_1; \ldots; s_{n+1}$ where s_{n+1} is the goal state. The reward of the solution is the sum of the operator reward $r(\alpha_i, s_i) \in \mathbb{R}, i = 0, \ldots, n$. A solution is optimal if it has the maximum reward.

III. FRAMEWORK

In this section, we formally define the goal-plan trees to model the intentions of a BDI agent, and we use these goal-plan trees in Section III-A to define the *conflict-free* and *maximal-merged* execution traces of intentions. In Section III-B and Section III-C, we outline a theoretical approach where planning is used to manage the intention interleaving in a way that maximises the intention merging while guaranteeing the achievability of all intentions.

A. Intention Formalisation

In BDI agent systems, the so-called goal-plan trees have been the canonical representation of intentions [14]. The root of a goal-plan tree is a top-level goal, and its children are plans that can be used to achieve such a top-level goal. Plans may also contain subgoals, giving rise to a tree structure representing all possible ways of achieving the goal. In this paper, we also use it to represent the underlying hierarchy in the plan library. We now give the definition of a goal-plan tree [14] which we formalise and simplify in this paper. Recall Π is a set of plans, Λ a set of actions, and $\mathcal{G}$ a set of (sub)goals.

Definition 1. *A goal-plan tree for an intention in a BDI agent to achieve a top-level goal $G \in \mathcal{G}$ is an AND/OR graph $T = (N_\vee \cup N_\wedge, L_\vee \cup L_\wedge, E_\vee \cup E_\wedge, \bar{n})$ where:*

1) $\bar{n} = G$ *(i.e. the top-level goal);*
2) $N_\vee \subseteq \mathcal{G} \cup \Lambda$ *(i.e. sub-goals or individual actions);*
3) $N_\wedge \subseteq \Pi$ *(i.e. plans to deal with goals);*
4) $L_\vee = \mathcal{L}$ *(i.e. the logical language);*
5) $L_\wedge \subseteq \mathbb{N}$*;*
6) $(G, \varphi, P) \in E_\vee$ *if* $P \in \Pi$ *such that* $head(P) = G$ *and* $context(P) = \varphi$*;*
7) $(G', \varphi', P') \in E_\vee$ *if there exists a path from* G *to* G' *with* $G' \in \mathcal{G}$ *such that* $P' \in \Pi$ *with* $head(P') = G'$ *and* $context(P') = \varphi'$*;*
8) $(P, j, h) \in E_\wedge$ *if there exists a path from* G *to* P *with* $P \in \Pi$ *such that* $O(h, P) = j$*;*

The root node of a goal-plan tree[1] is the top-level goal (1). Criterion (2) and (3) assign the BDI components to the nodes. Criterion (6) and (7) link a goal with its relevant plans using OR-edges (labelled (4) with the context of the corresponding plan), while (8) links a plan with its body using AND-edges (labelled (5) with a natural number which indicates the execution order). We now present an example and graphical illustration to explain the concepts in Definition 1 as follows:

Example 1. Let G_1 and G_2 be the goals of our Mars Rover to transmitting the soil experiment results and transmitting the image collection results, respectively. We have plan P_1 and P_2 to achieve G_1 and G_2 given as follows:

$$P_1 = G_1 : \varphi_1 \leftarrow a_1; a_2; a_4; \quad P_2 = G_2 : \varphi_2 \leftarrow a_1; a_3; a_4;$$

where φ_1 and φ_2 are the context of P_1 and P_2, respectively. The action a_1 (resp. a_4) stands for establishing (resp. breaking) the connection. Meanwhile, the action a_2 (resp.a_3) denotes transmitting the soil experiment (resp. image collection) results. The corresponding goal-plan trees are presented in Figure 1, which are constructed according to Definition 1 above.

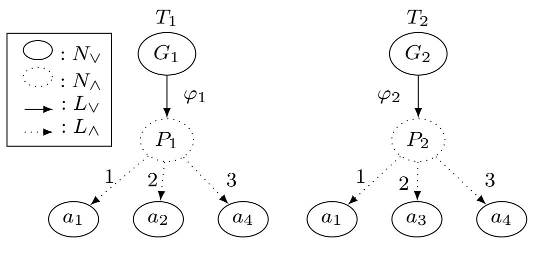

Fig. 1. AND/OR Graphs for Goal-plan Trees.

[1]Note that despite still being called a goal-plan tree here, it does not satisfy the definition of a tree as it is defined as a graph in this work.

We now look at the problem of intention interleaving in the context of goal-plan trees, as a BDI agent is typically pursuing multiple goals in parallel. We start with the definition of the *execution trace* of a single intention, which identifies every unique way in which a given intention can be achieved.

Definition 2. *Let T be a goal-plan tree. An execution trace of T is defined to be $\tau(T) = \tau(T(\bar{n}))$ such that*

1) $\tau(G) = G; \tau(P)$ *s.t.* $head(P) = G$*;*
2) $\tau(P) = P; \tau(h_1); \ldots; \tau(h_n)$ *s.t.* $body(P) = h_1; ...; h_n$*;*
3) $\tau(a) = a$*;*

where $T(\bar{n})$ denotes the top-level goal of T, $a, P, G \in T(N_\vee \cup N_\wedge)$ (i.e. the AND/OR nodes of T). We also denote the set of all execution traces of a goal G by $\omega(G)$, i.e. $\tau(G) \in \omega(G)$.

Definition 2 says that an execution trace of an intention is an execution trace of its top-level goal. An execution trace of a goal is the sequence beginning with the adoption of such goal followed by the execution trace of *one* of its relevant plans (1). The execution trace of a plan consists of the plan identifier (which stands for the selection of the plan) followed by the trace of the individual element of its body (2). Finally, the execution trace of action is trivially the action itself (3).

Example 2. Consider the goal-plan tree T_3 in Figure 2. It has two execution traces, namely $\tau_1(T_3)$ and $\tau_3(T_3)$ as follows:

$$\tau_1(T_3) = G_3; P_3; a_4; a_5; \qquad \tau_2(T_3) = G_3; P_4; b_4; b_5; b_6;$$

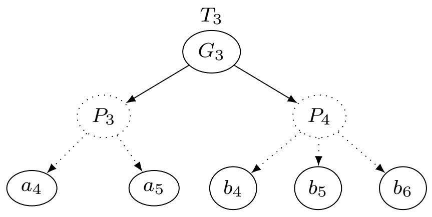

Fig. 2. A Goal-plan Tree with Two Relevant Plans.

So far we have defined the execution trace of a single intention. We now define an execution trace of a set of intentions $\{T_1, \cdots, T_m\}$. Recall $T_j(\bar{n})$ is the top-level goal of T_j and $\omega(T_j(\bar{n}))$ is the set of all execution traces of $T_j(\bar{n})$.

Definition 3. *An execution trace of a set of intentions $\{T_1, \ldots, T_m\}$ is any sequence σ obtained by interleaving a finite number of execution traces from the set of $\bigcup_{j=1}^{m} \omega(T_j(\bar{n}))$ such that $\mid \{i \mid \sigma[i] = T_j(\bar{n})\} \mid = 1$ where $\sigma[i]$ denotes the i^{th} element of σ and $1 \leq j \leq m$.*

Definition 3 says that the construction of an execution trace of a set of intentions is to interleave elements in the execution traces of different intentions. The requirement on the cardinality of the top-level goal of each intention ensures that there is one and only one execution trace of each intention being interleaved with the execution traces of other intentions (i.e. each intention only needs to be achieved once).

Example 3. In Figure 1, we can have that the intention T_1 has only one trace, i.e. $\tau(T_1) = G_1; P_1; a_1; a_2, a_4$. Therefore, one possible execution trace of intentions $\{T_1, T_3\}$ can be $\sigma = \mathbf{G_1}; \mathbf{P_1}; G_3; P_3; \mathbf{a_1}; a_4; \mathbf{a_2}; \mathbf{a_4}; a_5$ by interleaving $\tau(T_1)$ and $\tau_1(T_3)$, where the subsequence in bold is $\tau(T_1)$ and non-bold is $\tau_1(T_3)$, i.e. one of execution traces of T_3 (see in Example 2).

However, randomly interleaving intentions may cause negative interactions to arise. For example, a previously achieved effect may be undone before an action that relies on it begins executing, thus preventing that action from being able to execute. Therefore, we define a *conflict-free* execution trace of a set of intentions to model the successful interleaving which achieves all intentions (i.e. intention resolution).

Definition 4. *Let $\mathcal{B}_j$ be the belief base before the execution of the j^{th} element of an execution trace (i.e. $\sigma[j]$). An execution trace σ is conflict-free if and only if the followings hold:*

(i) *if $\sigma[j] = P \in \Pi$, then $\mathcal{B}_j \models context(P)$ (i.e. the context of plan P must be met before selection);*
(ii) *if $\sigma[j] = a \in \Lambda$, then $\mathcal{B}_j \models \psi(a)$ (i.e. the pre-condition of action 'a' must be met before execution).*

where $j \in \{1, \ldots, |\sigma|\}$ and $|\sigma|$ is the length of σ.

Definition 4 says that a conflict-free execution trace is an execution trace which can be fully executed to completion without failure (i.e. avoiding all possible negative interactions between intentions) once it starts executing.

There may also exist potential positive interactions between intentions. For example, there may be a common sub-intention of two intentions that need only be executed once (i.e. merging such two identical sub-intentions into one) in order to progress both these two intentions. Therefore, we discuss what the commonality of intentions implies in the execution trace. We start with the definition of the *mergeable* execution trace.

Definition 5. *An execution trace σ of $\{T_1, \ldots, T_m\}$ is a mergeable execution trace if and only if the followings hold:*

(i) *$\exists j \in \{1, \ldots, |\sigma|\}$ such that $\sigma[j] = \ldots = \sigma[j+k]$ where $|\sigma|$ is the length of σ and $2 \leq k \leq |\sigma| - j$;*
(ii) *$\forall l \in \{1, \ldots, m\}, \nexists s, t \in \{j, \ldots, j+k\}$ where $s \neq t$ such that $\sigma[s] \subseteq \tau(T_l) \subseteq \sigma$ and $\sigma[t] \subseteq \tau(T_l) \subseteq \sigma$.*
(iii) *σ^m is a conflict-free execution trace where σ^m is the merged execution trace of σ by reducing each subsequence consisting of consecutive identical elements characterised by (i) and (ii) in σ to only one element.*

Criterion (i) and (ii) capture the synchronisation stage (i.e. different intentions are ready to execute the same actions at the same time). Criterion (iii) formalises the intention merging stage such that the subsequent merged execution trace σ^m is still a conflict-free execution trace (i.e. a correct execution).

Example 4. In Figure 1, one of the execution traces of T_1 and T_2 can be $\sigma_1 = G_1; P_1; G_2; P_2; \mathbf{a_1}; \mathbf{a_1}; a_2; a_3; \mathbf{a_4}; \mathbf{a_4}$. We can conclude that σ_1 is mergeable according to Definition 5 and its merged execution trace $\sigma_1^m = G_1; P_1; G_2; P_2; \mathbf{a_1}; a_2; a_3; \mathbf{a_4}$ is indeed a conflict-free execution trace (see in Section I).

Finally, we can define the *maximal-merged* execution trace.

Definition 6. *The merged execution trace σ^m of a mergeable execution trace σ of $\{T_1, \ldots, T_m\}$ is maximal-merged if there is no another mergeable execution trace σ' of $\{T_1, \ldots, T_m\}$ such that $|\sigma'^m| < |\sigma^m|$ where $|\sigma|$ stands for the length of σ.*

We close this section by noting that we are interested in finding one *maximal-merged* trace for a set of intentions if one exists. To this end, in the next section, we leverage the power of FPP to help us find such a *maximal-merged* trace.

B. Intention Interleaving Planning Preparation

Off-the-shelf FPP planners can be used to identify a maximal-merged trace if one exists. Before we present our FPP approach, we start with some technical preparation.

Indexing nodes: We introduce some additional notations, i.e. indexes, to the nodes of goal-plan trees. If a node n is a top-level goal of intention T, it is already uniquely identified by the notation $T(\bar{n})$. For nodes of action and sub-goals, i.e. $n \in \Lambda \cup \mathcal{G} \setminus \{T(\bar{n})\}$ of T, we use $n^{P,j,T}$ to denote the j^{th} member of $body(P)$ in T. This ensures that e.g. the same action in distinct plans is seen as different. Similarly, we use n^T to denote a plan node $n \in \Pi$ in an intention T. For ease of reference, we denote $J(idx)$ to retrieve the actual node of the index idx. From now on, we assume that whenever we talk about the nodes, we refer to the indexes of these nodes.

Terminal and initial node sets: We introduce the *terminal node set* for a goal node $G \in \mathcal{G}$. This set encodes the completion condition of the goal node, namely the last element of an execution trace of a goal. To be precise, the *terminal node set* of goal node G is $\nu(G) = \{\tau(G)^\infty \mid \tau(G) \in \omega(G)\}$ where $\tau(n)^\infty$ stands for the last element of execution trace $\tau(n)$. Therefore, we have $z_g = \{tn_1, \ldots, tn_m\}$ to be a *terminal node set* of a set of intentions $I = \{T_1, \ldots, T_m\}$, denoted $z_g \rhd_{tn} I$, where $tn_j \in \nu(T_j[\bar{n}])$ and $j \in \{1, \ldots, m\}$. Similarly, the top-level goal of each intention in $I = \{T_1, \ldots, T_m\}$, denoted as $z_0 = \{T_1(\bar{n}), \ldots, T_m(\bar{n})\}$, is called an *initial node set* of I. This set announces the starting point of each intention.

Example 5. In Figure 1, we have the indexes and terminal/initial nodes of execution traces of T_1 and T_2 as follows:

$$\tau(T_1): \begin{pmatrix} node & G_1 & P_1 & a_1 & a_2 & a_4 \\ index & T_1(\bar{n}) & P_1^{T_1} & a_1^{P_1,1,T_1} & a_2^{P_1,2,T_1} & a_4^{P_1,3,T_1} \end{pmatrix}$$

$$\begin{array}{cc} \downarrow & \downarrow \\ \textit{initial node} & \textit{terminal node} \\ \uparrow & \uparrow \end{array}$$

$$\tau(T_2): \begin{pmatrix} node & G_2 & P_2 & a_1 & a_3 & a_4 \\ index & T_2(\bar{n}) & P_2^{T_2} & a_1^{P_2,1,T_2} & a_3^{P_2,2,T_2} & a_4^{P_2,3,T_2} \end{pmatrix}$$

Progression links: We introduce *progression links* to encode the progression information of the execution traces.

Definition 7. *Let σ be an execution trace. For every two adjacent elements with indexes n, n' in σ (i.e. $n; n' \subseteq \sigma$), we say that an item in the form of $(n \to n')$ is a primitive progression link in σ, denoted as $(n \to n') \in \sigma$.*

The primitive progression links visualise the progression order of execution trace elements in the context of indexes.

Example 6. (Example 5 continued). We can have the progression links of execution trace $\tau_1(T_1)$ and $\tau_2(T_2)$ as follows:

$$\begin{array}{lll} \tau(T_1): & (T_1(\bar{n}) \rightarrow P_1^{T_1}), & (P_1^{T_1} \rightarrow a_1^{P_1,1,T_1}), \\ & (a_1^{P_1,1,T_1} \rightarrow a_2^{P_1,2,T_1}), & (a_2^{P_1,2,T_1} \rightarrow a_4^{P_1,3,T_1}); \\ \tau(T_2): & (T_2(\bar{n}) \rightarrow P_2^{T_2}), & (P_2^{T_2} \rightarrow a_1^{P_2,1,T_2}), \\ & (a_1^{P_2,1,T_2} \rightarrow a_3^{P_2,2,T_2}), & (a_3^{P_2,2,T_2} \rightarrow a_4^{P_2,3,T_2}); \end{array}$$

Computing overlaps: We now discuss how to compute all potential overlapping programs among a set of intentions.

Definition 8. *The overlap set of* $\{T_1, \ldots, T_m\}$ $(m \geq 2)$ *is a set of tuples of the form* $\langle(idx_b^1 \rightarrow idx_e^1), \ldots, (idx_b^k \rightarrow idx_e^k)\rangle$ $(2 \leq k \leq m)$ *such that the following holds:*

(1) $J(idx_e^1) = \ldots = J(idx_e^k)$ *where* $J(idx_e^i)$ *stands for the actual node of the ending index* idx_e^i*;*

(2) $\forall l \in \{1, \ldots, m\}, \nexists s, t \in \{1 \ldots, k\}$ *and* $s \neq t$ *such that* $(idx_b^s \rightarrow idx_e^s) \in \tau(T_l)$ *and* $(idx_b^t \rightarrow idx_e^t) \in \tau(T_l)$.

Definition 8 says the overlap set groups progression links from different intentions (2) that reach the same program (1).

Example 7. (Example 6 continued). The overlap set of intentions $\{T_1, T_2\}$ has two elements (a) and (b) as follows:

(a) $\langle(P_1^{T1} \rightarrow a_1^{P_1,1,T_1}), (P_2^{T_2} \rightarrow a_1^{P_2,1,T_2})\rangle$
where $J(a_1^{P_1,1,T_1}) = J(a_1^{P_2,1,T_2}) = a_1$.

(b) $\langle(a_2^{P_1,2,T_1} \rightarrow a_4^{P_1,3,T_1}), (a_3^{P_2,2,T_2} \rightarrow a_4^{P_2,3,T_2})\rangle$
where $J(a_4^{P_1,3,T_1}) = J(a_4^{P_2,3,T_2}) = a_4$.

We now define the *overlap progression link* as follows:

Definition 9. *Let an element of overlap set of* $\{T_1, \ldots, T_m\}$ $(2 \leq m)$ *be* $\langle(idx_b^1 \rightarrow idx_e^1), \ldots, (idx_b^k \rightarrow idx_e^k)\rangle$ $(2 \leq k \leq m)$*. We have a corresponding overlap progression link* $(\{idx_b^1, \ldots, idx_b^k\} \rightarrow \{idx_e^1, \ldots, idx_e^k\}) \in \{T_1, \ldots, T_m\}$.

Definition 9 says that each element of the overlap set amounts to an overlap progression link. The overlap progression link $(\{idx_b^1, \cdots, idx_b^k\} \rightarrow \{idx_e^1, \cdots, idx_e^k\})$ essentially merges all primitive progression links $(idx_b^i \rightarrow idx_e^i)$ and can progress from the (b)eginning indexes $idx_b^1, \cdots, idx_b^k$ all the way to its (e)nding indexes $idx_e^1, \cdots, idx_e^k$ $(2 \leq k \leq m)$.

Example 8. (Example 7 continued). The overlap progression links of $\{T_1, T_2\}$ are (a') and (b') shown in the following:

(a') $(\{P_1^{T1}, P_2^{T_2}\} \rightarrow \{a_1^{P_1,1,T_1}, a_1^{P_2,1,T_2}\})$;
(b') $(\{a_2^{P_1,2,T_1}, a_3^{P_2,2,T_2}\} \rightarrow \{a_4^{P_1,3,T_1}, a_4^{P_2,3,T_2}\}$;

where the overlap progression link (a') and (b') correspond to the overlap set element (a) and (b), respectively in Example 7.

Finally, we introduce the size of an overlap progression link as the number of the primitive progression links it merges.

Definition 10. *Let an overlap progression link* $\alpha^o = (\{idx_b^1, \ldots, idx_b^k\} \rightarrow \{idx_e^1, \ldots, idx_e^k\}) \in \{T_1, \ldots, T_m\}$. *The size of* α^o *is* $size(\alpha^o) = k - 1$ *(i.e. merging* $k - 1$ *extra primitive progression links). By default, the size of a primitive progression link* α^p *is* $size(\alpha^p) = 0$ *(i.e. no merging at all).*

We close this section by noting that what we have done so far is essentially to compute the overlap progression links of a given set of intentions. How to incorporate such overlap progression links in FPP to facilitate intention merging is the subject of the following section.

TABLE I
STRIPS PROGRESSION LINKS

link α^p	$pre(\alpha^p)$	$del(\alpha^p)$	$add(\alpha^p)$
$(idx_b \rightarrow P^T)$	$idx_b \cup \varphi$	$\{idx_b\}$	$\{P^T\}$
$(idx_b \rightarrow a^{P,j,T})$	$idx_b \cup \psi(a^{P,j,T})$	$\phi^- \cup \{idx\}$	$\phi^+ \cup \{a^{P,j,T}\}$
$(idx_b \rightarrow G^{P,j,T})$	idx_b	$\{idx\}$	$\{G^{P,j,T}\}$

C. Intention Interleaving Planning Formalism

In this section, we incorporate the overlap information in Section III-B in FPP to facilitate intention merging. We now represent the problem of intention interleaving as an FPP problem in the following definition.

Definition 11. *A FPP problem of interleaving intentions* $I = \{T_1, \ldots, T_m\}$ *is a tuple* $\Omega = \langle\Sigma, X, O, s_0, S_G\rangle$ *where:*

- Σ *is a finite set of (propositional) atoms;*
- $X = \bigcup_{j=1}^m T_j(N_\vee \cup N_\wedge)$ *is a set of node indexes of* I*;*
- $O = O^p \cup O^o$ *is a set of progression links.*
- $s_0 = \mathcal{B}_0 \cup z_o \in 2^\Sigma \cup 2^X$ *is the initial state;*
- $S_G = \{z_g \mid z_g \rhd_{tn} I\} \subseteq 2^X$ *is the goal state;*

where O^p *(reps.* O^o*) denotes the collection of primitive (resp. overlap) progression links of a set of intentions* I *while* z_0 *(reps.* z_g*) stands for the initial (reps. terminal) node set of* I.

Definition 11 says an initial state s_0 is a finite set of (propositional) atoms encoding an initial belief base $\mathcal{B}_0$ and the initial node set z_0 of intentions I, whereas the goal state S_G encodes the terminal node set z_g of intentions I. The set of progression links O captures the state transitions e.g. the indexes in the execution traces. The progression link $\alpha \in O$ is of the form $\langle pre(\alpha), del(\alpha), add(\alpha)\rangle$ where $pre(\alpha)$, $del(\alpha)$, and $add(\alpha)$ are called the pre-condition, delete-list, and add-list, respectively. The pre-condition, delete-list, and add-list are sets of atoms and node indexes in which the delete-list (resp. add-list) specifies which atoms and node indexes are removed from (resp. added to) the state of specification.

Table I gives the STRIPS representation of primitive progression links in O^p where idx_b is the beginning node index. For example, the progression link $(idx_b \rightarrow P^T)$ in Table I captures the transition from idx_b to a plan P^T. The precondition of applying progression link $(idx_b \rightarrow P^T)$ says that the context of P^T is being met and the agent currently is at the node idx_b (i.e. $idx \cup \varphi \in pre(\alpha^p)$).

Definition 12. *Let an overlap progression link in* O^o *be* $\alpha^o = (\{idx_b^1, \ldots, idx_b^k\} \rightarrow \{idx_e^1, \ldots, idx_e^k\})$ *in which* $\alpha_i^p = (idx_b^i \rightarrow idx_e^i) \in O^p$ $(1 \leq i \leq k)$*. We can have* $\langle pre(\alpha^o), del(\alpha^o), add(\alpha^o)\rangle$ *such that the following holds:*

- $pre(\alpha^o) = pre(\alpha_1^p) \cup \ldots \cup pre(\alpha_k^p)$
- $del(\alpha^o) = del(\alpha_1^p) \cup \ldots \cup del(\alpha_k^p)$
- $add(\alpha^o) = add(\alpha_1^p) \cup \ldots \cup add(\alpha_k^p)$

Definition 12 confirms that the overlap progression link α^o essentially merges related primitive progression links $\alpha_i^p = (idx_b^i \rightarrow idx_e^i)$ $(1 \leq i \leq k)$ into one. Therefore, e.g. the pre-condition of α^o is the conjunction of pre-condition of α_i^p.

Definition 13. *The result of applying a progression link $\alpha \in O$ to a state $s = \mathcal{B} \cup z$ is described by the transition function $f : 2^{\Sigma} \cup 2^{X} \times O \to 2^{\Sigma} \cup 2^{X}$ defined as follows:*

$$f(s, \alpha) = \begin{cases} (s \setminus del(\alpha)) \cup add(\alpha) & if\, s \models pre(\alpha) \\ undefined & otherwise \end{cases}$$

Hence we have the result of applying a sequence of progression links to a state specification s defined inductively:

$$Res(s, \langle\rangle) = s$$
$$Res(s, \langle \alpha_0; \ldots; \alpha_n \rangle) = Res(f(s, \alpha_0), \langle \alpha_1; \ldots; \alpha_n \rangle)$$

We now formally define the solution to our planning problem of intention interleaving in Definition 14 as follows:

Definition 14. *A sequence of progression links $\Delta = \langle \alpha_0; \alpha_1; \ldots; \alpha_n \rangle$ is a solution to a planning problem $\Omega = \langle \Sigma, X, O, s_0, S_G \rangle$, denoted as $\Delta = sol(\Omega)$, iff $Res(s_0, \Delta) \models S_G$. We also say that Δ is optimal if the sum of the size of the progression link $size(\alpha_i)$ is maximum where $i = 0, \ldots, n$.*

Definition 14 says the solution to a planning problem in Definition 11 is a sequence of progression links Δ which, when applied to the initial state specification using the *Res* function, reach a state that supports the terminal specification. The optimal solution is the solution which not only accomplishes all intentions but also merges the highest number of primitive progression links (see in Definition 10). We now formally establish the equivalence of the maximal-merged execution of the set of intentions and the optimal solution of the corresponding intention interleaving planning problem.

Theorem 1. *Let $I = \{T_1, \ldots, T_m\}$ be a set of intentions and $\Omega = \langle \Sigma, X, O, s_0, S_G \rangle$ be its corresponding intention interleaving planning problem. We have a maximal-merged trace σ^m of intentions $I = \{T_1, \ldots, T_m\}$ if and only if there exists an optimal solution Δ to Ω.*

Proof. (proof sketch) Suppose that there exists a maximal-merged trace σ^m of intentions $I = \{T_1, \ldots, T_m\}$. Hence, σ^m is also a conflict-free trace according to Definition 5 and Definition 6. Therefore, the terminal nodes of intentions I can be achieved. By the construction of the planning problem Ω, we can infer that the goal state S_G can be reached (i.e. there exists a solution). From Definition 10, we can see that by definition the number of merged primitive progression links is the size of a progression link, i.e. $size(\alpha_i)$. Hence, there also exists an optimal solution according to Definition 14. For the other side, let the optimal solution Δ be $\alpha_0; \ldots; \alpha_n$ such that $\alpha_i = (\{idx_b^{i_1}, \ldots, idx_b^{i_k}\}, \{idx_e^{i_1}, \ldots, idx_e^{i_k}\}$ where $i = 0, \ldots, n$ and $k = 1, \ldots, m$. We can construct an execution trace σ in the following steps: (1) sequentialise α_i into $\alpha_i' = idx_b^{i_1}; \ldots; idx_b^{i_k}; idx_e^{i_1}; \ldots; idx_e^{i_k}$; (2) remove any duplicate beginning indexes in $\sigma = \alpha_0'; \ldots; \alpha_n'$; (3) reduce subsequence $idx_e^{i_1}; \ldots; idx_e^{i_k}$ in σ into $idx_e^{i_1}$; (4) retrieve the actual node of indexes in σ (see in Section III-B). Finally, we can say σ is maximal-merged by contradiction. To be precise, if σ were not maximal-merged, then we would have Δ were not the optimal solution (which contradicts the assumption). □

Algorithm 1: Intention Interleaving Replanning

```
Input: Planning problem $\Omega = \langle \Sigma, X, O, s_0, S_G \rangle$
1  $\alpha_0; \ldots; \alpha_n \leftarrow sol(\Omega)$          /* FPP solution */
2  $i \leftarrow 0, \alpha \leftarrow \alpha_0, s \leftarrow s_0$        /* initialisation */
3  while $s \notin \Upsilon$ do
4      if $f(s, \alpha) = undefined$ then
5          $idx_b \leftarrow$ BEGINNING-INDEX($\alpha$)
6          $G \leftarrow$ BACKTRACK($idx_b$)     /* backtrack */
7          $s_0 \leftarrow \mathcal{B} \cup z \setminus \{idx_b\} \cup \{G\}$  /* modify state */
8          $sol'(\Omega) \leftarrow$ FPP($\langle \Sigma, X, O, s_0, S_G \rangle$)   /* replan */
9          $\alpha_0; \ldots; \alpha_n \leftarrow sol'(\Omega)$
10         $\alpha \leftarrow \alpha_0, i \leftarrow 0$     /* re-initialisation */
11     EXECUTE $\alpha$
12     $s \leftarrow f(s, \alpha)$
13     $i \leftarrow i + 1$
14     $\alpha \leftarrow \alpha_{i+1}$
```

So far what we have discussed is known as *offline planning*, i.e. a complete plan is generated and then executed in full. However, the environment is dynamic and pervaded by uncertainty. It may imply that the change of the environment (e.g. exogenous events can occur) would block the execution of the complete plan generated from FPP. For example, in a smart home environment, there is an intelligent domestic robot which finished chores in the lounge and needs to move to the hall doing chores. The robot chooses a plan which needs to pass through the hallway door to reach the hall. However, the pet dog accidentally slammed the door shut before the robot reaches the hallway door. As a consequence, this plan would be undesirably blocked. In BDI agents, when an execution failure occurs, the agent will backtrack to the related motivating goal and tries another applicable plan to achieve such a goal. Therefore, different from the classical replanning which replanning takes place right from the current state where the execution failure happens, the BDI agent propagates the failure to its higher-level goal first. Therefore, for intention interleaving replanning, we need the prefix steps which backtracks to the higher-level goal and modifies the initial node. The steps of replanning are given in Algorithm 1 in which, e.g. **line 5-7** instruct the procedures for failure backtracking and initial node state modification.

Example 9. In Figure 2, if the agent is currently at the node a_4 and is no longer able to progress to a_5 (e.g. the environment changed unexpectedly). Then the agent should go back to its motivating goal G_3 and start replanning from there. Correspondingly, for its planning problem Ω the initial state $s_0 = \mathcal{B}_0 \cup \{a_4\}$ updates to $s_0 = \mathcal{B}_0 \cup \{G_3\}$ for replanning.

IV. IMPLEMENTATION

In this section, we provide the practical implementation of our FPP approach in PDDL representation [15] which consists of two parts: (i) an *operator file* containing progression links; (ii) a *fact file* encoding the initial and goal state description.

Operator File: We start with encoding the primitive progression link in PDDL in an *operator file*, namely $(idx_b \to P^T)$, $(idx_b \to a^{P,j,T})$, and $(idx_b \to G^{P,j,T})$ according to Table I in Section III-C. Note PDDL definitions require predicates. For legibility of presentation, however, we simply use the relevant mathematical symbols as syntactic sugar. Therefore, we can have the following list of actions in PDDL.

```
(:action ($idx_b \to P^T$)
:precondition (and $idx_b$ $context(P)$ )
:effect (and (not $idx_b$) $P^T$))
(:action ($idx_b \to a^{P,j,T}$)
:precondition (and $idx_b$ $\psi(a^{P,j,T})$ )
:effect (and (not $\phi^-$) $\phi^+$ (not $idx_b$) $a^{P,j,T}$))
(:action ($idx_b \to G^{P,j,T}$)
:precondition $idx_b$
:effect (and (not $idx_b$) $G^{P,j,T}$))
```

We now encode the overlap progression link in PDDL in an *operator file*. Let an overlap progression link be $\alpha^o = (\{idx_b^1, \dots, idx_b^k\} \rightarrow \{idx_e^1, \dots, idx_e^k\})$ where the primitive progression link $\alpha_i^p = (idx_b^i \rightarrow idx_e^i)$ $(1 \leq i \leq k)$. Therefore, we have the following:

```
(:action ($\{idx_b^1,\dots,idx_b^k\} \to \{idx_e^1,\dots,idx_e^k\}$))
:precondition (and $pre(\alpha_1^p)\dots,pre(\alpha_k^p)$ )
:effect (and $add(\alpha_1^p)\dots add(\alpha_k^p)$
        (not $del(\alpha_1^p)$ ) ... (not $del(\alpha_k^p)$ )
        (increase (efficiency-utility) $size(\alpha^o)$))))
```

where the syntax `(increase (efficiency-utility)` $size(\alpha^o)$`)` specifies the reward of the progression link to be its size.

Fact File: The *fact file* includes the initial state description and the goal state description. We start by declaring the objects present in the planning problem instance.

The objects consist of all indexes of elements of all execution traces besides other ground belief atoms.

`(:objects` $\forall x \in X$, $\forall$ BELIEF_ATOMS $\in \Sigma$)

The initial condition consists of initial belief base $\mathcal{B}_0$ and the top-level goals of intentions.

`(:init` $\mathcal{B}_0, \forall T \in I, T(\bar{n})$)

The goal for the planning problem is to reach any terminal node of each intention in $\{T_1, \dots, T_m\}$ $(1 \leq j \leq m)$.

`(:goal (and (or` tn_1^1 ... $tn_{k_1}^1$`) ... (or` $tn_1^m \dots tn_{k_m}^m$`))`

where $\{tn_1^j, \dots, tn_{k_j}^j\}$ is the terminal node set of the intention T_j and the syntax 'or' means that reaching any of the terminal nodes $\{tn_1^j, \dots, tn_{k_j}^j\}$ would achieve the intention T_j (i.e. `:disjunctive-preconditions` requirement in PDDL).

Finally, we show how to obtain a maximal-merged execution trace through the optimisation in PDDL. To do so, we add a fluent function `(:function(efficiency-utility))` to keep track of the efficiency utility with an initial efficiency utility specification `(=(efficiency-utility)0)`. Then we add a `:metric` section to the *fact file* with `(:metric maximise(efficiency-utility))` to specify that maximising the sum of efficiency-utility is the objective.

V. EVALUATION

In this section, we present some effectiveness results to show the feasibility of our approach. Consider a manufacturing scenario of using machining operations to make holes in a metal block. There are several different kinds of hold-creation operations (e.g. twisting-drilling, spade-drilling) available, as well as several different kinds of hole-improvement operations (e.g. reaming, boring). Each time the robotic arm switches to a different kind of operation or to a hole of different diameter, it must mount a different cutting tool on its arm. If the same cutting operation is to be performed on two (or more) holes of the same diameter, then these same operations can be merged by omitting the repetitive task of changing the cutting tools.

We generate such manufacturing scenarios in which the detailed design were varied by: (i) the number of blocks (n from 2 to 8); (ii) operations per blocks (m from 2 to 4), and (iii) the maximal number of overlap operations among all metal blocks (k from 1 to 4), resulting in 63 test cases in total. We assume that each operation has three actions, e.g. twisting-drilling task needs (i) action of taking on a twisting-drill, (ii) actual twisting-drilling action, (iii) action of taking off this twisting-drilling. For simplicity, the shared operations among a set of blocks are in the same order in each metal block. For example, if block 1 and block 2 share both twisting-drilling and reaming operation, we would expect the twisting-drilling operation before reaming operation in both blocks in practice. The dataset and instructions for reproduction are available online[2]. These cases were then solved via our FPP approach where a planner called **Metric-FF**[3] is employed.

TABLE II
EFFECTIVENESS ANALYSIS OF APPROACH

	2.1	2.2	3.1	3.2	3.3	4.1	4.2	4.3	4.4
2	17%	33%	11%	22%	33%	8%	17%	25%	33%
3	22%	44%	15%	30%	44%	11%	22%	33%	44%
4	25%	50%	17%	33%	50%	13%	25%	38%	50%
5	27%	53%	18%	36%	53%	13%	27%	40%	53%
6	28%	56%	19%	37%	56%	14%	28%	42%	56%
7	29%	57%	19%	38%	57%	14%	29%	43%	57%
8	29%	58%	19%	39%	58%	15%	29%	44%	58%

Table II shows the effectiveness results of our approach where rows are the number of metal blocks n from 2 to 8 and columns $m.k$ reads as there are m operations among which there are k overlapping operations. Compared to the default approach without capitalising on overlapping operations, our FPP approach not only successfully achieves all the intentions, but also reduces the amount of repetitive task of changing the cutting tools. The value in the table is the improved efficiency defined as the reduced number of actions divided by the total number of actions if without merging identical operations. For example, if there are 4 metal blocks, 3 operations for each metal, and 2 overlapping operations over these 3 operations, our approach can improve the efficiency by 33%, i.e. reducing 12 repetitive changing tool actions out of 36 actions in total if without intention merging. We also observe the efficiency to increase with the number of blocks (see in each column). When all operations for all blocks are the same, the efficiency is the same regardless of the number of blocks (see the same efficiency values in column 2.2, 3.3, and 4.4).

[2]https://github.com/Mengwei-Xu/manufacturing-evaluation
[3]https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html

VI. Related Work

Apart from the work of [13], [14], we are not aware of any other existing work on intention interleaving with the focus on discovering and exploiting identical sub-intentions in BDI agents. However, the concept of planning merging is not new in the planning community. In fact, a large body of research in planning focused on how to coordinate after plans have been constructed separately, particularly in the multi-agent setting [17]. For example, the classic STRIPS line of planning merging are studied by [18], [19] to merge alternative plans to reach the same goal. The work of [20] also presents a method of searching for and exploiting overlapping effects between different hierarchical planning agents in a multi-agent system. Some even studied a theory of rational choice where an agent evaluates its options in the context of existing plans [21]. However, the work above, like many others, requires that the plans are determined before execution. These do not apply to the BDI agents that we work with, which assume highly dynamic environments where plans depend on the current environment conditions (i.e. acting as it goes) and failure recovery is also supported.

There are also some other works which exploit the positive intention interaction for intention resolution in BDI agents. For example, the work of [22] studied the robust execution of BDI agent programs by exploiting synergies between intentions. Instead of backtracking to recover from an execution failure, they proposed an approach to appropriate scheduling the remaining progressable intentions to execute an already intended action which re-establishes a missing pre-condition.

Another noticeable work [23] combines work on both intention and planning. However, their purpose is to split the original actions into several stages of intention (i.e. refinement of action) to solve unary planning problems. In fact, a large number of works integrate automated planning techniques into BDI agents to generate plans at runtime, as surveyed by Meneguzzi and De Silva [24]. However, our work is one of the few which formally integrates planning techniques into BDI agents to managing intention interleaving.

VII. Conclusions

In this paper we showed how concurrent intention executions in BDI agents can be managed by first-principles planning. Our planning-centric approach to BDI agents not only guarantees the accomplishment of intentions, but also reduce the cost of execution via merging identical sub-intentions, thus improving the overall efficiency of the BDI agents. Our manufacturing experimental results indicate the effectiveness of our approach when compared to BDI agents that do not harness the advantages of commonality between intentions. A naive implementation of the algorithm to compute the overlap set of intentions has factorial time complexity. For future work, we believe the algorithm can be improved to incorporate hashing ideas, such as in [25], to make the algorithm viable for large scale problems. In addition, we plan to further test the costs and benefits empirically in a wider range of applications.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 research and innovation programme through the DEVELOP project, under grant agreement No. 688127.

References

[1] M. Bratman, "Intention, plans, and practical reason," 1987.

[2] A. Rao, "AgentSpeak (L): BDI agents speak out in a logical computable language," in *Proc. of MAAMAW '96*, 1996, pp. 42–55.

[3] K. V. Hindriks, F. S. De Boer, W. Van der Hoek, and J.-J. C. Meyer, "Agent programming in 3APL," *Autonomous Agents and Multi-Agent Systems*, pp. 357–401, 1999.

[4] S. Sardina, L. Silva, and L. Padgham, "Hierarchical planning in BDI agent programming languages: A formal approach," in *Proc. of AAMAS'06*, 2006, pp. 1001–1008.

[5] M. Dastani, "2APL: a practical agent programming language," *Autonomous Agents and Multi-Agent Systems*, pp. 214–248, 2008.

[6] R. Bordini and J. Hübner, "Programming multi-agent systems in AgentSpeak using Jason," in *Proc. of CLIMA'06*, 2006, pp. 143–164.

[7] A. Herzig, E. Lorini, L. Perrussel, and Z. Xiao, "BDI logics for BDI architectures: Old problems, new perspectives," *KI-Künstliche Intelligenz*, pp. 73–83, 2017.

[8] S. Shapiro, S. Sardina, J. Thangarajah, L. Cavedon, and L. Padgham, "Revising conflicting intention sets in BDI agents," in *Proc. of AAMAS'12*, 2012, pp. 1081–1088.

[9] B. Logan, J. Thangarajah, and N. Yorke-Smith, "Progressing intention progression: a call for a goal-plan tree contest," in *Proc. of AAMAS'17*, 2017, pp. 768–772.

[10] J. Thangarajah, L. Padgham, and M. Winikoff, "Detecting and avoiding interference between goals in intelligent agents," in *Proc. of AAMAS'03*. Morgan Kaufmann Publishers, 2003.

[11] Y. Yao and B. Logan, "Action-level intention selection for BDI agents," in *Proc. of AAMAS'16*, 2016, pp. 1227–1236.

[12] W. Shen, L. Wang, and Q. Hao, "Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 4, pp. 563–577, 2006.

[13] J. Thangarajah, L. Padgham, and M. Winikoff, "Detecting and exploiting positive goal interaction in intelligent agents," in *Proc. of AAMAS'03*. ACM, 2003, pp. 401–408.

[14] J. Thangarajah and L. Padgham, "Computationally effective reasoning about goal interactions," *Journal of Automated Reasoning*, vol. 47, no. 1, pp. 17–56, 2011.

[15] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. S. Weld, and D. Wilkins, "PDDL-the planning domain definition language," 1998.

[16] T. Keller and P. Eyerich, "Prost: Probabilistic planning based on uct." in *Proc. of ICAPS'12*, 2012.

[17] M. M. D. Weerdt, "Plan merging in multi-agent systems," *PhD Thesis, Delft University of Technology*, 2003.

[18] Q. Yang, D. S. Nau, and J. Hendler, "Merging separately generated plans with restricted interactions," *Computational Intelligence*, vol. 8, no. 4, pp. 648–676, 1992.

[19] D. E. Foulser, M. Li, and Q. Yang, "Theory and algorithms for plan merging," *Artificial Intelligence*, vol. 57, no. 2-3, pp. 143–181, 1992.

[20] J. S. Cox and E. H. Durfee, "Discovering and exploiting synergy between hierarchical planning agents," in *Proc. of AAMAS'03*. ACM, 2003, pp. 281–288.

[21] J. F. Horty and M. E. Pollack, "Evaluating new options in the context of existing plans," *Artificial Intelligence*, pp. 199–220, 2001.

[22] Y. Yao, B. Logan, and J. Thangarajah, "Robust execution of BDI agent programs by exploiting synergies between intentions," in *Proc. of AAAI'16*, 2016, pp. 2558–2565.

[23] J. Wolfe and S. Russell, "Bounded intention planning," in *Proc. of IJCAI'11*, 2011, pp. 2039 – 2045.

[24] F. Meneguzzi and L. Silva, "Planning in BDI agents: A survey of the integration of planning algorithms and agent reasoning," *The Knowledge Engineering Review*, vol. 30, pp. 1–44, 2015.

[25] O. Ertl, "SuperMinHash-a New Minwise Hashing Algorithm for Jaccard Similarity Estimation," *arXiv preprint arXiv:1706.05698*, 2017.