Sustainable and reliable robotics for part handling in manufacturing

Project no.:	**610917**
Project full title:	**Sustainable and reliable robotics for part handling in manufacturing**
Project Acronym:	**STAMINA**
Deliverable no.:	**D4.3.1**
Title of the document:	**Architecture Description for the Mission Planning Subsystem**

Contractual Date of Delivery to the CEC:	**31.12.2015**
Actual Date of Delivery to the CEC:	**21.12.2015**
Organisation name of lead contractor for this document:	**University of Edinburgh (UEDIN)**
Author(s):	**Matthew Crosby, Ron Petrick**
Work package contributing to the document:	**WP4**
Nature:	**R**
Version	**1.0**
Total number of pages:	**21**
Start date of project:	**01.10.2013**
Duration	**42 months – 31.03.2017**

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610917

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable reports on the architecture for the Mission Planner subsystem that is situated between the EIS and the individual robots in the STAMINA system. The Mission Planner is responsible for producing mission assignments for the individual robots in the fleet using information on kitting orders, robot skills, and the state of the operating environment provided by the Logistic Planner. Generated missions are returned to the Logistic Planner which ultimately relays them to the individual robots for execution.

Keyword List: Mission Planner, high-level architecture, automated multiagent planning

Document History

Version	Date	Author (Unit)	Description
1.0	14.12.2015	M.Crosby (UEDIN), R.Petrick (UEDIN)	Initial version of document.

Contents

1 Executive summary

1.1 Overview and background

A central objective of WP4 (Mission tasks and vertical enterprise integration) is to support the integration of a collection of robots with different skills into a heterogeneous robot fleet, by defining and implementing the architecture for a *mission planning subsystem*. The Mission Planner itself is responsible for generating mission assignments which map particular kitting orders to individual robots in the fleet. The mission planning architecture must therefore provide the necessary interfaces for communicating with the Logistic Planner, which supplies information about the status of the kitting orders, robots, and the environment, and for returning completed mission plans to the Logistic Planner for delivery to the robot fleet.

High-level mission planning capabilities in STAMINA are supplied by the ADP planner (Agent Decomposition Planner), which UEDIN is developing for use with multiple robots in a real-world factory environment. ADP is a multiagent automated planner which is designed to deduce assignments of goals and actions to particular agents (robots) from standard PDDL-based encodings of the planning problem, and to exploit these decompositions for efficient planning.

Like most automated planners, ADP operates best in discrete, symbolic state spaces described using logic-like representations such as PDDL (Planning Domain Definition Language), the de facto standard representation of the automated planning community. As a result, work that addresses the problem of integrating automated planning on real-world robot systems often centres around the problem of representation, and how to abstract the capabilities of the robot(s) and their working environment so that they can be put into a suitable form for use by a goal-directed planner. Integration also requires the ability to communicate information between system components. Thus, the design of the mission planning subsystem must take into consideration external concerns, to ensure proper interoperability with modules that aren't traditionally considered part of the core planning process.

In the first year of STAMINA (Month 1–Month 12), the work of WP4 considered the design of the high-level architecture of which the mission planning subsystem was a main component. An initial version of this architecture was presented in deliverable D4.1.1 (Specification of SOA for robot fleets). In the second year of project (Month 13–Month 24), this interface was further refined and an updated version of the high-level STAMINA architecture was presented in deliverable D4.2.3 (Conversion logic and high level operations scheduling). Likewise, an initial prototype of the Mission Planner was developed for offline testing during Year 1, and was integrated as part of the STAMINA system for initial online testing during Year 2.

This deliverable therefore reports on the current state of the mission planning architecture, with details of the current implementation of the Mission Planner itself, as of Month 27 of the project.

1.2 Main contributions and developments

The main contributions of this deliverable are reported in three attached papers. The first paper, *Mission Planner Interface* (Crosby & Petrick, 2015) is a technical report outlining the current state of the services that are implemented to support the behaviour of the Mission Planner. In particular, this paper does not present any technical details of the planning process, but simply describes the subsystem architecture relating to the mission planning task. The second paper, *ADP an Agent Decomposition Planner* (Crosby, 2015) provides technical details of the implementation of the ADP planner itself. Finally, the third paper, *Mission Planning for a Robot Factory Fleet* (Crosby & Petrick, 2015) describes the mission planning process in the context of the task of assigning kitting orders to a fleet of robots in a factory environment. As part of this process, initial plans are generated for individual robots as sequences of robot skills (WP3).

Overall, this deliverable reports on a number of significant developments:

- Specification of the Mission Planner system as it fits into the overall STAMINA architecture. This includes defining the services used for interaction between the Mission Planner and Logistic Planner.
- Implementation of the Mission Planner for use during the second test sprint. The Mission Planner component successfully interacted directly with the Logistic Planner (and indirectly with the skills framework) and returned an appropriate plan for all problems tested during the second test sprint.
- Integration of the existing in-house multiagent planning algorithm into the Mission Planner. The algorithm was re-implemented and entered into an international multiagent planning competition where it achieved first place in terms of coverage. Details of the planning competition itself can be found at `http://agents.fel.cvut.cz/codmap/`.
- Development of a new multiagent planning algorithm that can solve STAMINA-like multirobot problems in a way that produces more balanced plans. In particular, it was shown that plans can be returned for large STAMINA problem instances with interleaved action structure which is not possible with other multiagent planning techniques.

1.3 Ongoing and future work

A number of tasks remain open at the time of this report and constitute ongoing and future work:

- The current version of the mission planner is designed to return not only mission assignments (goals) but also skill sequences to the individual robots due to the fact that the robots did not include a task planning component during the second test sprint. Now that the Task Planner has been implemented, we are considering updates to the Mission Planner so that it will only return mission assignments, with skill sequence generation passed to the individual robot Task Planners. This will allow the Mission Planner to scale to larger numbers of robots and larger factory sizes. This will also remove redundancy since repeated calculations previously performed by the Mission Planner will now be distributed amonst the robot-level Task Planners.
- Once the above task is implemented, this will allow for further collaboration between Task 4.4 and Task 2.5 for multi-robot coordination. Simulated testing will be able to include all components of the STAMINA system. In particular, the interaction between different design choices in the Task Planner and Mission Planner components can be tested.
- The multiagent planning algorithms will continue to be tested and improved throughout the project.

For the latest public release of the ADP software, see `http://homepages.inf.ed.ac.uk/mcrosby1/`.

2 Attached papers

The following papers are attached to this report:

Mission Planner Interface
M. Crosby and R. Petrick
Technical Report, University of Edinburgh, 2015

Abstract: This document outlines the architecture for the Mission Planner subsystem in the STAMINA project. The Mission Planner is responsible for generating mission assignments for individual robots in the fleet, based on kitting order, robot, and environmental information provided by the Logistic Planner. Successfully-generated mission assignments are returned to the Logistic Planner for delivery to individual robots. This document focuses solely on the description of services that are implemented to support this behaviour in the Mission Planner. This document is a working document and subject to change as the project evolves and improvements are made.

ADP an Agent Decomposition Planner CoDMAP 2015
M. Crosby
Proceedings of the ICAPS Competition of Distributed and Multi-Agent Planner (CoDMAP), 2015

Abstract: ADP (an Agent Decomposition-based Planner) is designed to deduce agent decompositions from standard PDDL-encoded planning problems and then to exploit such found decompositions to generate a heuristic used for efficient planning. The decomposition process partitions the problem into an environment and a number of agents which act on and influence the environment, but can not (directly) effect each other. The heuristic calculation is an adaptation of the FF relaxation heuristic to incorporate multiagent information. Relaxed planning graphs are only ever generated for single-agent sub- problems. However, when cooperation is necessary, an agent's starting state may include facts added by others.

Mission Planning for a Robot Factory Fleet
M. Crosby and R. Petrick
Proceedings of the IROS Workshop on Task Planning for Intelligent Robots in Service and Manufacturing, 2015

Abstract: Planning is becoming increasingly prevalent as a tool for high-level reasoning in real-world robotics systems. This paper discusses the implementation of a high-level 'mission planner' that utilises AI planning techniques to find initial plans for a fleet of robots acting in a manufacturing factory. The paper introduces the system architecture, and then focuses on the ROS-based mission planning component, which requires the translation of low-level robot 'skills' and a world model to a high-level planning domain. This paper also introduces a new algorithm for decomposition-based planning that can find 'balanced' plans in large multi-robot domains where current state-of-the-art techniques fail.

MISSION PLANNER INTERFACE

Matthew Crosby and Ron Petrick
University of Edinburgh
m.crosby@ed.ac.uk, rpetrick@inf.ed.ac.uk

2015-12-14

Abstract

This document outlines the architecture for the Mission Planner subsystem in the STAMINA project. The Mission Planner is responsible for generating mission assignments for individual robots in the fleet, based on kitting order, robot, and environmental information provided by the Logistic Planner. Successfully-generated mission assignments are returned to the Logistic Planner for delivery to individual robots. This document focuses solely on the description of services that are implemented to support this behaviour in the Mission Planner. This document is a working document which is subject to change as the project evolves and improvements are made.

Revision history

2015-12-14 : Revised version of the mission planning architecture to reflect changes in Year 2 of STAMINA.

2014-09-15 : Update to include current state of integration between the Mission Planner and Logistics Planner.

2014-09-09 : Update to include current progress in relation to the mission planning architecture.

2014-06-09 : A preliminary overview of the mission planning architecture in STAMINA.

1 Introduction

This document outlines the architecture for the Mission Planning subsystem in the STAMINA project. The Mission Planner is responsible for generating mission assignments for individual robots in the fleet, based on kitting order, robot, and environmental information provided by the Logistic Planner. This is a working document and subject to change as the project evolves and improvements are made.

As of December 2015, the services between the Mission Planner and Logistic Planner are well-defined and fully implemented. It is likely that the output of the Mission Planner will be further refined now that the Task Planner components of the robots (WP3) is implemented (see discussion below in Section 6 for more details).

2 Terminology

- **Kitting Order:** List of parts and relevant information for a single kit. (This is referred to as a FAQ in the MES, but the term Kitting Order (abbreviated KO) will be used when talking about STAMINA components.
- **Robot:** The macro entity that will fulfil kitting orders by navigating, picking, placing etc. In STAMINA a robot consists of a mobile base on which an arm with a three fingered gripper is mounted. For the purposes of SkiROS, the mobile base and robotic arm may be referred to as different robots, but from the perspective of multi-robot planning performed by the mission planner, the multiple robots are the different macro entities.
- **Mission:** An instruction for one robot to fulfil one or two kits.
- **Supermarket:** The area of the factory in which the parts are contained and the robot can navigate.
- **MES:** The Manufacturing Execution System. In the case of STAMINA this is the MES already existing in PSA factories.

3 Mission Planner Role

The Mission Planner's goal is summarised as follows:

> The Mission Planner will take as input information about the kitting orders, robots, and supermarket layout and return an assignment of kitting orders to robots (missions) that (if executed correctly) will result in kits being delivered on time and in the correct order.

Any problems that arise during plan execution of a particular robot will be handled by the on-board robot Task Planner and Skill Managers.

To fulfil its role, the Mission Planner requires up-to-date information about the kitting orders that require processing, the robot skills information, and the layout of the supermarket.

The complexity of the Mission Planner's task grows exponentially with the number of robots deployed in the factory. As the test sprints will use only one robot, the Mission Planner will not be tested to its full potential during this. As such, the Mission Planner will continue to be tested offline during the project.

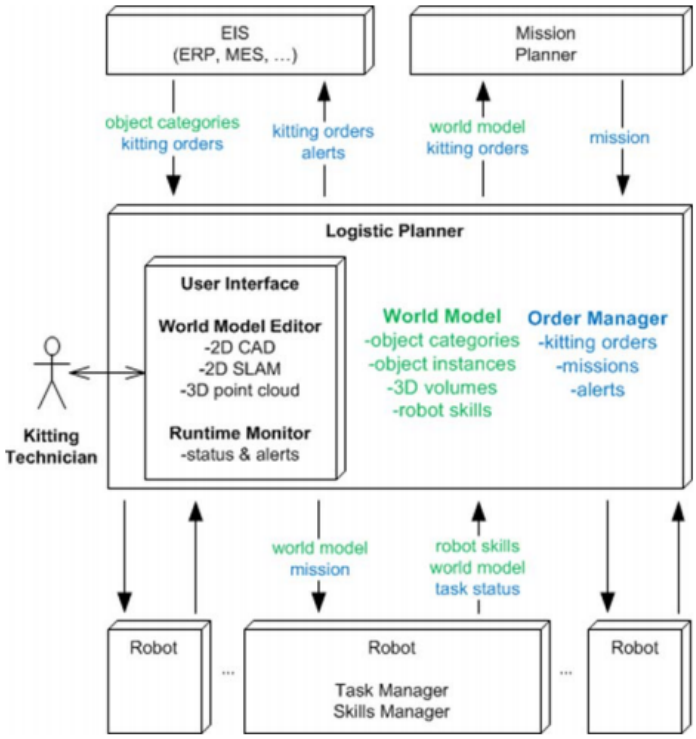

Figure 1: Diagram (from deliverable 4.2.3) showing the interaction between the Logistic Planner and the Mission Planner and how this fits into the high-level STAMINA system.

4 STAMINA Architecture

As shown in Figure 1 it was decided that the Mission Planner would not be directly connected to the robot fleet and would instead communicate via the Logistic Planner. The reason for this decision is that the Logistic Planner forms the central repository for information about the state of the world and is the point of access for the kitting technician. From the kitting technician's perspective, the Mission Planner functions in the background, silently providing mission assignments to the robot fleet. It is only interacted with via information sent to the Logistic Planner. This ensures that there is only one point of entry to the system, improving ease of use and allowing the interface to be more streamlined.

An initial version of the high-level STAMINA architecture, of which the Mission Planner is a component, was described as part of deliverable D4.1.1 (Specification of SOA for robot fleets) and later refined in deliverable D4.2.3 (Conversion logic and high level operations scheduling). Below, we focus primarily on the Mission Planner itself and describe the main services that are implemented to support this component.

5 Mission Planner Services

The interactions between the Mission Planner and Logistic Planner are provided by two services. The first service is used by the Logistic Planner to send all relevant information to the Mission Planner and trigger the creation of the next mission. The second is used by the Mission Planner to notify the Logistic Planner of the next missions to be sent to the robot fleet.

5.1 PlanKittingOrders

The service MissionPlanner_PlanKittingOrders is provided by the Mission Planner and consumes the kitting orders, robot skills and world model supplied by the Logistic Planner. It has the following syntax:

```
KittingOrder[] kittingOrders
RobotSkills[]  robotSkillsList
Node[]         nodeList
---
ActionAcceptedRefused actionAcceptedRefused
```

kittingOrders is an ordered list of kitting orders. These are all the kitting orders currently provided to the Logistic Planner from the MES. It is not expected that the Mission Planner returns mission assignments for all the missions in the list, only the missions that can be achieved one cycle by the robot fleet.

The robotSkillsList contains the information of each robots skill capabilities. This is not expected to change much during the course of a day, but may have crucial changes if new robots are introduced or old robots are retired from the fleet. The skills a robot is capable of will determine which missions are assigned to it.

The nodeList is the Logistic Planners internal representation of the world model. This information is used to provide initial skill sequences to the robots and to determine which robots are capable of performing which mission assignments most efficiently.

5.2 AssignMission

The service AssignMission is provided by the Logistic Planner and it is used by the Mission Planner to send the missions to the Logistic Planner. It has the following syntax:

```
string         missionId
string         robotId
KittingOrder[] kittingOrderList
Skill[]        plan
---
ActionAcceptedRefused actionAcceptedRefused
```

The robotId specifies which robot is to be sent the mission and the mission is given a unique ID for tracking purposes. The kittingOrderList lists the kitting orders that the mission will fulfil, this could be one or two, as the robot can carry up to two kitting boxes at a time. A plan is an ordered list of parameterised skills such that it can be executed by the Skill Manager of the robot. The returned plan can be used by the robot to begin execution straight away. This is especially useful in environments where there are only a few robots.

6 Upcoming Work

The final part of the interface, the assignMission service, is the most likely to be updated over the course of the project. In previous test-sprints the robots had no on-board task planner (the Task Planning subtask was not due to start until later in the project). Therefore, the Mission Planner sent plans to the robots so that they had a sequence of skills to execute. The Mission Planner effectively did the job of the task planner.

Now the Task Planner has been integrated into SkiROS, this frees the Mission Planner to perform more high-level mission planning. Instead of returning a plan (sequence of skills) to the robot, it makes more sense to return mission assignments (such as which kitting orders each robot will fulfil).

This has three main benefits. The first benefit is that it allows the robots complete autonomy to replan and find the best way from their perspective to achieve the assigned tasks. This also removes any duplicated effort between the Mission Planner and Task Planners which should improve system performance. The second benefit is that this allows the Mission Planner to scale to larger systems with more parts, racks and boxes in the supermarket as well as more robots. The final benefit is that this setup allows for improved collaboration between UEDIN and INESC and tasks 4.4 and 2.5 as the task 2.5 work can be more easily incorporated into this setup.

ADP an Agent Decomposition Planner CoDMAP 2015

Matthew Crosby
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
m.crosby@ed.ac.uk

Abstract

ADP (an Agent Decomposition-based Planner) is designed to deduce agent decompositions from standard PDDL-encoded planning problems and then to exploit such found decompositions to generate a heuristic used for efficient planning. The decomposition process partitions the problem into an environment and a number of agents which act on and influence the environment, but can not (directly) effect each other. The heuristic calculation is an adaptation of the FF relaxation heuristic to incorporate multiagent information. Relaxed planning graphs are only ever generated for single-agent subproblems. However, when cooperation is necessary, an agent's starting state may include facts added by others.

Introduction

ADP is a complete, satisficing (non-optimal) centralised planning algorithm that attempts to compute and utilise agent decompositions for the sole purpose of improving planning time. As such, it does not take into account common multiagent concerns such as privacy, trust or strategic considerations. It has been shown (Crosby, Rovatsos, and Petrick 2013; Crosby 2014) that useful decompositions can be found and successfully utilised in around forty percent of IPC domains (IPC 2011), a collection of domains which are not explicitly designed to be multiagent, yet contain some obviously multiagent settings.

The first section of this paper provides a brief high-level overview of the ADP algorithm, while the following section provides more detail including explicit discussion of the decomposition process and heuristic calculation. The third section presents a summary of the agent decomposition results for the domains used in CoDMAP 2015 after which some limitations and future plans for ADP are discussed. Technical details of the planner and other information relevant to CoDMAP 2015 can be found in the final section.

ADP Overview

ADP is split into two components, a decomposition phase and a heuristic calculation. The decomposition phase processes the planning problem and attempts to find a useful multiagent decomposition. As depicted in Figure 1, a decomposed domain is made up of a number of agents, each with an internal state, and an environment that the agents are acting upon. Actions that can influence or depend upon the internal state of an agent are assigned to only that agent. Actions that only involve the environment are assigned to all agents. The decomposition algorithm is guaranteed to return a decomposition in which no actions that influence (or rely upon) multiple agents' internal states exist. This means that agents can only influence themselves or the environment (thought the actions available to an agent in a given state may depend on how others have influenced the environment).

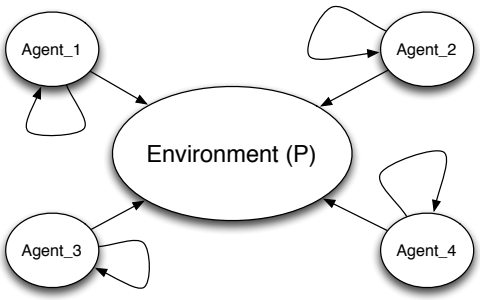

Figure 1: A depiction of an agent decomposition of the variables in a domain. Agents can only change the state of themselves and/or the environment.

When a decomposition is found, ADP calculates a heuristic value (used to guide a greedy best first search) that attempts to exploit the multiagent structure to find plans faster than the alternative single-agent approach. If no suitable decomposition can be found, then ADP defaults to the the single-agent FF heuristic (Hoffmann and Nebel 2001).

The main idea behind the multiagent heuristic calculation is to only ever generate planning graphs for a single agent subproblem at a time. In 'coordination points' each agent computes its heuristic value for each goal proposition by generating its relaxed planning graph from the current state. Cooperation is achieved (where necessary) by combining all individually reachable states and using this as input for successive rounds of individual relaxed planning graph generation. As a result of this process, a coordination point is as-

Algorithm 1: High-level Overview of ADP

Input : MPT $\langle V, I, G, A\rangle$
Output: Plan or $\perp$
1 Calculate Agent Decomposition Φ
2 $S \leftarrow I$
3 CoordPoint(S) [initialises $S.agent$, $S.goals$ and $S.macro$]
4 Greedy BFS from S using h_adp heuristic [When the successor of S is generated it copies $S.agent$, $S.goals$ and $S.macro$ from its predecessor]

Algorithm 2: Decomposition Algorithm

Input : MPT $\langle V, I, G, A\rangle$, Causal Graph CG, $\Phi = \{\}$
Output: Agent Decomposition $\Phi = \{\phi_1, \ldots, \phi_n\}$
1 $\Phi \leftarrow \{\{v\} : v \in V \wedge$ v root node of CG$\setminus$2-way cycles$\}$
2 **repeat**
3 **foreach** $\phi_i \in \Phi$ **do**
4 $\phi_i \leftarrow \phi_i \cup \{v \in V : v$ only successor of $\cup \phi_i\}$
5 $\Phi \leftarrow \Phi$ where agents sharing joint actions are combined
6 **until** Φ *can no longer be refined*

signed an agent (the one with the most individually achievable goals), a set of agent goals (all propositions the assigned agent can achieve alone along with all propositions necessary for other agents to achieve goals requiring cooperation) and a macro heuristic value that estimates the coarse distance to the goal. At non-coordination points, the heuristic value is only updated by the currently assigned agent's progress towards its currently assigned goals.

ADP Details

Algorithm 1 gives a high-level pseudo-code overview of ADP. The algorithm takes a multi-valued planning task (MPT) calculated by the Fast-Downward Planning System (Helmert 2006) as input and consists of an initial preprocessing decomposition phase, followed by greedy best-first search using the ADP heursitic.

An MPT is represented by $\Pi = \langle V, I, G, A\rangle$, where:

- V is a finite set of state variables v, each with an associated finite domain D_v,
- I is a state over V called the initial state,
- G is a partial variable assignment over V called the goal, and
- A is a finite set of (MPT) actions over V.

In what follow we assume the standard definition of preconditions *pre(a)* and effects *eff(a)*.

Decomposition

The decomposition algorithm creates a partitioning of the variable set V into variable subsets ϕ_i for each agent i, which leaves a public variable set P that contains the variables that pertain to the environment. An overview of the decomposition algorithm is shown in Algorithm 2.

Algorithm 3: h_adp Calculation

Input : State S with $S.agent$, $S.goals$ and $S.macro$
Output: h_adp
1 $S.micro \leftarrow$ *hff* $(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
2 **if** *S.micro == 0 or deadend* **then**
3 CoordPoint(S)
4 $S.micro \leftarrow$ *hff* $(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
5 h_adp = S.macro + S.micro

First, a list of potential agents is found from the causal graph by taking every root node that exists in the graph once all two-way cycles have been removed. These root nodes must still have at least one successor node to be considered.

In the next step, the potential decomposition is refined by first extending agent sets to be as large as possible and then reducing the number of sets based on the existence of any joint actions. Agent sets are extended by recursively adding any node of the causal graph that is a successor of only variables already included in that agent set.

Actions in the domain can be categorised based on potential decompositions (partitions). An action is said to be *internal* to agent i for a given decomposition $\Phi = \{\phi_1, \ldots, \phi_n\}$ iff: $\exists v \in pre(a) : v \in \phi_i$, and$v \in pre(a) \rightarrow v \in \phi_i \cup P$. In other words, the preconditions of a must depend on an internal state variable of i and can only change i's internal state variables or the domain's public variables.

A *public action* is any action where: $v \in pre(a) \rightarrow v \in P$, i.e., where the preconditions do not depend on the internal state of any of the agents. An agent's action set is the set of all internal actions of that agent, denoted by Act_i.

An action is *joint* between agents i and j given decomposition $\Phi = \{phi_1, \ldots, \phi_n\}$ iff. $\exists v \in pre(a) : v \in \phi_i$, and$\exists v \in pre(a) : v \in \phi_j$. An action can be joint between multiple agents. In the second stage of the algorithm all agents involved in any joint actions are merged.

In a decomposition returned by ADP, the sets ϕ_i are guaranteed to have the *agent property*. In particular, a variable set ϕ_i (as part of a full decomposition Φ) has the agent property when for all $a \in A$ and variables $v \in V$:

$$v \in \phi_i \wedge v \in \mathit{eff}(a) \rightarrow a \in Act_i.$$

In other words, any agent variable can only be modified by an action of that agent. The proof of this can be found in (Crosby 2014).

Heuristic Calculation

The h_adp heuristic calculation is formed of two parts as shown in Algorithm 3. There is a global *coordination point* calculation that is performed infrequently and is used to pick out a single agent and a set of goals for that agent to attempt to achieve. There is also a micro single-agent heuristic calculation that is identical to that used by FF (Hoffmann and Nebel 2001) on the planning problem restricted to the current chosen agent and its current set of goals and subgoals.

Algorithm 4: Coordination Point Calculation

Input : State S
Output: $S.agent$, $S.goals$ and $S.macro$
1 Iterated Relaxed Planning Graph Generation
2 **if** *Max layer* > 0 **then**
3 | Calculate Subgoals
4 Assign Goals
$S.agent \leftarrow$ agent with most goals with min h_add
$S.goals \leftarrow$ all goals achievable by $S.agent$
$S.macro \leftarrow N \times |G \setminus S.goals|$

Coordination Point Calculation An overview of the coordination point calculation is shown in Algorithm 4. It associates a chosen agent, a goal set and a macro heuristic value with the current state. This extra state information is carried over whenever a successor state is generated and (re)calculated whenever the current agent's goal set is completed or becomes impossible.

Iterated Relaxed Planning Graph Generation. Given a state, each agent generates their full relaxed planning graph for their restricted problem from that state. That is, each agent uses an iterative process to create a graph containing all possible actions it can perform (ignoring delete effects) and all possible propositions that can be reached by performing those actions.

It may happen that some propositions can only be reached if agents cooperate. For example, one agent may need to unlock a door before another can pass through. Because the agents create their own planning graphs (using only their own subproblems) they will not include any parts of the search space only reachable by cooperation. If not all goals are reachable by at least one agent after the first round of relaxed planning graph generation, the collected final state of all the agents is formed and used as input for a subsequent *layer* of relaxed planning graphs. Each successive iteration introduces a new *layer* with the first being *layer* 0. Repeating this process until no more states are added by any agent is guaranteed to cover every reachable state in the full (not decomposed) problem.

Calculate Subgoals: Any time a goal proposition appears for the first time in a layer above 0 this means that it cannot be reached by an agent on its own. In this case, subgoals are calculated. Plan extraction is used to find out which propositions are necessary from a previous layer in order to achieve each goal. These propositions are called subgoals. All subgoals from layer 0 are added as to the agent that achieved them first. Using the door example, if the second agent needs to pass through the door to achieve a goal, the subgoal of unlocking the door will be assigned to the first agent.

Assign Goals: The next part of the coordination point calculation chooses which agent is going to be performing the local search. First, each goal is assigned to the agent that can achieve it with the lowest estimated cost from the relaxed planning graphs. That is, it is assigned to the agent that added it with the lowest h_add value (Hoffmann and Nebel 2001). An agents goal set is then formed of all goals and subgoals assigned to it. The agent with the largest goal set is chosen as $S.agent$ along with all of its goals (and any subgoals that it may have been assigned).

As a final part of the coordination point calculation the value $S.macro$ is calculated. This is used to provide a global heuristic estimate of the distance through the overall search space. This is calculated as $N \times |G \setminus S.goals|$ where N is some large number chosen such that it dominates the single-agent FF heuristic value of a state.

Decomposition of CODMAP Domains

This section discusses the decompositions that ADP finds for the competitions domains used in CODMAP 2015. The results are shown in Table 1. The second column of the table 'Usable' reports whether or not ADP managed to find a usable decomposition for the domain. ADP found a suitable decomposition for nine of the twelve domains in the competition, failing to find one in blocksworld, driverlog and sokoban. This does not mean that no sensible decomposition exists for this domain, but that ADP could not find one that respects the *agent property* and contains no joint actions.

In general, decompositions returned by ADP are consistent across all problem instances for each particular domain. The one exception is Woodworking for which there was at least one problem instances in which ADP could not find a decomposition (represented by an asterisk in the table).

The third column of the table 'Joint' shows whether or not ADP found a decomposition that includes joint actions. The only domain for which this differs is Driverlog for which a decomposition was found (drivers+trucks) but was not used. There is no theoretical reason why the APD heuristic cannot be applied when joint actions exist, however in such domains, the algorithm tends to perform very poorly and much worse than if no decomposition is returned. Note that for some domains a decomposition including joint actions was found part-way through the decomposition algorithm, but the final decomposition did not include any after agents were merged.

The final three columns of the table compare the predefined decompositions for the problems to the decomposition that ADP finds. ADP found the same decomposition in exactly half of the ten remaining domains. In Driverlog and Taxis, ADP found a very similar decomposition including the trucks as well as the drivers in Driverlog and only including the taxis and not the passengers in Taxis. In Depot, ADP finds a completely different decomposition which treats the trucks as agents and also contains a separate agent that includes every single crate in the domain. The two remaining domains Woodworking and Wireless return fairly odd looking decompositions and it is expected that ADP will perform poorly on these domains.

ADP also found some extended agent sets not reported in the table. For example, in satellites the decomposition found by ADP includes the variables representing the state of the instruments' of each satellite with each individual satellite. In the Zenotravel domain the agent sets include the fuel levels for each plane.

Domain	Usable	Joint	Predefined Decomp	ADP Decomp	Match
blocksworld	✗	✗	agents	na	–
depot	✓	✓	drivers + dists + depots	trucks + crates*	✗
driverlog	✗	✓	drivers	drivers + trucks	✗
elevators	✓	✓	lifts(fast/slow)	lifts(fast/slow)	✓
logistics	✓	✓	trucks + airplanes	trucks + airplanes	✓
rovers	✓	✓	rovers	rovers	✓
satellites	✓	✓	satellites	satellites	✓
sokoban	✗	✗	players	na	–
taxi	✓	✓	taxis + passengers	taxis	✗
wireless	✓	✓	nodes + bases	messages by node + messages by base	✗
woodworking	✓(*)	✓	different tools	boards(available) + saw	✗
zenotravel	✓	✓	planes	planes	✓

Table 1: The decompositions found by ADP on the CODMAP problem domains.

Limitations and Future Work

This section briefly states some of the current limitations of ADP and current plans for improvement and extension of the planner in the future. The decomposition algorithm is known to return a decomposition with the agent property but it is currently unknown if it will always find such a decomposition if one exists. Further theoretical work and experimentation with different possible decomposition definitions is planned along with the release of a a standalone decomposer that can be used to find decompositions of PDDL-encoded planning problems.

The ADP heuristic calculation is based on the FF heuristic. However, there are a large number of other successful planning heuristics that have since been developed and it is likely that the overall multiagent approach can be applied to some of these techniques. It will also be interesting to extend ADP to include reasoning about action costs, numeric fluents and other extensions of PDDL or to include more multiagent aspects of the planning problems.

Finally, ADP is currently implemented as a single-threaded process. As each agent is always acting only working on their own internal problem, there is clearly scope for a multi-threaded version in which agents explore the search space independently.

ADP Details Summary

This final section presents a summary of the details of ADP relevant to CoDMAP 2015. ADP is a complete, satisficing (non-optimal) centralised single-threaded planning algorithm. ADP was implemented as a heuristic plug-in for the Fast-Downward planning system (Helmert 2006). The preprocessing of the planning problem into an MPT and the search algorithm are left unchanged. ADP simply calculates a decomposition and provides a heuristic value for each state that is queried during search and also stores a macro-heuristic value, agent, and goalset for each state. ADP is called with the option cost_type=2 and using the lazy_greedy search algorithm. Source code for ADP can be found online at the authors homepage.

ADP ignores the agent factorization presented in the MA-PDDL files, instead determining the agents present (if any) itself. The private/public separation is therefore also ignored. ADP does have an internal representation for private and public facts and actions used for decomposition calculation but does not use this directly for search.

Sometimes ADP will not be able to find an agent decomposition (defaulting to single-agent planning behaviour) or find a different decomposition as to that specified in the CoDMAP files. The details of these cases are explained in an earlier section of this paper.

Two versions of ADP were submitted to the CodMAP planning competition. Planner1 is the ADP implementation described in this paper. Planner2 is a legacy version of the code that is functionally identical except that instead of storing the macro-heuristic value and agent assignment, it (incorrectly) assumes that this can be carried over from the previously searched state. This legacy version was entered out of curiosity and the fact that it has produced some interesting results with some recent work in the planning community showing the possible value of including some element of randomness (essentially what the legacy version does) in the search process.

References

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Crosby, M. 2014. *Multiagent Classical Planning*. Ph.D. Dissertation, University of Edinburgh.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

IPC. 2011. http://www.plg.inf.uc3m.es/ipc2011-deterministic/. Web Site.

Mission Planning for a Robot Factory Fleet

Matthew Crosby[1] and Ronald P. A. Petrick[1]

Abstract— **Planning is becoming increasingly prevalent as a tool for high-level reasoning in real-world robotics systems. This paper discusses the implementation of a high-level 'mission planner' that utilises AI planning techniques to find initial plans for a fleet of robots acting in a manufacturing factory. The paper introduces the system architecture, and then focuses on the ROS-based mission planning component, which requires the translation of low-level robot 'skills' and a world model to a high-level planning domain. This paper also introduces a new algorithm for decomposition-based planning that can find 'balanced' plans in large multi-robot domains where current state-of-the-art techniques fail.**

I. INTRODUCTION

Robots have been used effectively in factories for many years, however, it is only recently that they have begun to take on roles requiring large amounts of autonomy and high-level reasoning capabilities. Autonomy is becoming increasingly necessary due to greater variability in factory-generated products (e.g., end-user customisation) leading to assembly lines that must involve more than just simple repetitions of the same task with the same components. For example, in car manufacturing—the application domain for this work—each car is built to the specification of the user, meaning that the parts required for assembly may be different for each car on the line. Not only does this require more sophisticated assembly line robotics, but robots are starting to be utilised in preparing the parts for delivery to the assembly line. The proposed solution in this paper is to use autonomous mobile robots with the ability to pick parts from around the factory and place them in a *kit* to be delivered to the assembly line at the appropriate time (see Fig. 1).

In our application domain, a centralised controller, called the *mission planner*, is tasked with creating and assigning initial high-level plans to the robots in the fleet. These initial plans are passed to the robots as input for the on-board robot task planners. Given that the robots can act in the environment concurrently, it is desirable that plans are evenly distributed amongst the robots so as to reduce total execution time. Furthermore, each robot in the domain is capable of filling multiple kits at a time and it is the mission planner's job to work out an efficient allocation of kits to robots, as well as to generate the high-level actions needed to fill these kits without obstructing the other robots.

The robot fleet utilises a skills framework which modularises robot capabilities into high-level, symbolic planning actions, meaning that the robots are pre-calibrated to use planning-like constructs. The generated plans are produced in a standard planning format that is used as input to the on-board robot task planners (responsible for reasoning about sensing actions and manage re-planning activities) so that execution can begin immediately. This setup allows the technology to be reused when robot capabilities and the environment change, as happens when the factory floor is rearranged, new parts become available, or new robots are added to the fleet.

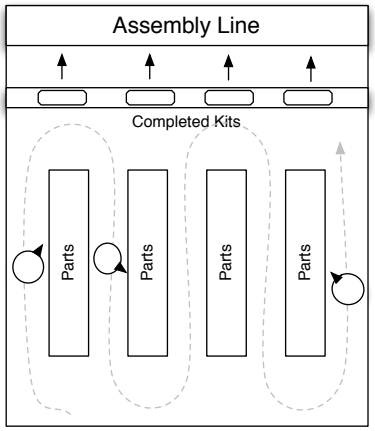

Fig. 1. The factory environment. Multiple autonomous mobile robots navigate a warehouse preparing kits to be brought to the assembly line.

For our application domain it was found that current planning techniques were either too slow (when dealing with multiple robots) or that the output plans did not have an acceptable makespan (i.e., were not distributed efficiently amongst the robots). The main contribution of this paper is a planning algorithm called ADBP that builds on previous work to allow for low makespan plans to be found in a time-frame sufficient for use in the factory.

II. BACKGROUND

Planning has a long association with robotics, stretching as far back as Shakey [1] and Handey [2]. However, the use of planning has often centred around robot-level task planning which, while abstracted from low-level robot control, still needs to deal with contingencies, sensing actions, and other robot-level considerations. This work considers a problem that is abstracted even further: while it avoids some of the complexities that arise in typical robotics planning problems, it also introduces new complications (e.g., multiple robots).

A key component of the framework for the application of this work is the concept of *robot skills* [3]. Robot skills

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement no. 610917 (STAMINA), http://stamina-robot.eu/.

[1]The authors are at the School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, Scotland, United Kingdom. Email: m.crosby@ed.ac.uk, rpetrick@inf.ed.ac.uk

are designed to bridge the gap between robot-level control operations and planning-level actions, by providing a structured representation of both the requirements (preconditions) and expected outcomes (effects) of a robot-level action, encapsulated together with a set of methods for verifying these conditions in the real world through sensing. Thus, skills not only provide a model of preconditions and effects, but also the ability to evaluate and verify these conditions.

The particular skills framework we use provides a standard STRIPS-style PDDL encoding of the domain [4], [5], enabling us to consider utilising successful off-the-shelf planners such as LAMA [6] and FF [7] which support this representation. However, while these planners perform well on small domain instances with two or less robots, they do not scale well with the number of robots.

Another approach to this problem is to use a temporal planning encoding with the features introduced in PDDL 2.1 [8]. Temporal planning allows for easy use of numeric fluents to simplify the encoding, and can be used with planners such as POPF2 [9], a forward chaining partial-order planner that was the runner up in the temporal track of the 2011 International Planning Competition (IPC) [10]. POPF2 automatically attempts to minimise makespan, returning well-balanced plans. Unfortunately, temporal planning does not scale any better than standard single-agent approaches with respect to the number of robots.

Given the previous results, this work instead focuses on multiagent techniques that attempt to exploit the underlying structure inherent in multiagent domains [11]. Due to the centralised nature of the problem, this approach circumvents common multiagent planning challenges such as decentralised planning, strategic elements, concurrency constraints for joint actions, and privacy concerns.

During the course of this work, we tested multiple multiagent planners and found that none could solve problems of the size required for our application: MA-FD [12] could not scale past two robots on our full size problem; MAPR [13] could not scale past 4 agents when using *load-balance*, a goal assignment strategy which attempts to keep a good work-balance among the agents; and both ADP and [14] and MAPR could solve the problem, but only if allowed to return a single-robot solution (i.e., one robot was assigned to do all the tasks regardless of the number of robots in the domain). Overall, the current techniques we tested could only scale beyond a few robots if they ignored the multiagent nature of the problem. This paper therefore focuses on showing it is possible to find truly multiagent plans in this domain that consider all the robots.

III. MISSION PLANNER AND SYSTEM ARCHITECTURE

Fig. 2 gives a simplified overview of the system architecture we use for this work. The architecture contains multiple robots, each with an on-board *task planner* for real-time planning and replanning, and a *skill manager* that handles the translation of low-level robot control nodes (*skills*) to the high-level parts of the system. The *logistics planner* acts as a system manager that controls the information and world model for the whole system and supplies the input connections for the goals and data provided at the factory level. The mission planner takes its inputs about goals and skills from the logistics planner and returns multi-robot plans which are, in turn, passed to the appropriate robots.

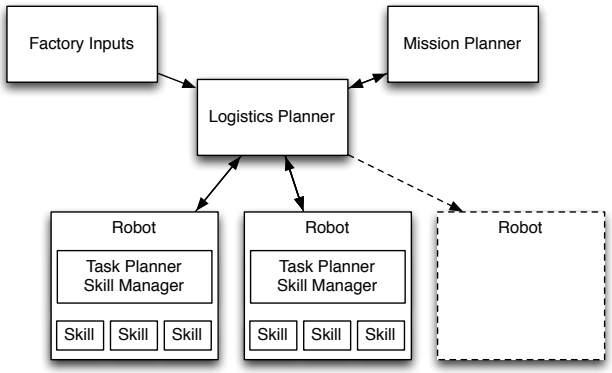

Fig. 2. System architecture diagram.

```
001: NAVIGATE(robot1, startloc, kit1)
002: GETKITBOX(robot1, kit1)
003: NAVIGATE(robot1, kit1, kit2)
004: GETKITBOX(robot1, kit2)
005: NAVIGATE(robot1, kit2, loc4)
006: PICK(robot1, part1, box4)
007: PLACE(robot1, part1, kit1)
008: NAVIGATE(robot1, loc4, loc12)
009: PICK(robot1, part5, loc12)
010: PLACE(robot1, part5, kit2)
...
035: NAVIGATGE(robot1, loc45, locoutput)
036: PLACEKIT(robot1, kit1, outputconveyor)
037: PLACEKIT(robot1, kit2, outputconveyor)
```

Fig. 3. An example plan in which `robot1` completes two kits and places them on the output conveyor.

The mission planner's inputs are a copy of the current world model, the skills available to each robot, and the current goal set. The goal set is comprised of an ordered list of kits that must be assembled, where each kit contains a list of parts that need to be picked. The mission planner outputs a plan for each robot consisting of an ordered list of instantiated skills. An example single-robot plan is shown in Fig. 3. In this plan, `robot1` fills two kits and places them on the output conveyor. In the remainder of the paper we focus solely on the mission planner component.

IV. PLANNING DOMAIN AND ENCODING

To test the high-level planning component of the system, a complete version of the warehouse domain was implemented and tested offline with multiple robots and hundreds of parts.[1] (See Fig. 4 for a pictorial representation of the domain

[1]The current system has also been tested online in a small domain, however, due to the limitations of real-world testing, this domain was restricted to a single robot with only a few parts available. The work in this paper instead focuses on potential future applications of this domain with multiple robots which need to handle all the parts in the factory.

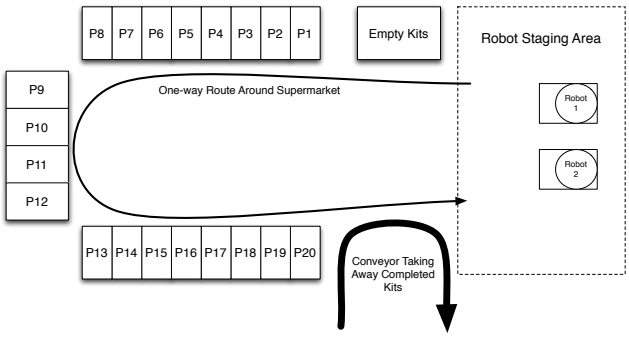

Fig. 4. The warehouse testing domain. Robots must follow the track around the warehouse and cannot overtake other robots. Parts are stored in labelled boxes, and must be appropriately selected to complete a kit. Completed kits are placed on the conveyor. Robots may carry up to two kits at a time and may only pass one another in the staging area.

setting.) In this domain, the robots cannot overtake each other except in the staging area (in which case they can return to their base and then leave before another robot). Each robot has space to carry up to two kits at a time, where a kit is a box with a number of compartments, each designed to fit a specific part. The aim of the mission planner is to provide initial plans for each of the robots in the fleet so that they can complete their task of assembling kits within a specific time frame and without conflicting with other robots.

For the encoding, we model the robots, the position of the kits on the robots, the kits themselves, the available parts, and the locations in the factory. From the skills framework, we are given that robots can perform the actions `NAVIGATE`, `PICK`, `PLACE`, `GETKIT`, and `PLACEKIT`.

One interesting aspect of the domain encoding concerns how to determine when a kit is full and ready to be placed on the conveyor. Multiple approaches can be considered, such as: 1) numeric fluents that count the number of parts in a kit and actions with conditional effects that consider whether a kit was full, 2) a 'close-kit' action that can only be performed when the numeric fluents count the correct number of parts in a kit, 3) both of the above approaches, except with the numeric fluents encoded in number objects with a successor relation, and 4) high-arity predicates or actions with a large number of parameters, coupled with either equality constraints (one for each possible pair of parts), or a careful encoding to ensure parts are guaranteed to be different. For example, a high-arity 'deliver-kit' action requiring 6 parts might include the parameters:

```
(?r - robot ?l - location ?k - kit
    ?p1 - part ?p2 - part ?p3 - part
        ?p4 - part ?p5 - part ?p6 - part)
```

Testing found that none of the above encodings allow for solutions to larger problem instances with any of the planners. One solution that works is to encode kit information as part of the goal for each problem, with separate propositions denoting 'delivered(kit)' and 'in-kit(part)'. While, in general, large goal sets are a problem in planning (e.g., the visit-all domain [15]), they avoid the need for high-arity predicates or parameters, numeric fluents, and inequalities, and are the only method that allowed the temporal and single-agent planners to solve even small domain instances. goal distribution during search performed by ADP.

Fig. 5 shows an overview of the mission planner's operation from input to output. A world model, set of skills, and current goals as used as input. The planning domain is created based on a skills list using a library of precomputed skill-to-PDDL conversions. This list may change depending on the active robots in the domain and the skills they possess.

When creating the problem file for a particular planning instance, the planner does not code any unnecessary information. For example, if a part does not appear in any of the goals then it does not need to be modelled by the planner. Therefore, when creating the problem file, the mission planner iterates over the parts that occur in the goals, looks up their locations and properties in the world model and adds them to the problem instance. Similarly, locations that only contain parts that do not appear in the goals are not included in the domain.

The planning domain and problem are used as input to a planner that returns a single multi-robot plan. The multi-robot plan is then broken down into a plan for each robot by separating out each agent's actions while maintaining the original plan ordering.

To create more efficient planning domains, the planning problems are modified such that the plans need to be pre-processed before being returned as actionable plans to the logistics planner. For example, the world model contains a total strict ordering of all the locations in the domain that respects the route the robots must take around the warehouse. This information is encoded using a `BEFORE` predicate, such as `BEFORE(loc1,loc2)`, and the `NAVIGATE` action can only be used to navigate from `locx` to `locy` if such a predicate exists for each point. In practice, we found that planners perform better if we encode this relationship only for adjacent locations so that, for example, `BEFORE(loc1,loc4)` does not exist. Instead, the robot will have to `NAVIGATE` from `loc1` to `loc2`, then `loc2` to `loc3`, and so on, until it reaches `loc4`. Any time two (or more) `NAVIGATE` actions appear in a plan, they are concatenated to form a single `NAVIGATE` action. From the previous example, the three `NAVIGATE` actions would be concatenated to form `NAVIGATE(loc1,loc4)`, before being sent to the logistics planner.

V. The ADBP Algorithm

Initial tests (see Table I) showed that ADP and MAPR were the best performing planners as the number of robots in the domain scaled, however, they could only return single-robot plans which are not helpful for practical purposes. No planners that could return low makespan plans could scale beyond problem 4 (this also includes MA-FD which could solve 1 and MAPR-lb which could solve 4). We now describe ADBP (Agent Decomposition Balanced Planner) which can solve up to problem 10 in under 10 seconds, and returns multi-robot solutions.

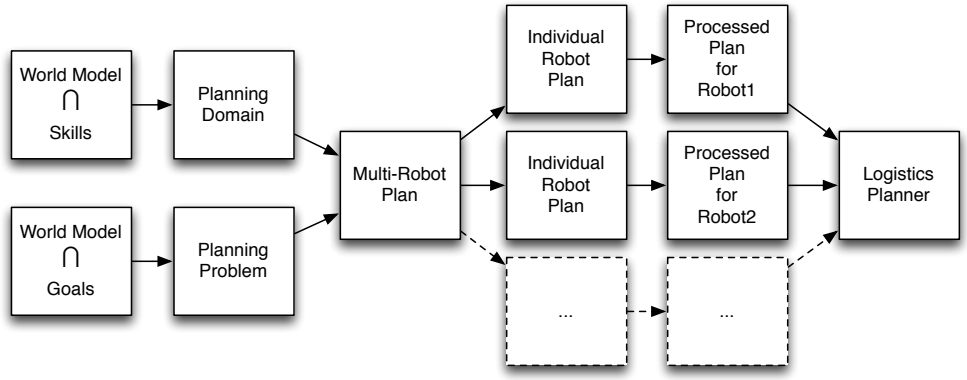

Fig. 5. An overview of the mission planning process from input to output and back to the logistics planner.

TABLE I

TABLE SHOWING THE PERFORMANCE OF PLANNERS ON TESTING DOMAINS AS THE SIZE OF THE PROBLEM INCREASES TO THE SIZE REQUIRED FOR APPLICATION IN THE REAL-WORLD DOMAIN. RESULTS SHOW THE TIME IN SECONDS, THE TOTAL COST, AND THE MAKESPAN WHEN THE OUTPUT IS CONTRACTED TO A MULTIAGENT PLAN WITH JOINT ACTIONS. A '–' MEANS THAT THE PLANNER DID NOT RETURN A PLAN WITHIN 300 SECONDS.

Problem	No. of	No. of	POPF2			FF			LAMA			ADP		
Number	Robots	Goals	time(s)	cost	ms	time(s)	cost	ms	time(s)	cost	ms	time(s)	cost	ms
1	2	4	0.52	29	13	0.01	30	18	0	24	12	0.01	24	24
2	2	4	–	–	–	0.34	54	27	0.01	54	27	0	54	54
3	2	4	–	–	–	0.01	74	37	0.01	74	37	0.01	74	74
4	2	6	–	–	–	7.11	132	74	0.04	111	74	0.02	132	132
5	4	6	–	–	–	–	–	–	–	–	–	0.02	111	111
6	4	8	–	–	–	–	–	–	–	–	–	0.03	148	148
7	6	8	–	–	–	–	–	–	–	–	–	0.05	148	148
8	6	10	–	–	–	–	–	–	–	–	–	0.06	185	185
9	8	10	–	–	–	–	–	–	–	–	–	0.08	185	185
10	10	10	–	–	–	–	–	–	–	–	–	0.11	185	185

Algorithm 1: Heuristic Value Calculation of ADBP

Input : State S, Goals G
Output: h_max, h_square, or h_total

```
Relaxed Planning Graph Generation (full)
if Max layer > 0 then
    Calculate Subgoals
    G ← G ∪ subgoals
foreach agent i do
    foreach Goal g ∈ G do
        if h_add(g, i) > 0 then
            ExtractRelaxedPlan(g, i)
    hff(i) ← RelaxedPlan(i).cost
h_max = max_i hff(i)
h_square = Σ_i hff(i)^2
h_total = Σ_i hff(i)
```

ADBP provides a heuristic value for a given state and is implemented as a derived class of the Heuristic class in Fast Downward [16]. This means that ADBP can be used as a heuristic for different planning search methods, although we found that it was best suited to greedy best-first search.

ADBP uses the same agent decomposition as ADP which partitions the variables of the MPT representation of the problem created by Fast Downward such that each variable either belongs to an agent or to the environment [17]. We found that in all configurations and encodings of the domain tested this decomposition returned the expected partitioning of variables that picked out those that correspond to the robots in the domain.

ADBP's heuristic value calculation is shown in Algorithm 1. Relaxed Planning Graph (RPG) Generation is based on that introduced by FF [7], but modified for the multiagent case. Each agent generates its own RPG based on the problem reduced to only their variables and environment variables. After this, the union of the final state of each agent is then used to create another layer of RPGs. This process repeats until no further propositions can be added by any agent and guarantees that the full relaxed plan space is explored (see [17] for details).

If more than one layer of RPGs have been generated, then subgoals are calculated. Any time a goal proposition appears for the first time in a layer above the first, this means that it cannot be reached by an agent on its own. Therefore, plan extraction [7] is used to find out which proposition is utilised from the previous layer (see [17] for details). All necessary propositions from the first layer are added as subgoals.

In the final step of the heuristic calculation, each agent creates a relaxed plan to every goal in the goal set (including subgoals) that it can achieve. The value $hff(i)$ is the cost of agent i's relaxed plan. This only counts each action once

(even if it is used to reach multiple goal propositions).[2]

A. Heuristic Values

There are three different versions of ADBP, based on the way the individual heuristic values calculated by each agent at each state are combined:

ADBP-max uses the value h_max which is the maximum hff value of all the agents. This discards a lot of the information that has been calculated, but is as close as possible to the estimated makespan of the plan from the current state. As ADBP is primarily concerned with minimising makespan, this is a natural heuristic to investigate. The downsides are that it discards a lot of potentially useful information and has little concern for how close the goal state actually is. For example, 5 agents with $hff = 9$ return a worse heuristic value than if four have $hff = 0$ and one has $hff = 10$.

ADBP-total takes the other extreme and uses the sum of each agent's hff values. This is much further from the makespan heuristic estimate but encodes more information about the distance from the state to the goal. It should be noted that this is different from the single-agent FF value for the state because goals are repeated by all agents that can achieve them and subgoals are included in the calculation. The downside of this calculation is that it does not take the makespan into account at all (beyond the fact the value is calculated over multiple agents).

ADBP-square attempts to sit in the space between the two extremes and uses the sum of squares of the hff values of the agents. The idea is to use all the information available whilst taking the variance between the agent values into account.

VI. Evaluation

The evaluation is split into three parts. The first part discusses the performance of existing state-of-the-art planners on the factory domain, the second part explores ADBP's performance on the factory domain, and the final part briefly shows ADBP's performance in other multiagent domains.

Initial Results: The complete version of the domain was modelled with 10 robots, 10 goals, and a kit size of 6, however, this problem was not solvable by most of the planners in our initial tests. We therefore created smaller problem instances for testing purposes by varying the number of agents and goals. Each problem contains the full number of locations and parts, and parts-per-kit, except for two problems (0 and 1) which were smaller. It can be seen that the limiting factor is related to the number of agents and, once they are increased above 4, then only single-agent solutions can be found.

Along with time and cost, the results also show the makespan of the returned plans. For POPF2, this is simply the makespan as the temporal domain was designed with a direct mapping from actions at a time step in the temporal plan to joint actions. For the other approaches, the makespan can be calculated manually by post-processing plans using the algorithm presented in [18]. For our particular domain, the highest number of actions assigned to a single agent is a good enough approximation of the final makespan, and there was no need to calculate the exact joint plan at this stage. We use this approximation in the results reported in this paper.

ADBP Results: Table II shows the results for the different versions of ADBP on the factory domain (for the same problems as are shown in I). The first thing to note is that the makespans of all the ADBP algorithms are significantly lower than those for the original ADP. All the versions achieve (to some extent) the goal of distributing the plans amongst the agents. The table also shows that ADBP-max is not able to solve any problem beyond number 3. However, this is a problem with four robots, so there is still an improvement over the single-agent approach. This is perhaps somewhat surprising given the apparent lack of information contained in the heuristic value.

The most interesting feature about the results for ADBP-max is not shown in the table, but in the plans it outputs. Both ADBP-total and ADBP-square output plans in which the robots perform many actions in a row, normally completing a goal before another agent moves. The plans returned by ADBP-max contain actions that are interleaved between the different robots: the second agent follows directly behind the first as they navigate the factory. Given that we can compute a reduced makespan plan with post processing, this may not seem important, but the result shows an area for future work where interleaved actions may be more significant.

Multiagent IPC Domains: ADBP was also tested on a set of multiagent IPC domains [15], the results of which are shown in Table III. The results show that ADBP is not competitive in terms of planning speed: the results are given for problem 10 of each set, with ADBP not scaling well to the larger problems. However, it is interesting that it can return plans at all for these domains when considering every agent in each heuristic calculation, and it does achieve reasonable makespans. It should be noted that the behaviour of ADP-max was repeated an it returned interleaved plans on the smaller domains that it could solve.

VII. Conclusion and Future Work

This paper presented a mission planner for autonomous mobile robots operating in a factory environment, using a new decomposition-based planning algorithm that returns balanced multiagent plans. The proposed algorithm improved the quality of the returned plans, whilst still finding plans within the allotted time constraints for the application domain. Variations of the algorithm also exhibit interesting behaviour such as the interleaving of agent's actions, suggesting that this type of approach is promising for future work in multiagent environments.

We also tested our algorithm on multiagent IPC domains, but found that it is not competitive in its current instantiation. However, it is interesting to observe that ADBP works at all for domains with different properties to the warehouse domain. This indicates that the heuristic function is returning

[2]Note that ADBP supports preferred operators. Any action that appears in a relaxed plan (of any agent) that is also applicable in the current state is set as a preferred operator.

TABLE II

TABLE SHOWING THE PERFORMANCE OF THE DIFFERENT VERSIONS OF ADP ON THE TESTING DOMAINS. KEY: ORIG IS THE ORIGINAL ADP ALGORITHM. MAX IS ADBP-MAX, SQU IS ADBP-SQUARE, AND TOTAL IS ADBP-TOTAL.

Problem	No. of	No. of	Time (seconds)				Plan Length				Makespan			
Number	Robots	Goals	orig	max	squ	total	orig	max	squ	total	orig	max	squ	total
1	2	4	0.01	0.01	0.01	0.01	24	24	24	24	24	12	12	12
2	2	4	0	0.01	0.01	0.01	54	54	54	54	54	27	27	27
3	2	4	0.01	0.03	0.02	0.02	74	74	74	74	74	37	37	37
4	2	6	0.02	0.46	13.11	13.07	132	132	153	154	132	66	95	95
5	4	6	0.02	3.91	0.19	0.19	111	174	111	111	111	58	37	37
6	4	8	0.03	–	0.35	0.35	148	–	147	147	148	–	37	37
7	6	8	0.05	–	0.96	0.95	148	–	148	148	148	–	37	37
8	6	10	0.06	–	1.59	1.6	185	–	185	185	185	–	37	37
9	8	10	0.08	–	3.42	3.41	185	–	185	185	185	–	37	37
10	10	10	0.11	–	6.33	6.31	185	–	185	185	185	–	74	74

TABLE III

TABLE SHOWING THE PERFORMANCE OF DIFFERENT VERSIONS OF ADP ON PROBLEM 10 OF SEVERAL IPC DOMAINS. KEY: A IS ADP, A-M IS ADBP-MAX, A-S IS ADBP-SQUARE, AND A-T IS ADBP-TOTAL.

Prob	Search Time (s)						Plan Length						Agent Makespan					
No.	ff	lama	a	a-m	a-s	a-t	ff	lama	a	a-m	a-s	a-t	ff	lama	a	a-m	a-s	a-t
Rovers	0.01	0.02	0.01	0.86	0.19	0.19	37	40	37	37	41	41	13	17	16	17	18	18
Satellite	0.01	0.04	0.02	–	3.58	3.58	35	39	51	–	43	43	14	18	47	–	19	19
Elevators	0.01	0.08	0.01	–	36.28	14.69	29	32	27	–	31	30	15	17	15	–	14	14

a somewhat useful estimate, even when the domain is not symmetrical and not all goals are achievable by all agents. The algorithm is currently at a state where it satisfies its intended role in our application domain, but will need to be improved and updated as the requirements of the environment evolve.

While the algorithm is still at an early stage of development, this paper shows that multiagent heuristic information can be calculated and used effectively in domains previously only solvable by multiagent techniques that can effectively ignore the interactions between agents. This shows that there is perhaps more room than previously thought for investigating different algorithms based on agent decompositions. For example, ADBP currently does not exploit the fact that some actions are not influencing and therefore cannot change the capabilities of the other agents. After a non-influencing action is used to generate a successor state, a reduced heuristic calculation could be performed. We also intend to explore to what extent optimal planning can be employed, and it seems likely it will be possible to find interesting pruning techniques based on the structure afforded by the domain decomposition.

REFERENCES

[1] N. J. Nilsson, "Shakey the robot," AI Center, SRI International, Tech. Rep., 1984.

[2] T. Lozano-Pérez, J. Jones, E. Mazer, and P. A. O'Donnell, "Task-level planning of pick-and-place robot motions," *IEEE Computer*, vol. 22, pp. 21–29, 1989.

[3] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your Robot have Skills?" in *Proceedings of the International Symposium on Robotics (ISR)*, 2012.

[4] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[5] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL – The Planning Domain Definition Language (Ver. 1.2)," Yale Center for Computational Vision and Control, Technical Report CVC TR-98-003/DCS TR-1165, 1998.

[6] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[7] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[8] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[9] A. J. Coles, A. I. Coles, M. Fox, and D. Long, "Forward-Chaining Partial-Order Planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2010, pp. 42–49.

[10] A. J. Coles, A. Coles, A. G. Olaya, S. J. Celorrio, C. L. López, S. Sanner, and S. Yoon, "A Survey of the Seventh International Planning Competition," *AI Magazine*, vol. 33, no. 1, pp. 83–88, 2012.

[11] R. I. Brafman and C. Domshlak, "From One to Many: Planning for Loosely Coupled Multi-Agent Systems," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2008, pp. 28–35.

[12] R. Nissim and R. I. Brafman, "Multi-agent A* for Parallel and Distributed Systems," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012, pp. 1265–1266.

[13] D. Borrajo, "Plan sharing for multi-agent planning," in *Proceedings of the ICAPS Workshop on Distributed and Multi-Agent Planning*, 2013, pp. 57–65.

[14] M. Crosby, M. Rovatsos, and R. P. A. Petrick, "Automated Agent Decomposition for Classical Planning," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2013, pp. 46–54.

[15] International Planning Competition, "http://www.plg.inf.uc3m.es/ipc2011-deterministic/," 2011.

[16] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[17] M. Crosby, "Multiagent Classical Planning," Ph.D. thesis, University of Edinburgh, 2014.

[18] M. Crosby, A. Jonsson, and M. Rovatsos, "A single-agent approach to multiagent planning," in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2014.