Gabriella Cortellessa, Daniele Magazzeni,
Marco Maratea, Ivan Serina (Eds.)

IPS 2015

Proceedings of the 6th Italian Workshop on Planning and Scheduling

Ferrara, Italy, September 22, 2015

Editors' address:
CNR - Consiglio Nazionale delle Ricerche
Istituto di Scienze e Tecnologie della Cognizione
Via San Martino della Battaglia, 44
00185 Rome - Italy
gabriella.cortellessa@istc.cnr.it

King's College London
Department of Informatics
Strand, London WC2R 2LS, United Kingdom
daniele.magazzeni@kcl.ac.uk

Università degli Studi di Genova
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
viale F. Causa,15
16145 Genova, Italy
marco@dibris.unige.it

Università degli Studi di Brescia
Dipartimento di Ingegneria dell'Informazione
Via Branze 38
25123 Brescia, Italy
ivan.serina@unibs.it

Preface

This volume contains the papers presented at IPS 2015, the 6th Italian Workshop on Planning and Scheduling (`http://ips2015.istc.cnr.it`), held within the XIV Conference of the Italian Association for Artificial Intelligence (AI*IA 2015), in Ferrara, Italy, on September 22nd, 2015.

The aim of this series of workshop is to bring together researchers interested in different aspects of planning and scheduling, and to introduce new researchers to the community. Although the primary target of IPS workshops is the Italian community of planning and scheduling, the aim is also to attract an international gathering, fostering contributions and participations from around the world. In particular, this year 12 papers were accepted for presentation at the workshop, involving many authors from Italy and other European countries.

The papers mainly focus on applications of planning and scheduling, metrics and tools, heuristics and planning and scheduling algorithms applied to several domains.

Gabriella Cortellessa, Daniele Magazzeni, Marco Maratea, Ivan Serina

Workshop Organizers

Programme Chairs

Cortellessa, Gabriella	CNR - Consiglio Nazionale delle Ricerche, Italy
Magazzeni, Daniele	King's College London, United Kingdom
Maratea,Marco	Università degli Studi di Genova, Italy
Serina, Ivan	Università degli Studi di Brescia, Italy

Programme Committee

Baioletti, Marco	Università degli Studi di Perugia, Italy
Bernardini, Sara	King's College London, United Kingdom
De Benedictis, Riccardo	CNR - Consiglio Nazionale delle Ricerche, Italy
De Giacomo, Giuseppe	Sapienza Università di Roma, Italy
Della Penna, Giuseppe	Università degli Studi dell'Aquila, Italy
Dimopoulos, Yannis	University of Cyprus, Cyprus
Fratini, Simone	ESA/ESOC, Germany
Garrido, Antonio	Universitat Politecnica de Valencia, Spain
Geffner, Hector	Universitat Pompeu Fabra, Spain
Gerevini, Alfonso Emilio	Università degli Studi di Brescia, Italy
Giunchiglia, Enrico	Università degli Studi di Genova, Italy
Kuter, Ugur	Smart Information Flow Technologies, US
Linares Lopez, Carlos	Universidad Carlos III de Madrid, Spain
Mccluskey, Lee	University of Huddersfield, United Kingdom
Mercorio, Fabio	Università degli Studi di Milano Bicocca, Italy
Micalizio, Roberto	Università degli Studi di Torino, Italy
Oddi, Angelo	CNR - Consiglio Nazionale delle Ricerche, Italy
Orlandini, Andrea	CNR - Consiglio Nazionale delle Ricerche, Italy
Patrizi, Fabio	Free University of Bozen-Bolzano, Italy
Policella, Nicola	ESA/ESOC, Germany
Rasconi, Riccardo	CNR - Consiglio Nazionale delle Ricerche, Italy
Refanidis, Ioannis	University of Macedonia, Greece
Saetti, Alessandro	Università degli Studi di Brescia, Italy
Schaerf, Andrea	Università degli Studi di Udine, Italy
Vallati, Mauro	University of Huddersfield, United Kingdom
Venable,Kristen Brent	Tulane University and IHMC, US

Contents

Regular Papers

Papers not included here and published elsewhere

Evaluating Autonomous Controllers: An Initial Assessment

Pablo Muñoz[1], Amedeo Cesta[2], Andrea Orlandini[2], and María Dolores R-Moreno[1]

[1] Universidad de Alcalá, Alcalá de Henares, SPAIN,
{pmunoz,mdolores}@aut.uah.es
[2] ISTC-CNR – Italian National Research Council , Rome, ITALY
{amedeo.cesta,andrea.orlandini}@istc.cnr.it

Abstract. This work describes the progress of a research line started two years ago that aims at creating a framework to assess the performance of planning-based autonomy software for robotics. In particular it focuses on an open problem in the literature: the definition of a methodology for fairly comparing different approaches to deliberation, while synthesizing a tool to automate large test campaigns for different autonomy architectures under the same robotic platform. We have produced a framework, called OGATE, that supports the integration, testing and operationalization of autonomous robotic controllers. It allows to run series of plan execution experiments while collecting and analyzing relevant parameters of the system under a unified and controlled environment. The software platform supports also for the definition of different metrics for evaluating different aspects of a plan-based controller. This paper, first presents the framework capabilities and the methodology to support experiments, then, briefly describes an autonomous controller that follows a timeline-based deliberation, and finally presents some results obtained exploiting OGATE to perform tests to analyze the performance of the controller over a targeted robot.

1 Introduction

Modern robotics platforms are becoming increasingly sophisticated and capable. The deployment of Artificial Intelligence (AI) planning technologies for robotic autonomy is considered an important technological advancement to endow robots with enhanced abilities once addressing real world scenarios. In this regard, the interleaving of automated planning and execution is a crucial reference problem for Planning and Robotics research communities. Focusing on the literature in autonomous controllers, we can observe several approaches employing different technologies for planning and execution – see [10, 2, 3, 18, 22] for some examples.

One limitation in current research that is shared by different research initiatives is the rather specific validation methodologies, experimental settings and assessment analysis usually performed in a manner that is hardly exportable and

scarcely reproducible on different platforms. This lack of methodology leads to perceive the different evaluations more like a *"proof of concept"* [8] for specific case studies. Then, an interesting open issue consists in defining an evaluation methodology for autonomous controllers capable of being exportable and reproducible with different plan-based schemes for autonomous robotics so as to allow comparisons on the basis of common reference points.

The authors current research initiative is dedicated to the design and development of a software framework to support and facilitate the deployment of control architectures for robotics platforms [17]. In general, the aim is to address the above mentioned open issue by means of the combination of both (i) a *research effort* to discriminate the key factors in planning and execution in order to evaluate the performance of a generic autonomous controllers and (ii) an *engineering effort* to identify requirements to design and implement a general purpose environment to support testing and validation for plan-based autonomous robotics platforms. This paper reports on the current progress of this initiative aiming at providing a well defined methodology and a software framework to assess the performance evaluation of autonomous controllers. In particular, we have defined a methodology that is operationalized in a general and domain independent software framework, called On-Ground Autonomy Test Environment (OGATE), that allows to define relevant metrics according to specific evaluation goals, to define a set of application scenarios to be exploited in order to evaluate autonomous controllers over actual robotic platforms or associated simulators under controlled and reproducible experimental conditions.

Related Works. Evaluating and characterizing autonomous controllers have been investigated in different perspectives. On the one hand, there are theoretical works that aim to define the relevant parameters to measure for an autonomous system [1, 12] and those who try to create valid methodologies for the testing process [13, 11]. On the other hand, there are robotics competitions which allow us to compare different solutions for the same problem with different platforms/controllers [21, 5]. Notwithstanding the relevance of such works, they are mainly focused on functional capabilities and exploit really specific evaluation criteria [19, 16], while others rely on expensive or exclusive robotic platforms. In any case, the complexity of exploiting these systems in automated test campaigns remains an open issue.

Paper structure. The rest of the paper is structured as follows. First, we present a set of general metrics applicable to plan-based autonomous controllers and our proposed methodology to deal with their evaluation. In the next section we provide a general view of the OGATE tool, that is able to perform automatic campaigns to evaluate autonomous controllers. A planetary exploration case study and the robotic platform employed to assess experimental campaigns are presented. Then, considering an autonomous controller in the specific case study, we present and discuss the evaluation of the performance of such controller as a function of the considered metrics within OGATE. Finally, some conclusions end the paper.

2 Evaluation of autonomous controllers

One of the contribution of the paper is to define a general evaluation methodology for supporting the assessment process of autonomous controllers when applied to a robotics platform. In this regard, a sequence of evaluation steps has been identified and is discussed in the following.

In general terms, given an autonomous controller to be assessed, a set of evaluation objectives should be isolated and some specific performance metrics should be identified and defined accordingly. Then, a set of suitable tests should be defined and performed so as to collect relevant information constituting a quantitative basis for the evaluation process. Finally, a synthetic view of measurements should be generated, e.g., through PDF reports, to point out the performance of the controller according to the evaluation objectives and metrics defined in the first phase. More in detail, the methodology proposed to analyze and evaluate autonomous controllers can be thought as the composition of three sequential phases: evaluation design, tests execution and, report and assessment.

Evaluation Design. First, it is required to **identify which is the evaluation objective**. In fact, according to the evaluation target different aspects may result relevant (or not). For instance, measuring the deliberation time or considering the number of dispatched goals in different scenarios could provide relevant information about the behavior of the autonomous controller. In this case, very specific parameters can be considered and analyzed. More in general, a set of parameters applicable to any deliberative system should be considered in order to enable also the possibility to compare performance of different control systems in the same operative scenario.

According to evaluation objectives, a **metrics definition** task is to define parameters that should be measured during execution. This is key as the result of the evaluation strongly depends on the selected metrics. It is important to define (at least) a small set of metrics that can be applicable to different autonomous controllers. Later in the paper, we will provide a set of general applicable metrics which we exploit in our experiments to assess performance evaluation.

Then, the **definition of different scenarios and configurations** to be tested should be implemented. The scenarios can be defined as the set of constraints and goals that the autonomous controller takes as input. However, to also deal with uncertainty, scenarios should be defined considering external agents that can dynamically generate additional goals or possible failures that may occur during execution. Such scenarios definition requires advanced capabilities such as replanning and failure recovering schemes. More than one scenario can be defined in order to investigate the behavior of the autonomous controller under different conditions. We consider three general cases with which an autonomous controller shall deal: (i) **nominal execution**, when everything goes as expected; (ii) **dynamic goal injection**, an extension of the nominal execution in which one or more goals are dynamically included during the system operation; and (iii) **execution failure**, when some components of the system induce a not nominal

behavior so as to force the controller adapting its plan to overcome the contingency. Failures in that case can be due to external perturbations, mechanical failures or degradation of the system over time.

Tests Execution. Performing tests entails the execution of each scenario that is to be monitored. Typically, uncertain and/or uncontrollable tasks are part of the problem, so, each scenario should be performed several times, to collect average behaviors and metric values.

In this regard, a **scenario instantiation** step is required to generate the set of models needed to define a suitable set of planning domains and required goals. Also, autonomous controllers can be deployed with different internal settings and, thus, scenarios instantiation should consider also to enable the execution of tests under different conditions.

Then, actual **tests execution** is needed. This is an important step for instantiating, executing, monitoring and collecting the data after several executions of an autonomous controller in a given scenario. During the tests execution modifications of the nominal execution should be considered, by automatically injecting goals or failures to also test not nominal scenarios.

Report and Assessment. Once all the tests are completed, a **report** on the information gathered during the several executions shall be provided. Reports contains an insight of the controller behaviors, providing values for each metric as well as generating compact views, e.g., by means of graphical representations to support the users while analyzing system performances.

In fact, the information provided within reports is to inform users and enable a **performance assessment** allowing an objective evaluation of the control architecture in the different considered scenarios. After execution, a huge amount of generated data is expected and then a general representation for the data produced is to be defined.

2.1 Metrics definition and graphical report

Definition and presentation of metrics deserve a more detailed discussion and, in the following, a detailed formalization is provided. In the above methodology, a set of metrics M is considered, being a metric denoted as $\mu_i \in M$ and defined in a range $\mu_i^{lb} \geq \mu_i \geq \mu_i^{ub}$ with μ_i^{lb} and μ_i^{ub} are (respectively) the lower and upper bounds for the i metric. Also, for each metric an extra parameter is to be considered, i.e., the weight, μ_i^W, that represents the *relevance* of the metric within the global evaluation. Considering the size of M (i.e., the number of defined metrics) as n, the sum of all weights is supposed to be 100:

$$\sum_{i=0}^{n} \mu_i^W = 100 \tag{1}$$

After execution, the average value for each measured metric, μ_i^V, is considered in the report and this value is considered to compute a *metric score* μ_i^S as follows:

$$\mu_i^S = \left[100 - \left(\frac{100}{|\mu_i^{ub} - \mu_i^{lb}|} \cdot \mu_i^V\right)\right] \cdot \frac{\varepsilon^C}{\varepsilon^T} \quad (2)$$

considering that the upper bound of the metric is the worst score. If the metric value is out of the defined range, its score is 0. Last factor, $\varepsilon^C/\varepsilon^T$, expresses the impact of execution failures in the metrics scores, being ε^C the number of correct executions and ε^T the total runs.

In order to objectively evaluate and compare autonomous controllers we need to use a common set of metrics. In this way, for the evaluation presented in this paper we have gathered the following metrics that can be measured in any autonomous controller:

Operational time. Is the time spent by each part of the controller to update its internal state and schedule its goals for execution

Goal processing time. The time required for each part of the controller to analyze what are its particular objectives.

State processing time. Is the time required by each part of the controller to analyze the incoming information from other part of the system, such as the sensors or other layers states required to evaluate its own status.

Deliberation time. Is the time spent by the controller in generating a long-term plan to achieve its goals.

In the current metrics set presented before we are not taking into consideration the execution time as part of the evaluation. However, it is possible to define metrics in which the execution time is relevant (as a factor of the μ_i^S).

It is worth underscoring that we are not measuring the time that the functional layer takes to complete the actions: our methodology is focused on the deliberation and executive capabilities of a controller. Also, some highly specialized works are focused on analyze the functional support –i.e. [8].

Graphical report. As stated before, a suitable way to provide reports is by means of graphical representations. For example, in Autonomous Levels For Unmanned Systems (ALFUS) [16] and Performance Measures For Unmanned Systems (PerMFUS) [12], a three axis representation based on the mission and environment complexity and human independence is presented.

In a similar way, here, a circular graphic representation, such as the one depicted in fig. 1, is proposed to represent the autonomous controller performance. Such representation presents three different areas. Namely, starting from the center, the Global Score (GS), the execution times and the metrics scores area.

The Global Score (GS) presented in the center of the figure represents a synthetic evaluation for the architecture in a scale between 0 and 10, that can be compared with the Sheridan's model [20]. In that model, the score increases with the level of autonomy demonstrated by the controller, being 10 a fully autonomous system. In our evaluation, a higher score represents a better evaluation as a function of the defined metrics. To compute the GS value, only metrics

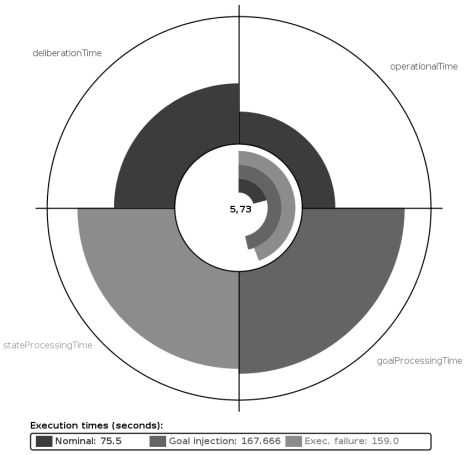


Fig. 1. The OGATE graphical report.

scores are considered, being the score directly proportional to the filled area of the ring, and computed as follows:

$$GS = \frac{\sum_{i=0}^{n} \left(\mu_i^S \cdot \mu_i^W\right)}{1000} \tag{3}$$

Surroundings the GS area, three circular bars are depicted. These bars represent the average time required by the considered autonomous controller to complete each scenario. Starting from the center, these bars represent: the execution time in (i) nominal execution, (ii) in dynamic goal injection and (iii) execution failures.

Finally, the external ring in the chart is decomposed into four quadrants. The smallest circumference of the ring represents the smallest score for a metric ($\mu_i^S = 0$, when the metric score is equal or bigger to its upper bound), while the outside circumference is the best value ($\mu_i^S = 100$, or a metric value closer to the lower bound). In each quadrant there are one or more metric scores represented as a filled circular sector. So, depicting a metric requires metric weight (μ_i^W provided by the user) and metric score (μ_i^S obtained from the execution using eq. 2). As a result, the higher is the weight of the metric and its score, the higher is the filled area of the ring. Then, a evaluation with a GS of 10 is this one in which the metrics score fills all the ring.

This methodology constitutes a generic and reproducible process to evaluate autonomous controllers while considering varying execution scenarios. In this regard, the definition of metrics, i.e., weights, bounds and experimental cases, is the basic step on which rely to reproduce the evaluation results. Also, with the proposed minimal metrics set and graphical representation, we can analyze and compare different aspects of controllers performance in an straightforward manner.

3 The OGATE framework

Autonomous controllers are often tested using hand tailored testbenchs. Such efforts require a great work that is specifically done for the controller and platform under study and, thus, hardly exportable and reproducible.

OGATE constitutes an engineering effort that, taking advantage of the research effort described above, aims to facilitate the definition, execution and reporting of large testbenchs that can be shared and reproduced between different researchers. In this regard, OGATE is a testing environment that can be exploited in order to implement a suitable sequence of evaluation steps for supporting the objective assessment of autonomous controllers.

To achieve these objectives OGATE provides services for instantiating, executing and monitoring the required components of an autonomous controller, while generating reports after tests execution with the collected information under a unified and controlled environment. Furthermore, OGATE also constitutes an interactive tool to help designers and operators of autonomous controllers providing an interface for in-execution control and inspection of the controlled system during execution.

Figure 2 provides a conceptual vision of the OGATE system in which the three relevant modules that constitutes the framework are depicted. These modules are directly related to the main services provided: instantiation is responsibility of the *Mission Specification* module, while execution and monitor are carried out by the *Mission Execution*. Finally, the *Graphical User Interface (GUI)* enables the user to interact with the system in a friendly manner, trying to encapsulate the complexity of the controlled system.

When we have defined the tests objective, metrics and the controller to evaluate, we need to provide OGATE the required information that enables the system to attach the different configurations of the controller under study, the scenarios and the relevant parameters to measure. This is done by means of an eXtensible Markup Language (XML) configuration file that can be created within the OGATE GUI. The tool is general and does not provide the metrics to measure, is responsibility of the user to define them. In OGATE the metrics are represented by its name and the required values to perform the evaluation presented previously –at least the value range, relevance of the metric in the final evaluation and position in the graphical report. The configuration of the autonomous controller entails some engineering knowledge of the controller, while configuring the scenarios –goals and metrics– is more related to operators and planning experts skills. Also, the platform (real or simulated) exploited for the tests shall be properly provided.

A particular capability of the OGATE system is the possibility of automatically generate different scenarios and controller configurations by exploiting a template schema. In the OGATE configuration file it is possible to define different parameters –i.e goals, initial conditions among others– as templates, and, for each template, provide a set of instances. Before execution, OGATE is able to combine all instances possibilities to automatically attach the configuration files to create different scenarios and configurations to be tested.

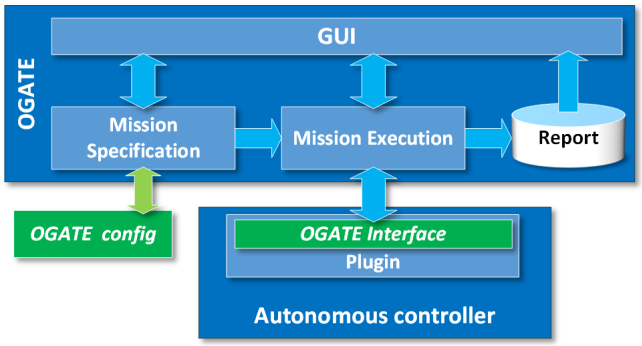


Fig. 2. OGATE concept.

With the information provided in the XML configuration file, OGATE performs the execution by activating the different components of the autonomous controller, accordingly to the configurations provided. Then, the tool is in charge of supervising and monitoring the controller execution by inspecting internal monitors of the different components and retrieving relevant information about its performance. When testing challenging scenarios –dynamic goal injection or execution failures–, OGATE is also responsible of modifying the nominal execution by interacting with the controlled system sending the required telecommands and/or telemetry messages that lead to include a new goal in the controlled system or to modify the execution outcomes. The telemetry/telecommands required shall be provided by the user in a format that is understood by the autonomous controller; OGATE acts as a relay by simulating the operator –goal injection– or the functional support –execution failure.

Finally, the collected information are exploited to generate detailed reports to support assessments based on the analysis of the considered metrics. In this way, OGATE –at the end of the execution– provides a graphical report as the one presented in fig. 1, but also the temporal profiles of the selected metrics and their representative values –minimum, maximum, average and aggregated value– in a Comma Separated Values (CSV) file that can be assessed with other analysis tools. Also, during execution, the OGATE GUI provides to the user the representative metrics values and temporal profiles in real-time.

In order to control and to retrieve data from the controlled system, some parts of the autonomous controller shall be accessible during execution. To deal with this requirement, OGATE implements a set of *interfaces* to enable external system interconnection. Those parts which implement interfacing with OGATE are called OGATE *plugin*. By means of these interfaces, the status of a plugin can be monitored and modified by OGATE, while also the relevant metrics can be gathered and provided to the user during execution. The implementation of such interfaces in the autonomous controller shall be done to exploit the OGATE capabilities; anyway, a similar effort shall be done in order to perform hand tailored tests campaigns. In this sense, the benefits of exploit OGATE are

related to the saved effort related to prepare the tests campaign and the later data collection and analysis.

Regarding this, to work with OGATE, first it is required to perform an engineering effort to implement the required interfaces to enable the control and data inspection of the relevant parts of the autonomous controller and, then, provide the XML configuration file, including the controller configuration, scenarios description and metrics to measure. With this information, OGATE automatically performs tests execution, data gathering and report generation at the end of the execution. Technically, OGATE is implemented in Java and the communication with the autonomous controller is done by means a simple message protocol constructed over TCP/IP.

Finally, OGATE has been designed to directly connect the planning and/or execution layers of an autonomous controller. So, performing experiments with either simulated or actual robotic platforms is not relevant

4 A planetary exploration case study

To assess the test campaign presented later in this paper, we have employed a planetary exploration case study employing the DALA robotic platform. In particular, DALA is an iRobot ATRV robot that provides a number of sensors and effectors, allowing to be used for autonomous exploration experiments. It can use vision based navigation, as well as a Sick laser range finder, being the vision system formed by two cameras mounted on top of a Pan-Tilt Unit (PTU). Also, it has a panoramic camera and a communication facility

In this paper to execute tests, the DALA rover has been simulated by means of a software environment[3] based on OPRS [14], that offers the same robotic functional interface as well as fully replicating the physical rover behaviors (i.e., random temporal duration for uncontrollable tasks).

The objective of the robotic platform is to address a planetary exploration problem. The mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration. During the mission, the Ground Control Station (GCS) may be not visible for some periods. Thus, the robotic platform can communicate only when the GCS is visible. A graphical representation of the problem is presented in fig. 3.

The rover must operate following some operative rules to maintain safe and effective configurations. The conditions that it must hold during the overall mission are: (**C1**) while the robot is moving the PTU must be in the safe position; (**C2**) pictures can only be taken if the robot is still in one of the requested locations while the PTU is pointing at the desired direction; (**C3**) once a picture has been taken, the rover has to communicate the picture to the GCS; (**C4**) while communicating, the rover has to be still; and (**C5**) while communicating, the GCS has to be visible.

[3] DALA software simulator courtesy of Felix Ingrand and Lavindra De Silva from LAAS-CNRS.

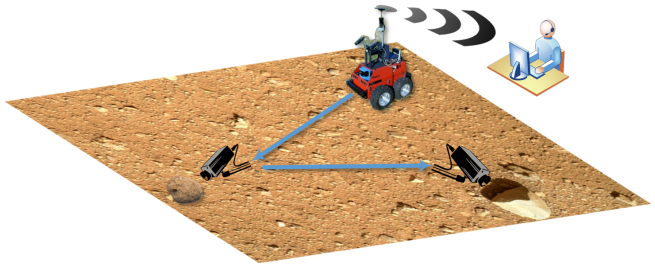


Fig. 3. Example of the planetary exploration with DALA.

5 The GOAC controller

Thanks to the Robotic Department of the European Space Agency (ESA) we have been able to use the Goal Oriented Autonomous Controller (GOAC) [6], an effort from the agency to create a reference platform for robotic software for different space missions. The GOAC architecture is the integration of several components: (i) a timeline-based deliberative layer which integrates a planner, called OMPS [9], built on top of Advanced Planning & Scheduling Initiative (APSI) – Timelines Representation Framework (TRF) [7] to synthesize flexible temporal action plans and revise them according to execution needs; (ii) a Teleo-Reactive Executive (TREX) [22] to synchronize the different components under the same timeline representation; and (iii) a functional layer which combines Generator Of Modules (G$^{\text{en}}$oM) [15] with a component based framework for implementing embedded real-time systems Behaviour Interaction Priority (BIP); [4].

GOAC aims to constitute a general purpose autonomous controller capable to be tailored for different missions/platforms. In that sense, a GOAC instance is a determined and functional configuration to successfully accomplish an objective. The aspect that determines the capabilities of the architecture is the number and hierarchy of the TREX reactors. A reactor is an entity that operates over one or more timelines by (a) deliberating over their required status to achieve the mission goals and/or (b) modifying the status of the timelines as a result of an operation or for an environment change.

In this paper, we exploit a rather simple instance with two reactors as shown in fig. 4: a *Deliberative reactor* and a *Command dispatcher* reactor. The first one is responsible of performing the deliberative task given a domain and a problem encoded in Domain Definition Language (DDL) and Problem Definition Language (PDL) respectively, following a sense-plan-act paradigm. The deliberative reactor can operate with two different planning policies: a *single goal* policy, in which goals are planned as a sequnce (i.e., one after the other), following a sort of batch schema; or, a *all goals* policy, in which a unique planning step generates a solution plan for all the goals. The *Command dispatcher* is in charge of

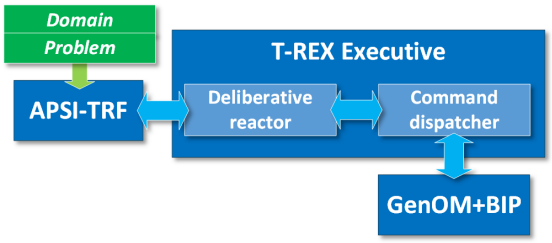


Fig. 4. GOAC instance used in this paper.

executing commands and collecting execution feedback, being connected to the functional layer.

A plan in GOAC has the form presented in fig. 5, in which the involved timelines are depicted. The *Deliberative reactor* generates the different transitions accordingly to the constraints and temporal relations defined in the domain, while the *Command dispatcher* encodes the planned values into actual commands for the rover and uses the feedback provided by the functional layer to produce observations on the low-level timelines that represent the current status for the robot systems.

Finally, it is worth observing that in GOAC the planning and execution are interleaved: while the functional layer is executing a command, the executive is permanently observing the environment, so, it is capable of detect changes and respond in a short time by exploiting reactive planning schemes, instead of perform a replanning process, often more expensive.

6 Experimental results

This section presents the evaluation of the performance for the GOAC autonomous controllers using the planetary exploration case study with the DALA robotic platform introduced above. The evaluation takes advantage of the OGATE framework to automatically perform the tests under different circumstances – nominal execution, dynamic goal injection and execution failure. The experiments have been ran on a PC endowed with an Intel Core i5 CPU (2.4GHz) and 4GB RAM.

More specifically, to perform the tests execution, a suitable GOAC plugin for OGATE has been implemented in order to send to OGATE all the relevant information from the internal components. Also, the different configuration parameters for the GOAC system have been adapted to exploit the OGATE template system. In this way, different templates have been defined to identify the deliberative planning policy, mission goals, temporal uncertainty in action durations and the number of communication opportunities. More in particular, for the different templates we have provided the following set of instances varying the complexity of the problem and the execution conditions: (i) **planning policy**, selecting between the *single goal* or the *all goals*; (ii) **plan length** by

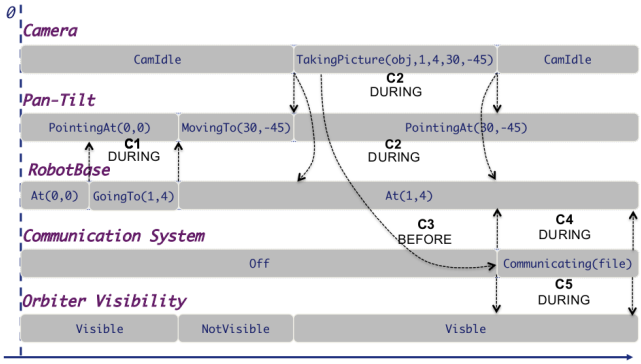


Fig. 5. A plan for the planetary exploration in GOAC.

increasing the number of requested pictures (from 1 to 3); (iii) **plan flexibility** or temporal uncertainty, in which for each uncontrollable activity (i.e., robot and PTU movements as well as camera and communication tasks), a minimal duration is set, but temporal flexibility on activity termination is considered, i.e., the end of each activity presents a tolerance ranging from 0 to 10 seconds; and (iv) **plan choices** as function of the number of communication opportunities spanning from 1 to 4. In general, among all the generated problems instances, the ones with higher number of required pictures, higher temporal flexibility, and higher number of visibility windows result as the hardest.

Then, OGATE has been exploited to: (i) generate the considered scenarios, (ii) carry out all the different controller executions and (iii) collect performance data from the controller. For each execution setting, 10 runs have been performed and average values for the defined metrics are reported. After the collection of performance information in all the considered scenarios, OGATE is able to generates a report containing a wide set of charts corresponding to different control configurations, planning problem instances and execution settings. Finally, for the 10 executions we have considered 4 nominal executions, 3 with goal injection and 3 for execution failure. For the dynamic injection scenario, a new picture is requested between the seconds 40 and 60 after the mission begin and, for the execution failure, a miss-configuration of the PTU occurs during its first reorientation.

For measuring performance, we exploited the metrics presented earlier in the paper. The metrics are captured for both reactors in the GOAC controller and the following ranges (in seconds) have been considered for the one picture scenario: [0, 4] for the operational time; [0 ,1] for the goal processing time; [0, 7] for the state processing time; and [0, 4] for the deliberation time. The ranges for the metrics have been obtained analyzing the results of different executions of the GOAC architecture in the considered scenarios. Also, as more pictures are required, these times are bigger, thus we have increased the previous ranges to

Table 1. OGATE score for all instances clustered by the testbench parameters using two planning policies.

Plan Choices	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Plan Length	***1 picture***							
Plan Flexibility	**All goals**				**Single goal**			
0	5.11	5.82	5.70	5.94	5.25	5.25	5.25	5.22
5	5.78	5.80	4.60	5.48	5.24	5.18	5.78	5.72
10	5.74	3.64	3.07	2.77	5.24	5.43	4.96	5.73
Average	*4.92*				*5.36*			
Plan Length	***2 pictures***							
Plan Flexibility	**All goals**				**Single goal**			
0	5.92	5.48	5.86	5.79	5.49	5.39	5.30	5.28
5	5.42	5.73	6.81	5.70	5.91	5.83	5.18	5.83
10	4.80	1.63	4.67	4.07	6.13	4.30	3.09	3.92
Average	*4.94*				*5.14*			
Plan Length	***3 pictures***							
Plan Flexibility	**All goals**				**Single goal**			
0	4.21	0.00	0.00	0.00	6.11	5.14	5.15	5.11
5	5.44	0.00	0.00	0.00	5.99	5.82	5.86	5.80
10	4.65	1.13	0.00	0.00	5.83	3.27	1.47	2.27
Average	*1.06*				*4.80*			

be fair in the evaluation. Finally, all the metrics have the same weights, being each quadrant reserved for each metric in the graphical evaluation.

Considering all the possible combinations, we obtain 72 possible instances of the GOAC controller. For each of them we obtained a graphical report such as the one presented in fig. 1 – that particular one shows the scenario for one picture with the *single goal* policy, 4 communications opportunities and 10 seconds of temporal flexibility. As we cannot provide a detailed discussion on each possible scenario, we will focus on the *Global Scores* computed for each instance, as stated in table 1.

A first straightforward evidence that can be elicited observing the *Global Score* is that the controller performs similarly with 1 and 2 pictures for both planning policies but has a performance fall for the *all goals* when executing scenarios with 3 pictures that does not occur with the *single goal*.

In all the tested scenarios, the GOAC deliberative component is able to generate a valid plan, but the controller fails in properly completing its execution in some of them. In particular, the execution failure scenario is never completed: when the deliberative component receives the PTU miss-configuration, it does not correspond to its planned states, producing a failure that leads to a system halt. Otherwise, the nominal and dynamic goal injection scenarios are usually completed, except in particular cases of the *all goals policy*: for 1 picture with 2 and 3 communications opportunities and 10 seconds of temporal flexibility; for 2 pictures with 2 communications opportunities and 10 seconds of temporal flexibility; and, for 3 pictures, the *all goals* has several problems to achieve the

mission goals except for the one communication opportunity scenario, in which completes the nominal and dynamic goal injection scenarios for all temporal flexibilities. Instead, the *single goal* policy only fails to perform the dynamic goal injection scenario for the hardest cases: 3 pictures with 3 and 4 communications opportunities and 10 seconds of temporal flexibility. If we analyze the average values for the different planning policies clustering only the scenarios by the number of pictures, we can see that there is no relevant difference between both planning policies for 1 and 2 pictures, but, for 3 pictures, the *single goal* policy seems to be more adequate to be deployed. In fact, the *single goal* policy usually outscores the *all goals* policy.

7 Conclusions

This paper has presented some recent results in addressing the open issue of evaluating the performance of a planning and execution system. To deal with this problem the paper first proposes a methodology to properly guide the testing phase and achieve an objective evaluation. Second, it describes the operationalization of such methodology in a software environment, called OGATE, that is able to perform large test campaigns for different challenging scenarios without user interaction.

The described approach has been used to test the GOAC autonomous controllers. The paper has presented a minimum set of metrics over which the controller performance has been profiled. It is worth underscoring that performing such tests without the described tool requires a large amount of time and a non trivial work to set up different configurations and retrieve information related to the considered metrics. The experiments have been able to characterize the different planning policies of the GOAC deliberative component and the performance of the system as a function of the complexity of the given problem.

Among future works, the definition of a more thorough set of standard metrics constitutes a key immediate step. Additionally, OGATE will be used to compare different plan-based deliberative platforms on the same benchmark tests (a natural extension of the current status).

Acknowledgements

Pablo Muñoz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative (NPI) *Cooperative systems for autonomous exploration missions*. CNR authors are partially supported by the Italian Ministry for University and Research (MIUR) and CNR under the GECKO Project (Progetto Bandiera "La Fabbrica del Futuro"). Authors want to thank to the ESA's technical officer Mr. Michel Van Winnendael for his continuous support.

References

1. Ad Hoc ALFUS Working Group: Autonomy Levels for Unmanned Systems (ALFUS) Framework – Framework Models. Tech. Rep. 1011-II-1.0, National Institute of Standards and Technology (December 2007)
2. Alami, R., Chatila, R., Fleury, S., Ghallab, M., Ingrand, F.: An architecture for autonomy. Field Robotics, Special Issue on Integrated Architectures for Robot Control and Programming 17, 315–337 (1998)
3. Aschwanden, P., Baskaran, V., Bernardini, S., Fry, C., R-Moreno, M.D., Muscettola, N., Plaunt, C., Rijsman, D., Tompkins, P.: Model-unified planning and execution for distributed autonomous system control. In: Association for the Advancement of Artificial Intelligence (AAAI) 2006 Fall Symposia. Washington DC, USA (October 2006)
4. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: 4th IEEE Int. Conference on Software Engineering and Formal Methods. Washington DC, USA (September 2006)
5. Behnke, S.: Robot competitions – ideal benchmarks for robotics research. In: 2006 IEEE/RSJ International Conference on Robots and Systems (IROS) Workshop on Benchmarks in Robotics Research. Beijing, China (October 2006)
6. Ceballos, A., Bensalem, S., Cesta, A., Silva, L.D., Fratini, S., Ingrand, F., Ocón, J., Orlandini, A., Py, F., Rajan, K., Rasconi, R., Winnendael, M.V.: A Goal-Oriented Autonomous Controller for Space Exploration. In: ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation. Noordwijk, the Netherlands (April 2011)
7. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: Developing an end-to-end planning application from a timeline representation framework. In: IAAI-09. Proc. of the The Twenty-First Innovative Applications of Artificial Intelligence Conference. Pasadena, CA, USA (July 2009)
8. Fontana, G., Matteucci, M., Sorrenti, D.G.: RAWSEEDS: Building a benchmarking toolkit for autonomous robotics. In: Amigoni, F., Schiaffonati, V. (eds.) Methods and Experimental Techniques in Computer Engineering, pp. 55–68. SpringerBriefs in Applied Sciences and Technology, Springer International Publishing (2014)
9. Fratini, S., Pecora, F., Cesta, A.: Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. Archives of Control Sciences 18(2), 231–271 (2008)
10. Gat, E.: Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In: the Tenth National Conference on Artificial Intelligence (AAAI). pp. 809–815. San Jose, CA, USA (July 1992)
11. Gertman, D.I., McFarland, C., Klein, T.A., Gertman, A.E., Bruemmer, D.J.: A methodology for testing unmanned vehicle behavior and autonomy. In: Performance Metrics for Intelligent Systems (PerMIS'07) Workshop. Washington, D.C. USA (August 2007)
12. Huang, H.M., Messina, E., Jacoff, A., Wade, R., McNair, M.: Performance measures framework for unmanned systems (PerMFUS): Models for contextual metrics. In: Performance Metrics for Intelligent Systems (PerMIS'10) Workshop. Baltimor, MD, USA (September 2010)
13. Hudson, A.R., Reeker, L.H.: Standardizing measurements of autonomy in the Artificially Intelligent. In: Performance Metrics for Intelligent Systems (PerMIS'07) Workshop. Washington, D.C. USA (August 2007)

14. Ingrand, F., Chatila, R., Alami, R., Robert, F.: PRS: A high level supervision and control language for autonomous mobile robots. In: in Proc. of the 1996 IEEE International Conference on Robotics and Automation (ICRA'96). Minneapolis, MN, USA (September 1996)
15. Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., Ingrand, F.: GenoM3: Building middleware-independent robotic components. In: 2010 IEEE Proc. of the International Conference on Robotics and Automation. Anchorage, Alaska, USA (May 2010)
16. McWilliams, G.T., Brown, M.A., Lamm, R.D., Guerra, C.J., Avery, P.A., Kozak, K.C., Surampudi, B.: Evaluation of autonomy in recent ground vehicles using the autonomy levels for unmanned systems (ALFUS) framework. In: Performance Metrics for Intelligent Systems (PerMIS'07) Workshop. Washington, D.C. USA (August 2007)
17. Muñoz, P., Cesta, A., Orlandini, A., R-Moreno, M.D.: First steps on an on-ground autonomy test environment. In: 5th IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT). IEEE (2014)
18. Nesnas, I., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau, J., McHenry, M., Shu, I.H., Apfelbaum, D.: CLARAty: Challenges and steps toward reusable robotic software. Advanced Robotic Systems 3(1), 23–30 (2006)
19. Orebäck, A., Christensen, H.I.: Evaluation of architectures for mobile robotics. Journal of Autonomous Robots 14, 33–49 (2003)
20. Parasuraman, R., Sheridan, T.B., Wickens, C.D.: A model for types and levels of human interaction with automation. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 30(3), 286–297 (May 2000)
21. del Pobil, A.P.: Why do we need benchmarks in robotics research? In: 2006 IEEE/RSJ International Conference on Robots and Systems (IROS) Workshop on Benchmarks in Robotics Research. Beijing, China (October 2006)
22. Py, F., Rajan, K., McGann, C.: A Systematic Agent Framework for Situated Autonomous Systems. In: AAMAS-10. Proc. of the 9^{th} Int. Conf. on Autonomous Agents and Multiagent Systems (2010)

Quality Metrics to Evaluate Flexible Timeline-based Plans

Alessandro Umbrico[1], Andrea Orlandini[2], Marta Cialdea Mayer[1]

[1] Dipartimento di Ingegneria
Università degli Studi Roma Tre
[2] Istituto di Scienze e Tecnologie della Cognizione
Consiglio Nazionale delle Ricerche, Roma

Abstract. Timeline-based Planning has been successfully applied in several contexts to solve Planning and Scheduling (P&S) problems. A key enabling feature of the timeline-based approach to planning is its capability of dealing with temporal flexibility. Temporal flexibility is an important feature in real world scenarios. Indeed, it can be exploited by an executive system for robust on-line execution of flexible plans in order to absorb possible delays during the execution. In this regard, it is useful to define quality metrics to evaluate the robustness of flexible timeline-based plans. In this paper, a set of quality metrics for flexible timeline-based plans are defined and discussed when applied to a specific timeline-based framework. In fact, we consider the framework EPSL, developed to support the design of P&S applications, that allows the definition of different planners endowed with specific heuristics. Then, an experimental analysis is presented exploiting the planners to solve planning problem instances related to a real-world manufacturing case study. And, finally, an evaluation of planners performance is presented and discussed comparing results also considering robustness of generated plans.

1 Introduction

The Timeline-based approach to planning has been successfully applied in several real world scenarios, especially in space like contexts [1,2,3,4]. Besides these applications, several timeline-based Planning and Scheduling (P&S) systems have been deployed to define domain specific applications, see for example EUROPA [5], IXTET [6], APSI-TRF [7]. Like in classical planning [8], it is important to consider different aspects of the plans generated by timeline-based P&S systems, in order to evaluate their quality. Some temporal metrics have been defined in the literature for temporal networks (often used to represent timeline-based plans) and scheduling [9,10]. In general these metrics characterize the robustness of a schedule by considering the temporal constraints between its activities.

Analogously, the robustness of a timeline-based plan can be evaluated by means of similar measures, which are to be defined without relying on the underlying temporal network (TN). This is particularly important when time flexibility is taken into account. In fact, representing a flexible timeline-based plan as a TN (or a schedule) entails a sort of simplification of the associated plan structure causing a lost of information on the

"dependencies" among its components which can usefully be taken into account. Such information is useful both to generate the plan, as described in [11], and to make a more detailed analysis of the temporal features which are relevant to evaluate the overall plan robustness. For instance, some timeline in the plan can "dominate" the behavior of other timelines, since its temporal deviations are most likely to propagate to the others.

This paper represents a first step towards the characterization of temporal qualities of flexible timeline-based plans. After a brief presentation of the basic concepts underlying flexible timeline-based planning, some of such metrics are introduced. Their concrete application is shown by using planners implemented in the EPSL (*Extensible Planning and Scheduling Library*) framework [12]. EPSL is a domain independent, modular and extensible software environment supporting the development of timeline-based applications. Recently, some improvements concerning the representation and solving capabilities of EPSL have been presented in [11]. Specifically, the framework is enriched with the capability to model and reason on renewable resources and domain independent heuristics supporting the planning process. The general structure of EPSL allows for preserving "past experiences" by providing a set of ready-to-use algorithms, strategies and heuristics that can be combined together. In this way, it is possible to develop and evaluate different solving *configurations* in order to find the one which better addresses the features of the particular planning problem to be solved. This paper resorts to the EPSL framework and its application to a real-world manufacturing case study, in order to evaluate and compare the performances of planners using different heuristics and the robustness of the generated plans.

2 Flexible Timeline-based Planning

This section briefly introduces the main concepts to define flexible timeline-based plans, along the lines of [13]. The timeline-based approach to P&S aims at controlling a complex system by synthesizing temporal behaviors of its features in terms of timelines. A planning domain is modeled as a set of features, represented by *multi-valued state variables*, that must be controlled over time. Causal and temporal constraints specify, for each feature, the set V of the values the variable may assume, the allowed value transitions (by means of a function $T : V \rightarrow 2^V$), and the allowed minimum and maximum duration of each valued interval (by means of a function D associating to each value $v \in V$ a pair of time values). Moreover, the values of different state variables may be linked by (so-called) *synchronization rules*, requiring that, for every time interval where a given state variable x has the value v, there exist other time intervals where either the same or other state variables assume some given values, which are related by some temporal relation. All these constraints are specified in the *domain specification*. The planning process aims at synthesizing temporal flexible plans that satisfy the domain constraints and fulfill some given goals.

The evolution of a single temporal feature over a temporal horizon is called the timeline of that feature. In general, plans synthesized by temporal P&S systems may be temporally flexible. They are made up of flexible timelines, describing transition events that are associated with temporal intervals (with given lower and upper bounds), instead of exact temporal occurrences. In other words, a flexible plan describes an envelope of

possible solutions aimed at facing uncertainty during actual execution. In this regard, they can be exploited by an executive system for robust execution. Some formalizations have been recently proposed to describe timelines and timeline-based plans [13,14].

Broadly speaking a timeline consists of a sequence of values that are temporally qualified by means of *tokens*. A token specifies the temporal interval (start and end times) assigned to a specific value over a timeline. *Flexible tokens* specify a flexible interval for values over timelines (i.e. values have a start interval and end interval, instead of time points) and, as proposed in [13], they can be defined as follows (where $\mathbb{T}$ is the set of "time points" and $\mathbb{T}^\infty$ denotes $\mathbb{T} \cup \{\infty\}$):

Definition 1 *Let $x = (V, T, D)$ be a state variable describing the set of allowed values V, the value transition function T and the value duration function D for a domain feature. A flexible token for the state variable x is a tuple of the form*

$$(x^i, v, (b, b'), (e, e'), (d, d'))$$

where $i \in \mathbb{N}, b, b', e, e', d \in \mathbb{T}, d' \in \mathbb{T}^\infty, v \in V, b < e'$ and $d_{min} \leq d < d' \leq d_{max}$, where $(d_{min}, d_{max}) = D(v)$. The element x^j is the token identifier.

Definition 2 *A flexible timeline for x is a finite sequence of flexible tokens for x, whose identifiers are $x^0, x^1, ..., x^k$:*

$$\begin{aligned} FTL_x = (x^0, v_0, (b_0, b'_0), (e_0, e'_0), (d_0, d'_0)) \\ ...(x^k, v_k, (b_k, b'_k), (e_k, e'_k), (d_k, d'_k)) \end{aligned}$$

where $b_0 = b'_0 = 0$, $e_k = e'_k = H$ is the temporal horizon of the timeline and for all $i = 0, ..., k-1$, $e_i = b_{i+1}, e'_i = b'_{i+1}$ and $v_{i+1} \in T(v_i)$.

A flexible token represents the set of its instances, i.e. the set of all non-flexible tokens that satisfy the value duration constraints. Similarly a *flexible timeline* FTL_x represents the set of its instances. Namely, an instance of a *flexible timeline* FTL_x is made up of a sequence of instances of the tokens of FTL_x and an instance of a set **FTL** of timelines is a set of instances of the timelines in **FTL**.

However the representation of *flexible plans* must include also information about the relations that have to hold between tokens in order to satisfy the synchronization rules of the planning domain. Thus, the representation of flexible plans must include also a set of temporal relations on tokens, guaranteeing that such rules are satisfied.

Let us consider two flexible tokens, with token identifiers x^i and y^k, belonging respectively to the state variables x and y. A synchronization rule of the domain may require that the flexible token x^i precedes the flexible token y^k. In such a case the set of temporal relations of included in the flexible plan must contain the relations x^i before y^k (i.e. $e'_x < b_y$) in order to satisfy the synchronization rule of the domain.

Like in [15], a small set of primitive temporal relations can be considered, in terms of which all the usual quantitative constraints can be defined, such as, for instance, the relations x^i $\text{contains}_{[lb_1,ub_1][lb_2,ub_2]}$ y^k and x^i $\text{overlaps}_{[lb_1,ub_1][lb_2,ub_2]}$ y^k (where $lb_j \in \mathbb{T}$ and $ub_j \in \mathbb{T}^\infty$, for $j = 1, 2$) .

Definition 3 *A flexible plan Π over the horizon H is a pair (**FTL**, R), where **FTL** is a set of flexible timelines over the same horizon H and R is a set of relations on tokens, involving token identifiers in some timelines in **FTL**.*

A flexible plan represents the set of its instances. An instance of a plan Π=(**FTL**, R) is an instance of **FTL** that respects the relations in R. When there are different ways how a synchronization rule can be satisfied by the same set of flexible timelines **FTL**, each flexible plan represents a choice among them, and different plans with the same set **FLT** and different sets of relations R represent different ways to satisfy synchronization rules.

A planner like those described in the following can leave timelines *open* on the right, i.e. they can leave an *undefined* temporal interval at the end of a timeline. This means that the planner does not *decide* which is the temporal evolution of the timeline after its last meaningful token.

3 Characterizing Robustness for Timeline-based Plans

Given a *flexible timeline-based planner* it is important to define some metrics that allow to characterize the capacity of the generated plans to absorb temporal deviations, i.e. their *robustness*. In this section, some quality metrics concerning temporal features of plans are introduced.

There are several works in the literature that define quality metrics for evaluating plans and planning algorithms [8,16,17,18]. In this regards we consider temporal metrics such as *fluidity* and *disruptibility* (introduced by [9] to characterize the *robustness* of schedules on temporal networks), in order to check temporal flexibility and provide an assessment of the robustness of timeline-based plans. In particular we adapt fluidity and disruptibility metrics to timelines and define the notion of the *makespan* of a timeline to indicate the "useful" portion of a timeline. Note that the term "makespan" is typically used in scheduling problems to express the maximum duration of a schedule. Here, we are using the same term with a slightly different meaning.

The *timeline fluidity* metric is an estimate of the capacity of the timeline to absorb temporal deviations w.r.t. other timelines. It is defined as follows:

Definition 4 *If FTL_x is a flexible timeline for the state variable x, the fluidity of the timeline w.r.t. the other timelines FTL_y of the plan is*

$$\xi(FTL_x) = \sum_{x^i \in FTL_x, y^j \in FTL_y} \frac{\rho(x^i, y^j)}{H \times n \times (n-1)} \times 100$$

where x^i is a token in the timeline FTL_x, y^j is a token in a timeline $FTL_y \neq FTL_x$, H is the temporal horizon of the plan, and n is the number of tokens involved in the computation.

Fluidity is computed by taking into account the temporal slack between tokens, that is a measure of the temporal flexibility between the end time of a token and the start time of another one:

$$\rho(x^i, y^j) = |d_{max}(x^i_{end}, y^j_{start}) - d_{min}(x^i_{end}, y^j_{start})|$$

where d_{max} and d_{min} are respectively the maximum and minimum allowed temporal distances between the end time of x^i and the start time of y^j.

The higher is the value of the *fluidity* of a timeline FTL_x, the higher is the capacity of other timelines of the plan to absorb its temporal deviations. Namely the higher is the value, the lower is the risk of cascading changes on other timelines of the plan. This metric provides a measure of temporal dependencies among the timelines of the plan.

The *timeline disruptibility* metric measures the amount of changes in a plan caused by the introduction of a delay in a timeline and is defined as follows:

Definition 5 *If FTL_x is a flexible timeline for the state variable x, the disruptibility of the timeline w.r.t. other timelines FTL_y of the plan is*

$$\psi(FTL_x) = \frac{1}{n} \sum_{x^i \in FTL_x, y^j \in FTL_y} \frac{\rho(0,x^i)}{|\{y^j : y^j \in \Delta(x^i,\delta)\}|}$$

where x^i and y^j are token of the timelines FTL_x and FTL_y, respectively. $\Delta(x^i, \delta)$ is the set of tokens in the plan that change after the introduction of a delay δ on the token $x^i \in FTL_x$.

Disruptibility is computed by taking into account the slack $\rho(0, x^i)$ of tokens, which is a measure of the temporal flexibility between the temporal origin of the timeline and the start time of the token.

$$\rho(0, x^i) = |d_{max}(0, x^i_{start}) - d_{min}(0, x^i_{start})|$$

It is a measure of the flexible temporal allocation of the token over the timeline.

The disruptibility metric $\psi(FTL_x)$ counts the number of changes (temporal deviations) in the plan due to a temporal delay on tokens x^i of the timeline FTL_x.

Finally the *makespan* metric of a timeline is a measure of the temporal flexibility between the last (meaningful) token of a timeline and the temporal horizon, when the timeline leaves an undefined temporal interval at its end.

Definition 6 *Given a flexible timeline FTL_x for the state variable x with k tokens, the makespan of the timeline is*

$$\mu(x) = \frac{\rho(x^k,H)}{H} \times 100$$

where $\rho(x^k, H)$ is the slack between the last token of the timeline $x^k \in FTL_x$ and the horizon H

$$\rho(x^i, H) = |d_{max}(x^i_{end}, H) - d_{min}(x^i_{end}, H)|$$

It is a measure of the portion of timeline left to be used.

The higher is the value of $\mu(FTL_x)$, the larger is the width of the flexible distance between the last token of FTL_x and the horizon.

In the next section we consider a case study in order to make an experimental evaluations of the different planners we have defined by means of our timeline-based planning

framework EPSL. In particular we aim at characterizing qualities of plans generated using different planner configurations. It is also interesting to evaluate relations among the defined metrics as, in some cases, metrics may be in contrast. This means that it is not possible to obtain a plan with the maximum level of all desired qualities but that the planner must be carefully configured in order to obtain the desired balance among all desired qualities

4 Extensible Planning and Scheduling Library

EPSL [12] is a layered framework built on top of APSI-TRF[1] [7]. It aims at defining a flexible software environment for supporting the design and development of timeline-based applications. The key point of EPSL flexibility is its interpretation of a planner as a "modular" solver which combines together several elements to carry out its solving process.

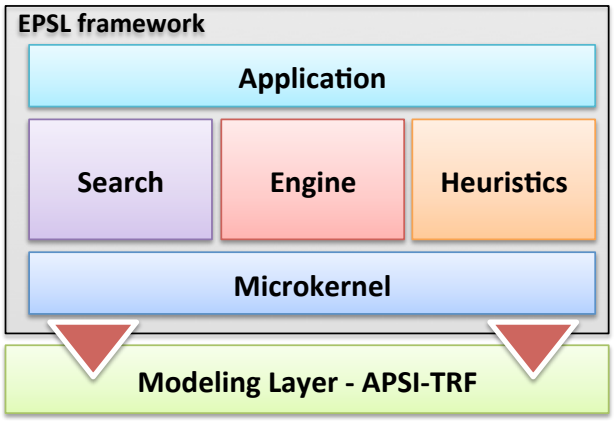


Fig. 1. EPSL Architectural Overview

Figure 1 describes the main architectural elements of the architecture of EPSL. The *Modeling layer* provides EPSL with timeline-based representation capabilities to model a planning domain in terms of *timeline*s, *state variable*s, *synchronization*s and manage *flexible plan*s. The *Microkernel layer* is the key element which provides the framework with the needed flexibility to "dynamically" integrate new elements into the framework. It is responsible to manage the lifecycle of the solving process and the elements composing the application instances (i.e. the planners). The *Search layer* and the *Heuristics layer* are the elements responsible for managing strategies and heuristics that can be used during the solving process.

[1] APSI-TRF is a software framework developed for the European Space Agency by the Planning and Scheduling Technology Laboratory at CNR (in Rome, Italy).

The *Engine layer* is the element responsible for managing the portfolio of algorithms, called *resolver*s, available to EPSL-based planners. Resolver encapsulate the logic for refining timeline-based plans. Each resolver, solves some specific conditions (called *flaws*) that threat the completeness or correctness of a plan. The set of available resolvers characterizes the expressiveness of the framework. Namely they determine what EPSL-based planners can actually do to solve problems.

Finally the *Application layer* is the top-most element which carries out the solving process and finds a solution if any. EPSL architecture allows to define modular planner instances which can be configured in different ways according to the particular feature of the problem to address. An EPSL-user configure planners by selecting the elements (e.g. heuristics, strategies and resolvers) that compose the application instance. Similarly the user can also extend EPSL capabilities by integrating domain-specific elements.

EPSL solving approach is a standard plan refinement search procedure which can be adapted to the particular problem to solve by changing the the planner configuration. As a matter of fact the particular strategy or heuristic applied can strongly affect planner behaviour and performances. In particular, the planner has two important choice points during a the search: (i) *node selection* choice concerning the selection of the search space node to expand next and (ii) *flaw selection* choice concerning the selection of the flaw to solve next in order to refine the current plan (i.e. node expansion).

We focused our attention on the *flaw selection* choice which is not a backtracking point of the search but, in our experience, a crucial aspect to enhance planner performances. Indeed, a careful selection of the next flaw to solve can *prune* the search space by cutting off branches that would not lead to solutions. Flaws can have dependencies, indeed, and the resolution of a flaw can simplify the resolution of other flaws in the plan. In this regard, EPSL allows to define *heuristic*s that support the search in selecting the most promising flaws to solve. Typically these *heuristic*s encapsulate some evaluation criteria that allow to rate plan flaws by taking into account one or more feature. In [11], we developed a domain independent heuristic, called *Hierarchical Flaw Selection heuristic* (HFS), which rate flaws by analyzing the *hierarchical* structure of a timeline-based domain.

4.1 Hierarchical Flaw Selection heuristic

A timeline-based domain specifies relations between different timelines by means of *synchronization rule*s. Thus, given a timeline A and a timeline B, a synchronization rule $S_{A,B}$ from a token $x \in A$ to a token $y \in B$ implies a *dependency* between these timelines. Namely, tokens on timeline B are subject to tokens on timeline A.

Therefore, it is possible to build a *Dependency Graph* (DG) among timelines by looking at synchronization rules. Figure 2(a) shows a set of timelines with *synchronization rules* and Figure 2(b) shows the resulting dependency graph (note that the dependency graph defined here is different from the graph used in EUROPA2 [19]).

The DG encodes dependencies among timelines, and a hierarchy can be extracted by analysing the graph. An edge from a node A to a node B in the DG represents a dependency between timeline A and timeline B (i.e. tokens of timeline B depend on tokens of timeline A). Consequently the hierarchy level of timeline A is not lower than

the hierarchy level of B. If no path in the DG connects B to A, then A is at a higher level in the hierarchy than B (i.e. timeline A is more independent than timeline B). Conversely if A is connected to B and vice-versa in the DG (i.e. a cycle is detected) then timelines A and B have the same hierarchical level, and they are said to be hierarchically equivalent. For instance the hierarchy extracted from the DG in Figure 2 is $A \prec C \prec B \prec D$.

Usually planning domain specifications that follow a hierarchical modeling approach (like the approach described in [11]), generate a non-flat hierarchy of timelines (and sometimes even an acyclic DG).

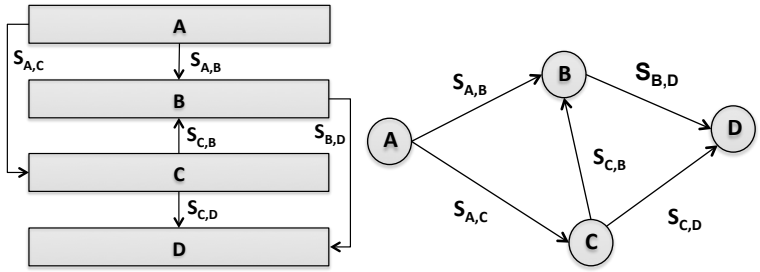


Fig. 2. From Synchronization rules to Dependecy Graph: (a) domain timelines and synchronization rules; (b) the dependency graph resulting from synchronization rules between timelines

The HFS exploits this hierarchy to define a *flaw hierarchy feature* and characterize the *independency* degree of plan flaws. The idea is to solve first "independent" flaws, i.e. flaws belonging to the top most timeline in the hierarchy (e.g. flaws on timeline A w.r.t. Figure 2), in order to *simplify* the resolution of "dependent" flaws. In addition to the hierarchy feature, HFS uses a *flaw type feature* to define a structure for the solving process and the *flaw degree feature* to characterize the *criticality* of a flaw (similarly to the *fail first principle* in constraint satisfaction problems).

The HFS selects the best flaw to solve next by combining together all the features described above as a *pipeline* of *filter*s:

$$\Phi^0(\pi) \xrightarrow{f_h} \Phi^1(\pi) \xrightarrow{f_t} \Phi^2(\pi) \xrightarrow{f_d} \Phi^3(\pi) \rightarrow \phi^* \in \Phi^3(\pi)$$

where f_h filters plan flaws according to the *flaw hierarchy* feature (i.e. it returns only the subset of flaws belonging to the most independent timeline of the hierarchy), f_t filters flaws according to the *flaw type* feature and f_d filters flaws according to the *flaw degree* feature. Then, given a set of flaws of a plan $\Phi^0(\pi)$ every filter extracts the subset of the relevant flaws according to the related feature. The pipeline resulting set $\Phi^3(\pi) \subseteq \Phi^0(\pi)$ is composed by flaws representing *equivalent choices* from the heuristic point of view, so HFS randomly select the "best" one to solve next $\phi^* \in \Phi^3(\pi)$.

5 Experimental Evaluation

The objective of the experimental analysis is to evaluate flexible timeline-based plans generated by EPSL-based planners with different configurations. In particular we evaluate plans by considering the *fluidity* and *makespan* metrics[3] as well as the planners time performances. In this regards, we use two configurations of EPSL-based planners by selecting two different *heuristic* functions: (i) the *HFS* planner, using the HFS heuristic; (ii) the *TFS* planner (*Type Flaw Selection* planner), which uses a heuristic function based only on the *flaw type* feature to select flaws (configuration used before the introduction of the HFS heuristc).

5.1 A pilot plant

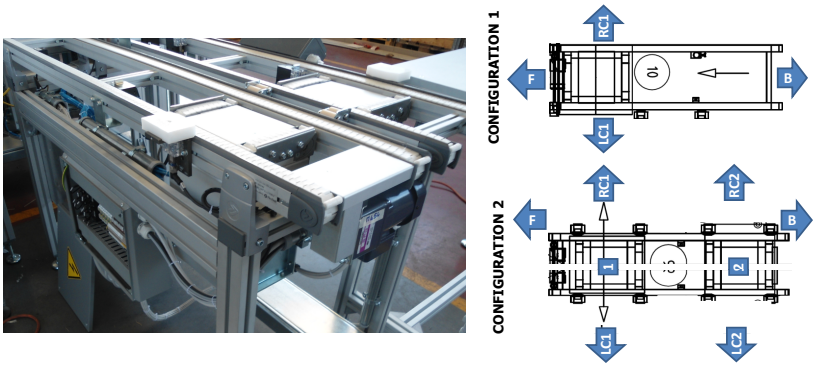
Here, we consider a pilot plant involved in an on-going research project called *Generic Evolutionary Control Knowledge-based mOdule* (GECKO): a manufacturing system for Printed Circuit Boards (PCB) recycling [20]. The objective of the system is to analyze defective PCBs, automatically diagnose their faults and, depending on the gravity of the malfunctions, attempt an automatic repair of the PCBs or send them directly to shredding.

The pilot plant contains 6 working machines (M1,..., M6) that are connected by means of a Reconfigurable Transportation System (RTS), composed of mechatronic components, i.e., transport modules. Figure 3(a) provides a picture of a transport module. Each module combines three transportation units. The units might be either unidirectional or bidirectional; specifically the bidirectional units enable the lateral movement (i.e., cross-transfers) between two transportation modules. Thus, each transport module can support two main (straight) transfer services and one to many cross-transfer services. Figure 3(b) depicts two possible configurations.

Configuration 1 supports the forward (F) and backward (B) transfer capabilities as well as the left (LC1) and right (RC1) cross transfer capabilities. Configuration 2 extends Configuration 1 by integrating a further bidirectional transportation unit with cross transfer capabilities LC2 and RC2. The maximum number of bidirectional units within a module is limited just by its straight length (three, in this particular case). The transport modules can be connected back to back to form a set of different conveyor layouts. The manufacturing process requires PCBs to be loaded on a fixturing system (pallet) in order to be transported and processed by the machines. The transportation system is to move one or more pallets and each pallet can be either empty or loaded with a PCB to be processed.

Such an RTS generally allows for a number of possible routing solutions for each single pallet which is associated to a given destination. Transport modules control systems have to cooperate in order to define the paths the pallets have to follow to reach their destinations. These paths are to be computed at runtime, according to the actual status and the overall conditions of the shop floor (i.e., no static routes are used to move pallets).

[3] In this preliminary experimental analysis, *disruptibility* metric does not provide relevant data



(a) (b)

Fig. 3. (a) A transport module; (b) Their transfer services

The description of the distributed architecture and some experimental results regarding the feasibility of the distributed approach w.r.t. the part routing problem can be found in [21]. *Transportation Module*s (TMs) rely on P&S technology to synthesize activities for supporting the work flow within the shop floor. As a matter of fact TM agents are endowed with Timeline-based planners (build on top of EPSL framework) that assess modules' internal capabilities during the part routing computation process and build modules' plans for coordination and transportation task.

Figure 4 shows the timeline-based model of a generic *transportation module* (TM) of the GECKO case study extended with energy consumption resource to estimate energy consumption profile of transportation tasks. The *Channel* state variable models the high-level transporting tasks the TM is able to perform. Each value of the *Channel* state variable models a particular transportation task indicating the ports of the module involved in the execution of the task. For instance *Channel_F_B* models the task of transporting a pallet form port F to port B w.r.t. Figure 3(b).

The *Change-Over* state variable models the set of internal configuration the transportation module can assume in order to actually exchange pallets with other modules. Namely configurations identify the internal paths a pallet can follows to traverse the module. For instance, *CO_F_B* in Figure 4 represents the configuration needed to transport a pallet from port F to port B.

The *Energy-Consumption* resource models the *energy consumption* policy o the TM. It estimates the energy consumption of transportation tasks of the module, i.e. the energy requirements for channel activities of the module. Activities such as moving conveyors or cross transfers require energy to be performed and we model a system requirement which entails that the instant energy consumption of TMs cannot exceed a predefined limit for safety and optimization reasons of the physical device. In this way, the planner must organize activities in order to not violate the energy consumption constraint of the module.

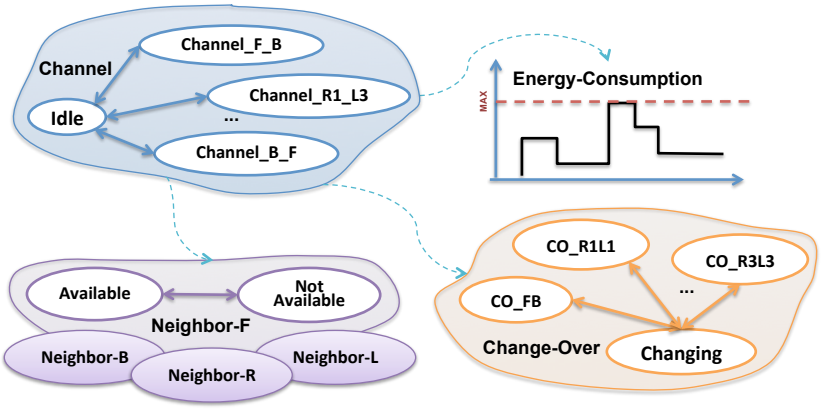


Fig. 4. Timeline-based model for a full instantiated TM

A set of *external state variable*s complete the domain by modeling possible states of TM's neighbor modules. Neighbors are modeled by means of external state variables because they are not under the control of the module. Namely, the TM cannot decide the state of its neighbors. However it is important to *monitor* their status because a TM must cooperate with them in order to successfully carry out its tasks. For instance the TM must cooperate with *Neighbor_F* and *Neighbor_B* to successfully perform a *Channel_F_B* task. Therefore *Neighbor_F* and *Neighbor_B* must be *Available* during task "execution".

Finally a set of *synchronization rule*s specify how a TM *implement*s its *channel* tasks. Figure 4 groups these rules in the dotted arrows between state variables for readability reasons. These rules specify *operative* constraints (causal and temporal constraints) describing the sequence of internal configurations and "external" conditions needed to safely perform *Channel* tasks. For instance, a synchronization rule for the *Channel_F_B* task require that the module must be set in configuration *CO_F_B* and that neighbor F and neighbor B must be *Available* during the "execution" of the task.

5.2 Evaluating Plan Robustness

In the GECKO case study we have defined four planning domain variants by considering different physical configurations of a TM of the manufacturing plant, i.e., by varying the number of cross-transfer units composing the module and under the assumption that module's neighbors are always available for cooperation. Namely, the tests were run on the following planning domains: (i) *simple*, the configuration with no cross transfer; (ii) *single*, the configuration with only one cross transfer unit; (iii) *double*, the configuration with two cross transfer units; (iv) *full*, the configuration with three cross transfer units (the maximum allowed in the case study we considered). The higher is the number of available cross transfers, the higher is the number of elements and constraints the planner has to deal with at solving time.

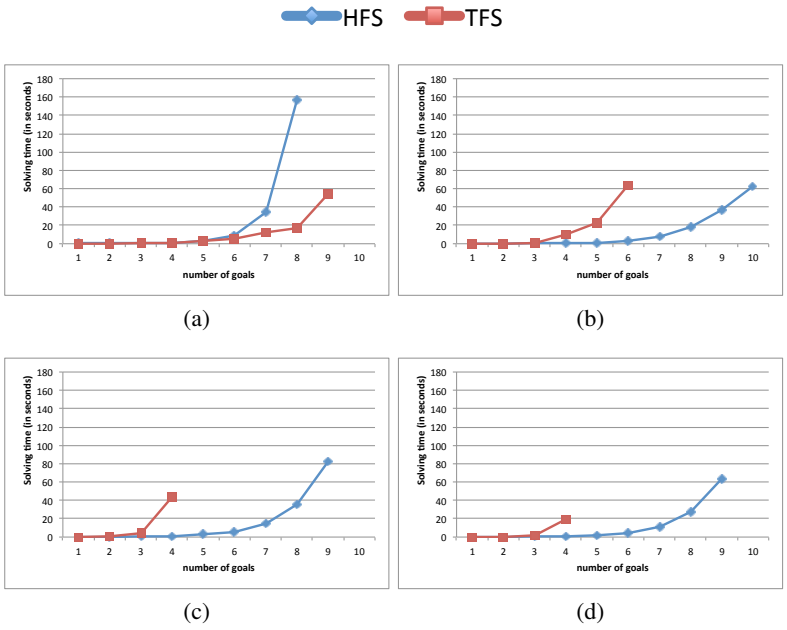


Fig. 5. *HFS* and *TFS* planners performances on: (a) *simple* configuration; (b) *single* configuration; (c) *double* configuration; (d) *full* configuration

The charts in Figure 5 show the solving time trends of the EPSL-based planners (within a timeout of 180 seconds) w.r.t. the growing dimension of the planning problem (i.e. a growing number of goals) and the growing complexity of the module to control (i.e. the number of available cross transfers). The results show that the *HFS* planner dominates *TFS* planner on the considered planning domains. The introduction of the HFS heuristic in the solving process entails a general improvement of the performances in terms of both solving time and scalability of framework.

Let us consider the subset of problems solved by both *HFS* and *TFS* planners in Figure 5 and make a comparison of generated plans. Figure 6 compares the generated plans by considering the *fluidity* metric previously introduced.

The chart in Figure 6(a) shows the trend of the fluidity of the plans w.r.t. the growing complexity of the addressed problems in terms of number of goals and constraints to manage. Results show that the higher is the complexity of the problems the lower is the amount of fluidity of the generated plans. Thus, as expected, the higher is the number of tokens on timelines the higher is the probability that a temporal deviation on a token causes temporal deviations on tokens of other timelines.

The charts in Figure 6(b) and (c) compare respectively the total fluidity and the average makespan of the generated plans. Results show that *TFS* planner generates plans more robust than *HFS* planner w.r.t. fluidity metric. The total fluidity of the plans generated by *TFS* planner, indeed, is higher than the total fluidity of the plans generated by *HFS* planner. Thus, the introduction of HFS heuristic in the solving process seems to

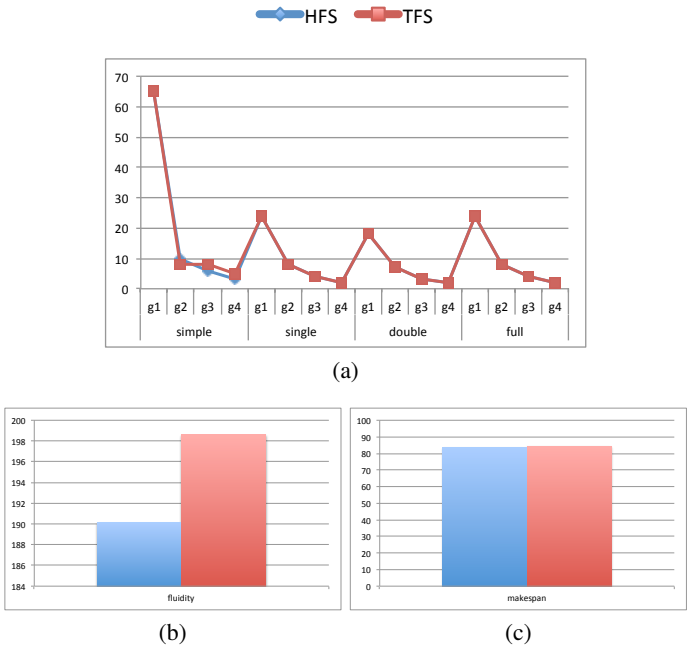
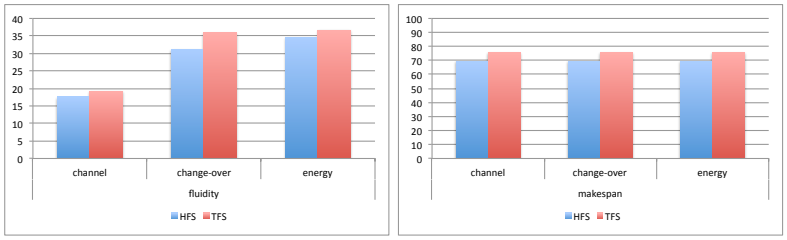


Fig. 6. Comparison of HFS and TFS planners on: (a) plan fluidity trend; (b) total plan fluidity; (c) average plan makespan

lead to a loss of the robustness of the generated plans despite a significant improvement of the solving capabilities as shown in Figure 5.

This can be explained by considering that the *TFS* planner maintains a "complete" view of the plan during the solving process. The planner reasons about the overall plan by taking into account all the timelines of the domain, so that it can organize the tokens on timelines as best as it can. Conversely, the *HFS* planner maintains a "local" view of the plan which is related to single timelines of the domain. Namely, the *HFS* planner builds one timeline at a time. Therefore, when the planner must manage "intermediate" timelines, its choices are partially constrained by the choices made before.

This behavior of HFS heuristic can lead also to a loss of performances in some specific cases such as the *simple* domain in Figure 5(a). The chart shows that, despite the general trend, the *TFS* planner performs better than *HFS* planner in the *simple* domain. Because of its "local" approach, the *HFS* planner is not able to effectively organize tokens on timelines as *TFS* planner does. In particular *TFS* planner is able to efficiently group tokens of *channel* timeline requiring the same configuration of the module, i.e. the same type of token on the *change-over* timeline. Conversely the *HFS* reasons on one timeline at a time, so it is not able to group tokens requiring the same configuration on *channel* timelines. As a consequence the planner must manage much more tokens on the *change-over* timeline increasing the complexity of the problem resolution.

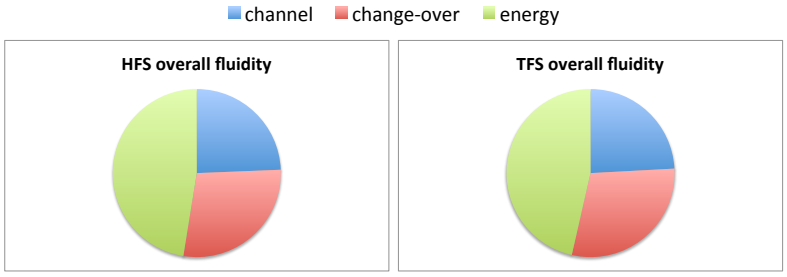


(a) (b)

Fig. 7. Comparison of plans generated for the *simple* domain on: (a) the total fluidity, (b) the average makespan

As a matter of fact the chart of Figure 7(a) shows that the fluidity of the *change-over* timeline in the *TFS* generated plans is quite higher than the fluidity of the same timeline in the *HFS* generated plans.

Finally the introduction and analysis of temporal metrics allow also to extract useful information about planning domains. Let us consider the contribution of single timelines to the total fluidity of the generated plans shown in Figure 8.



(a) (b)

Fig. 8. Contribution of the single timelines to the overall plan fluidity for (a) *HFS* generated plans and (b) *TFS* generated plans

It is possible to see that, regardless the heuristic applied during the solving process, the relationships among domain timelines remain constant. The charts in Figure 8, show that, in general, the *energy* timeline is the one with the highest value of fluidity while the *channel* timeline is the one with the lowest value of fluidity. Thus, the *channel* timeline is the most critical one w.r.t. the robustness of the plan. Namely a temporal deviation on a token of the *channel* timeline is more likely to produce side effects on other timelines of the domain than tokens on other timelines. This sort of information is very interesting since they can be used to classify planning domains and introduce some learning mechanism to adapt the solving process to the specific features of a planning domain.

6 Conclusions and Future works

In this paper we have presented our preliminary work about the introduction of quality metrics to evaluate the robustness of flexible timeline-based plans. In particular, we have adapted some temporal metrics defined in scheduling to timelines. Then we have introduced EPSL, a timeline-based planning framework we use to design and develop P&S applications. Finally we made an experimental evaluation of two EPSL-based planners on a manufacturing case study.

Experimental results have shown how temporal metrics allow to make an assessment of the heuristics applied during the solving process. In particular, temporal metrics let us able to evaluate EPSL-based planners both from the solving performance and the plan quality point of views. An important issue to be addressed in future works is the introduction of a "dynamic" measure of flexibility in order to make a deeper analysis and assessment of the robustness of a plan. In particular we believe that an analysis of this sort is especially needed to provide a valid estimate of the disruptibility metric. Another objective is to develop some parametrized search heuristics heuristcs allowing to exploit temporal features of (partial-)plans during the solving process.

Moreover the experimental results have shown that these metrics charaterize information about the structure and relationships among domain timelines. Thus, another interesting future goal is to introduce some learning mechanism for dynamically adapting heuristics to the specific features of the problem to address.

Acknowledgments. Andrea Orlandini is partially supported by the Italian Ministry for University and Research (MIUR) and CNR under the GECKO Project (Progetto Bandiera "La Fabbrica del Futuro").

References

1. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed.: Intelligent Scheduling. Morgan Kauffmann (1994)
2. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in Interplanetary Space: Theory and Practice. In: Proceedings of the 5th International Conference on AI Planning and Scheduling (AIPS). (2000)
3. Cesta, A., Cortellessa, G., Denis, M., Donati, A., Fratini, S., Oddi, A., Policella, N., Rabenau, E., Schulster, J.: MEXAR2: AI Solves Mission Planner Problems. IEEE Intelligent Systems **22**(4) (2007) 12–19
4. Ceballos, A., Bensalem, S., Cesta, A., de Silva, L., Fratini, S., Ingrand, F., Ocón, J., Orlandini, A., Py, F., Rajan, K., Rasconi, R., van Winnendael, M.: A Goal-Oriented Autonomous Controller for space exploration. In: Proceedings of the 11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA). (2011)
5. Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., Smith, D.: EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In: ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling. (2012)
6. Ghallab, M., Laruelle, H.: Representation and control in ixtet, a temporal planner. In: AIPS. Volume 1994. (1994) 61–67

7. Cesta, A., Fratini, S.: The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In: PlanSIG-08. Proc. of the 27^{th} Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12. (2008)
8. Roberts, M., Howe, A., Ray, I.: Evaluating diversity in classical planning. In: Proceedings of the 24th International Conference on Planning and Scheduling (ICAPS). (2014)
9. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: ICAPS. Volume 4. (2004) 209–218
10. Cesta, A., Oddi, A., Smith, S.F.: Profile-based algorithms to solve multiple capacitated metric scheduling problems. In: AIPS. (1998) 214–223
11. Umbrico, A., Orlandini, A., Cialdea Mayer, M.: Enriching a timeline-based planner with resources and hierarchical reasoning. In: Proceedings of the 14th Conference of the Italian Association for Artificial Intelligence (AIxIA). (2015) To appear.
12. Cesta, A., Orlandini, A., Umbrico, A.: Toward a general purpose software environment for timeline-based planning. In: 20th RCRA International Workshop on" Experimental Evaluation of Algorithms for solving problems with combinatorial explosion. (2013)
13. Cialdea Mayer, M., Orlandini, A., Umbrico, A.: A formal account of planning with flexible timelines. In: The 21st International Symposium on Temporal Representation and Reasoning (TIME), IEEE (2014) 37–46
14. Cimatti, A., Micheli, A., Roveri, M.: Timelines with temporal uncertainty. In: AAAI. (2013)
15. Cialdea Mayer, M., Orlandini, A.: An executable semantics of flexible plans in terms of timed game automata. In: The 22st International Symposium on Temporal Representation and Reasoning (TIME). (2015) To appear.
16. Benton, J., Coles, A.J., Coles, A.: Temporal planning with preferences and time-dependent continuous costs. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS). Volume 77. (2012) 78
17. Smith, D.E.: Choosing objectives in over-subscription planning. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS). Volume 4. (2004) 393
18. Nau, D., Ghallab, M.: Measuring the performance of automated planning systems. Technical report, DTIC Document (2004)
19. Bernardini, S., Smith, D.E.: Towards search control via dependency graphs in europa2 (2010)
20. Borgo, S., Cesta, A., Orlandini, A., Rasconi, R., Suriano, M., Umbrico, A.: Towards a cooperative -based control architecture for a reconfigurable manufacturing plant. In: 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014), IEEE (2014)
21. Carpanzano, E., Cesta, A., Orlandini, A., Rasconi, R., Valente, A.: Intelligent dynamic part routing policies in plug&produce reconfigurable transportation systems. CIRP Annals - Manufacturing Technology **63**(1) (2014) 425 – 428

New Heuristics for Timeline-based Planning

Riccardo De Benedictis and Amedeo Cesta

CNR, Italian National Research Council, ISTC, Rome, Italy
{name.surname}@istc.cnr.it

Abstract. The timeline-based approach to planning represents an effective alternative to classical planning in complex domains where different types of reasoning are required in parallel. The iLoC domain-independent planning system takes inspiration from both Constraint Programming (CP) and Logic Programming (LP). By solving both planning and scheduling problems in a uniform schema, iLoC is particularly suitable for complex domains arising from real world dynamic scenarios. Despite the planner captures elements that are very relevant for applications, its theory is quite challenging from a computational point of view and its performance are rather weak compared with those of state-of-the-art classical planners, particularly on those domains where such planners, typically, excel. In previous works, a resolution algorithm for the iLoC system has been proposed and enhanced with some (static and dynamic) heuristics that help the solving process. In this paper we propose a first improvement of the data structures underlying the proposed heuristics, producing a more informed heuristic and studying its effectiveness as a solving strategy. We perform tests on different benchmark problems from classical planning domains like the Blocks World to more challenging temporally expressive problems like the Temporal Machine Shop and the Cooking Carbonara problems, showing how the iLoC planner compares with respect to other state-of-the-art planners.

1 Introduction

Most of the current timeline-based planners [23], like Europa [20], ASPEN [8], IxTeT [18] and Apsi-Trf [17, 6], are defined as complex software environments suitable for generating planning applications, but quite heavy to foster research work on specific aspects worth being investigated. Such architectures are, typically, inherently quite inefficient and, therefore, rely on a careful engineering phase of the domain, possibly supported by the definition of domain-dependent heuristics. Exception made for some works (e.g., [3]), their search control part has always remained significantly under explored.

Mostly based on the notion of partial order planning [28], timeline-based planners have usually neglected advantages from classical planning triggered from the use of GraphPlan and/or modern heuristic search [4, 5, 19]. Furthermore, timeline-based architectures mostly rely on a clear distinction between a module for temporal reasoning and other modules that perform other forms

of constraint reasoning, while there is not enough exploration of other forms of reasoning.

In order to cope with such pitfalls, in a recent work [14] we presented a new framework, called iLoC, able to solve both planning and scheduling problems in a uniform schema. In addition, we described its resolution algorithm and endowed it with some (static and dynamic) heuristics. The initial heuristic followed the general principle of simplifying the initial problem, solving such simplified problem, and then use the solution for guiding the search of the initial, more complex, problem. At this initial stage, we left only causal relations and removed all the other types of constraints from the problem, resulting in a heuristic which, despite allowed us to greatly improve the performance of the reasoner, ended up being too uninformed.

This paper reintroduces some of the "removed" constraints (in particular the disjunctions) in the heuristic, thus enriching the informativeness and enabling improved performances of the resolution algorithm. In particularly we targeted those domains in which performances were worse. To explain our technique, we first introduce the basic principles underlying the iLoC system, then describe the new heuristic, and show how the system reasons about timelines, then compare it with other planners on different domains.

2 iLoC: An Integrated Logic and Constraint Reasoner

The aim here is to describe to the reader a minimalistic core that should be both sufficiently expressive as well as easily extensible so as to adapt as much as possible to the most variety of user requirements. Specifically, the basic core of the iLoC architecture provides an object oriented virtual environment for the definition of objects and constraints among them. Similarly to most object oriented environments, every object in the iLoC environment is an instance of a specific *type*. iLoC distinguishes among *primitive types* (e.g., bools, ints, reals, strings, etc.) and user defined *complex types* (e.g., robots, trucks, locations, etc.) endowed with their *member variables* (variables associated to a specific object of either primitive or complex type), *constructors* (a special type of subroutine called to create an instance of the complex type) and *methods* (subroutines associated with an object of a complex type). Defining a navigation problem, for example, might require the definition of a *Location* complex type having two numeric member variables x and y representing the coordinates of each *Location* instance. In the following, we will address objects and their member variables using a Java style *dot* notation (e.g., given a *Location* instance l, its x-coordinate will be expressed as $l.x$).

Once objects are defined, iLoC allows the definition of constraints among them. For example, in case a robot r should always be more East of a location l, the iLoC user could assert a constraint such $[\![l.x < r.x]\!]$. iLoC considers constraints as logic propositions and, as such, it allows the possibility for negating them (e.g., $\neg[\![l.x \leq 5]\!]$), for expressing conjunctions (e.g., $[\![l.x \leq 10]\!] \wedge [\![l.x \geq 5]\!]$), disjunctions (e.g., $[\![l.x \leq 5]\!] \vee [\![l.x \geq 10]\!]$) and logic implications (e.g., $[\![l.x \geq$

$10]\!] \rightarrow [\![l.y \geq 10]\!]$). In order for a solution to be valid, such constraints must always be consistent among themselves therefore, whenever an inconsistency is detected (e.g., $[\![l.x \leq 10]\!] \wedge [\![l.x \geq 15]\!]$), the system will return a failure.

In addition, it is possible to impose constraints on existentially quantified variables (e.g., $\exists l \in \mathit{Locations} : l.x \geq 10$) as well as universally quantified variables (e.g., $\forall l \in \mathit{Locations} : l.x \leq 100$). By combining logical quantifier and object oriented features, iLoC allows to manage, in one shot, all the instances of a given complex type.

A rather straightforward method for managing this kind of problems is to translate them into a Satisfiability Modulo Theories (SMT) problem (see, for example, [26]). There are several available SMT solvers having different performances, capabilities as well as licenses. Since iLoC has been written in Java the only available choices are, to the best of our knowledge, the SMTInterpol [9], the MathSAT 5 [10] and the Z3 [15] solvers[1].

Although this basic core allows the definition of quite complex problems (without providing any demonstration, we can state that NP-Complete problems are covered), some of the problems we are interested in are in PSPACE and thus excluded from the possibility of being modelled with this formalism. In order to overcome these limitations, we need something more powerful. Something that, roughly speaking, is able to "decide" the number of involved variables, together with their value. For this purpose, we have chosen to extend the above formalism by allowing many-sorted first-order Horn clauses[2], i.e., clauses with at most one positive literal, called the *head* of the clause, and any number of negative literals, forming the *body* of the clause. For example, we could use a predicate such as $\mathit{FirstQuadrant}$, with a $\mathit{Location}\ l$ argument, within the clause $\mathit{FirstQuadrant}\,(\mathit{Location}\ l) \Leftarrow [\![l.x \geq 0]\!] \wedge [\![l.y \geq 0]\!]$, for describing locations in the first quadrant of a Cartesian coordinate system. Furthermore, we do not allow constraints in the head of a clause but we slightly relax the "positive" literals in the body by allowing constraints to appear in any logical combination (i.e., we could rewrite the above example as $\mathit{FirstQuadrant}\,(\mathit{Location}\ l) \Leftarrow \neg[\![l.x < 0]\!] \wedge \neg[\![l.y < 0]\!]$).

A consequence of what we have seen is that iLoC planning problems can be described by a collection of clauses. There are two types of clauses: *rules* and *requirements*. A rule is of the form $\mathit{Head} \Leftarrow \mathit{Body}$. While the head of rules is limited to predicates, a rule's body consists of a set of calls to predicates, which are called the rule's *sub-goals*, and a set of constraints, the latter, in any logical combination. We consider rules having the same head as disjunctive. Clauses with an empty head are called requirements and can be calls to predicates (either *facts* or *goals*), or constraints, the latter, in any logical combination. Example of requirements are $\mathit{goal} : \mathit{FirstQuadrant}\,(\mathit{Location}\ l)$, $[\![l.x \geq 5]\!]$ and $[\![l.y \geq 5]\!]$,

[1] While SMTInterpol provides a pure Java implementation, MathSAT and Z3 provide Java wrappers to their native API. We have not found other SMT solvers that provide, directly or indirectly, a Java API.

[2] This means, in general, sacrificing decidability.

through which we are asking the planner to find a location l, among those which are in the first quadrant, having both coordinates greater than or equal to 5.

It is worth highlighting how the object oriented architecture binds with the discussion above. Intuitively, each variable that appears as the argument of a predicate inside rules is considered as universally quantified. Conversely, each variable that appears as the argument of a predicate inside a requirement is considered as existentially quantified. The object oriented architecture, combined with the many-sorted logic, allows to consider only the instances of a specific complex type, rather than all the defined objects, as the allowed values for the object variables.

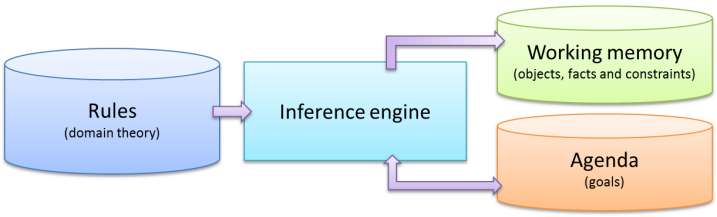


Fig. 1. A high-level view of the iLoC reasoning engine.

From an operational point of view, iLoC uses an adaptation of the *resolution principle* [25] for first-order logic, extended for managing constraints in the more general scheme usually known as *constraint logic programming* (CLP) [1]. Starting from the initial set of objects, facts and constraints, as described by the initial requirements, the reasoner maintains an agenda of the current (sub)goals. Incrementally, the system chooses (sub)goals from the agenda and, by exploiting rules, adds facts and constraints into the working memory. Figure 1 shows a general description of the iLoC reasoning engine.

For each goal $P\left(t_1^g, \ldots, t_i^g\right)$, in general, a branch in the search space is created. Resolution, at first, will try to *unify* goals with existing facts, if any, creating a single branch for all the possible unifications. Specifically, given the existing facts $P\left(t_1^1, \ldots, t_i^1\right), \ldots, P\left(t_1^j, \ldots, t_i^j\right)$, having the same predicate of the goal, the formula $[\![t_1^g = t_1^1 \wedge \ldots \wedge t_i^g = t_i^1]\!] \vee \ldots \vee [\![t_1^g = t_1^j \wedge \ldots \wedge t_i^g = t_i^j]\!]$ is added to the current solution. Intuitively, the purpose of unification is to avoid considering goals whose (any) rule has already been applied. In addition, a branch is also created for each of the rules whose head unifies with the chosen goal and, whenever such a branch is chosen by the resolution algorithm, the body of the corresponding rule is added to the current solution possibly generating further goals to be managed. Summarizing, the basic operations for refining a partial solution π toward a final solution are the following:

1. find the (sub)goals of π (i.e., the agenda).
2. select one such (sub)goals.

3. find ways to resolve it.
4. choose a resolver for the (sub)goals.
5. refine π according to that resolver.

The process follows an A* search strategy that aims at minimizing the number of goals in the agenda, proceeding until there are no more goals into the agenda and while all the constraints in the working memory are consistent. Whenever the constraints become inconsistent the system performs a backtracking step.

3 The MinReach Heuristic

Since all the goals must be solved sooner or later, there is almost no difference among *which* goal is solved first. Selecting the "right" goal, however, impacts heavily with the efficiency of the resolution algorithm. In order to overcome this obstacle we can take advantage of some heuristics. In our previous work [14] we have presented a data structure, called *static causal graph*, and we showed how information could be extracted from it to guide the search process.

The static causal graph has a node for each of the predicates that appear in our rules and, for every rule, an edge from the head of the rule to each of the predicates that appear in the body of the same rule. The cost for solving a goal, as suggested by our heuristic, is equal to the number of reachable nodes from the node relative to the predicate associated to the goal. The rough idea behind this strategy is to evaluate goals by considering a kind of worst case scenario where none of the formulas unify. Another way of looking at it is to consider, for each predicate, a new problem having rules without any constraints and a sole goal of the same predicate. We called such a strategy AllReachable (AR) goal selection heuristic.

$A(z) \Rightarrow B(y) \wedge [\![z < y]\!]$
$B(x) \Rightarrow [\![x < 10]\!]$
$C() \Rightarrow D()$
$E() \Rightarrow F() \wedge G()$
$G() \Rightarrow F() \wedge H()$
$G() \Rightarrow I()$

$A(w) \wedge [\![w = 5]\!]$
$B(v) \wedge [\![v = 7]\!]$

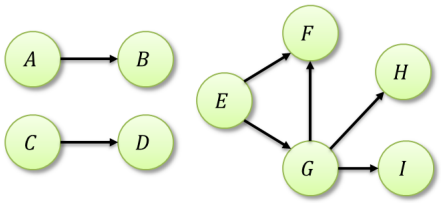


Fig. 2. A set of rules and requirements, along with the associated static causal graph.

Figure 2 shows a set of rules and the static causal graph resulting from them. As an example, the cost for solving an $A(w)$ goal, according to AR, is 1 (since

the sole node B is reachable from node A) while the cost for solving an $E\,(t)$ goal is 4 (since all the nodes F, G, H and I are reachable from node E). Such an heuristic is completely agnostic of disjunctions, putting at the same level all the predicates that appear in the body, whether they were in a disjunction or not, resulting in a too uninformed heuristic and, consequently, in bad performance of the search strategy. Indeed, solving a $G\,()$ goal would be evaluated as having cost 3, regardless of the two (disjunctive) rules having $G\,()$ as head.

A slight improvement to our heuristic is constituted by the addition of disjunctions into the static causal graph by means of two special nodes representing conjunctions (AND nodes) and disjunctions (OR nodes). Figure 3 shows the improved static causal graph generated from the example in Figure 2. The cost for solving a goal is now evaluated as the *minimum* number of reachable nodes starting from the node associated to the goal predicate. The general idea here is the following: whenever the resolution algorithm finds a disjunction, the application of the rule that would lead to the minimum number of formulas should be chosen. We call such a strategy MinReach (MR). As an example, the cost for solving a $G\,()$ goal is now reduced from 3 to 1 since all the nodes F, H and I are reachable from node E, yet introducing a sole formula $I\,()$ (second rule associated to predicate G) is probably preferable than introducing both formulas $F\,()$ and $H\,()$ (first rule associated to predicate G), and far more preferable than introducing all the three formulas $F\,()$, $H\,()$ and $I\,()$ as expected by heuristic AR.

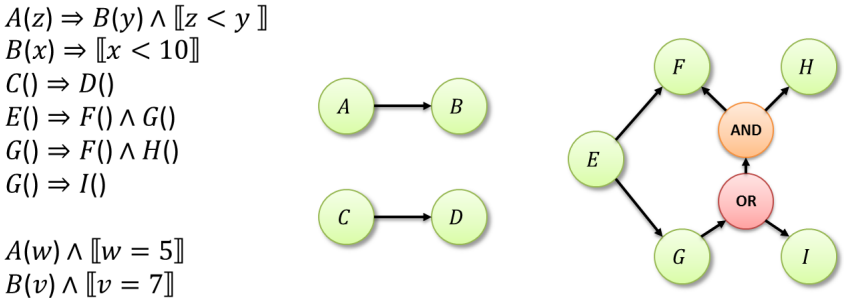


Fig. 3. An AND/OR static causal graph.

One might argue that by introducing disjunctions into the static causal graph we increase the complexity of the evaluation from polynomial to exponential. However, just as the AR heuristic, this graph and, consequently, the costs for each of their nodes, solely depend from the rules, therefore, our heuristic is independent from the requirements and thus can be built once and for ever at the beginning of the solving process, allowing constant-time cost retrieval. Nevertheless the problem can easily be encoded into a MIN-ONE SAT problem (i.e., given a propositional formula, if it is satisfiable, find the variable assignment

that contains the minimal number of positive literals) and let a SAT-solver (e.g., Sat4j [21]) solve it for us. The encoding is trivial:

- a boolean variable is associated to each predicate and to each AND node;
- for each arc $\langle s, t \rangle$, going from source node with boolean variable s to target node with boolean variable t, a clause $(\neg s, t)$ is added;
- for each arc $\langle s, OR \rangle$, going from source node with boolean variable s to an OR target node, we consider the variables $b_1, \ldots, b_n$ associated to all the n nodes directly reachable from the OR node and a clause $(\neg s, b_0, \ldots, b_n)$ is added.

Each predicate can now be evaluated as follows: we assume a unit clause containing the variable associated to the predicate we want to evaluate, solve the resulting MIN-ONE SAT problem, count the number of positive literals associated to predicates and subtract 1, since we don't count the starting node. As an example, the resulting MIN-ONE SAT problem associated to predicate G of Figure 3 is the following (we use lowercase names for the associated boolean variables):

$$(g)\,(\neg a, b)\,(\neg c, d)\,(\neg e, f)\,(\neg e, g)$$
$$(\neg g, and, i)\,(\neg and, f)\,(\neg and, h)$$

resulting in the sole g and i positive literals and, consequently, in an estimated cost of 1.

Similar to what we did in [14] for the AR heuristic, we exploit the MR heuristic both for goal selection and for node selection. Also, we refine the MR heuristic with the less merges dynamic heuristic (see that paper for further details).

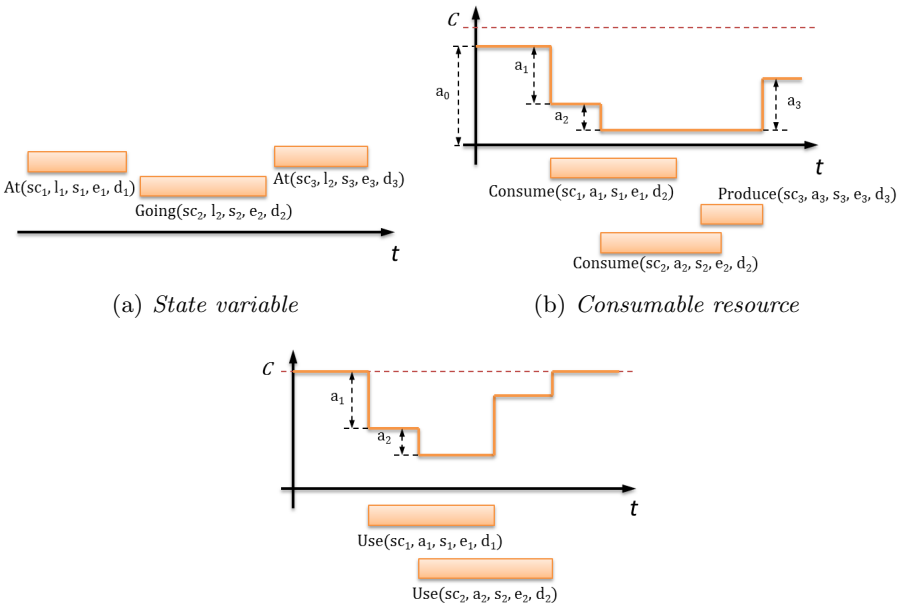
4 Timeline-based Planning and iLoC

The search space of a timeline-based planner has typically *partially specified plans* as nodes and *plan refinement operations* as arcs. Plan refinement operations are intended to further complete a partial solution, i.e., to achieve an open goal or to remove some possible inconsistency. Intuitively, these refinement operations avoid adding to the partial plan any constraint that is not strictly needed for addressing the refinement purpose (this is called the *least commitment principle*). The solving procedure starts from an initial node corresponding to an empty solution and the search aims at a final node containing a solution that correctly achieves the required goals.

A possible approach to the resolution of timeline-based planning problems is to provide the predicates described in the previous sections with numerical arguments in order to represent their starting times, their ending times and their durations. Also, it will be required to define some specific complex types, whose instances will be called *timelines*, in order to add further "implicit" constraints

among the formulas defined "over" their instances. This will also result in a slight adaptation of the resolution procedure in order to check the consistency for every object in the current partial solution so as to make explicit the just mentioned implicit constraints.

What does it mean to define a formula "over" a timeline? We simply add a parameter having the same type as the timeline to the predicates and call such a parameter *scope*. It is worth noting that most timeline-based planners like EUROPA, or APSI-TRF, indeed, consider timelines as a sort of "containers" for formulas. In our approach, since the core reasoning element are the atomic formulas, and consistently with a classical logical approach, we choose to incorporate the timelines "inside" the formulas. In other words, the type of our scope variables will be a "distinguisher" for triggering further reasoning. Furthermore the resulting scope variables are, to all effects, *variables* and, therefore, could be subject to constraints.



(a) *State variable* (b) *Consumable resource*

(c) *Reusable resource*

Fig. 4. Different kinds of timelines with formulas and resource profiles.

In the following we describe the minimal set of the complex types commonly used in timeline-based planning.

State variables. They are used to describe the "state" of a dynamical system as, for example, the position of a specific object at a given time or a simple

manufacturing tool that might be operating or not. The semantics of a state variable (and thus the implicit constraints we need to make explicit) is simply that, for each time instant $t \in \mathbb{T}$, the timeline can assume only one value. Figure 4(a) represents an example of state variable with three atomic formulas (parameter types are omitted for sake of space). The example shows a robot r_0, a state variable of type *Robot*, which might be *At* a given location or might be *Going* to another location. We thus have the two predicates $At\,(sc, l, s, e, d)$ and $Going\,(sc, l, s, e, d)$ each having a parameter sc of type *Robot* describing the scope of the formulas and parameters l, s, e and d respectively for the location, the start, the end and the duration. The planner will take care of adding the proper constraints for avoiding the temporal overlapping of the incompatible states (i.e., all the formulas which have the same scope and do not unify) or for "moving" the states on other instances of type *Robot* (i.e., choosing another value, for example r_1, for the scope of the formula).

Resources. They are entities characterized by a *resource level* $\mathcal{L} : \mathbb{T} \to \mathbb{R}$, representing the amount of available resource at any given time, and by a *resource capacity* $\mathcal{C} \in \mathbb{R}$, representing the physical limit of the available resource. We can identify several types of resources depending on how the resource level can be increased or decreased in time. A *consumable resource* is a resource whose level is increased or decreased by some activities in the system. An example of consumable resource is a reservoir which is produced when a plan activity "fills" it (i.e., a tank refueling task) as well as consumed if a plan activity "empties" it (i.e., driving a car uses gas). Consumable resources have two predefined rules, each having an empty body, and a predicate $Produce\,(sc, id, a, s, e, d)$ ($Consume\,(sc, id, a, s, e, d)$) as head, so as to represent a resource production (consumption) on the consumable resource sc of amount a from time s to time e with duration d (we use an id parameter to prevent unification among these formulas). In addition, the consumable resource complex type has four member variables representing the initial and the final amount of the resource, the min and the max value for the resource level. Quite popular in the scheduling literature, *reusable resources* are similar to consumable resources where productions and consumptions go in tandem at the start and at the end of the activities. Reusable resources can be used for modelling, for example, the number of programmers employed on a given project for a given time interval. Reusable resources have one predefined rule having an empty body and a predicate $Use\,(sc, id, a, s, e, d)$ as head so as to represent an instantaneous production of resource sc of amount a at time s and an instantaneous consumption of the same resource sc of the same amount a at time e. In addition, the reusable resource type has a member variable for representing the capacity of the resource. Figures 4(b) and 4(c) represent, respectively, an example of consumable resource and an example of reusable resource with some associated formulas.

By introducing these complex types, we require the reasoner to add further constraints so as to avoid object inconsistencies (e.g., different states overlapping for some state variable; resource levels $\mathcal{L}$ exceeding resource capacity $\mathcal{C}$ or going lower than min, etc.). We chose to refine our resolution process by introducing a

step for detecting such inconsistencies and for adding required constraints which would remove them. The resulting basic operations for refining a partial solution π toward a final solution are thus the following:

1. find the (sub)goals of π.
2. select one such (sub)goals.
3. find ways to resolve it.
4. choose a resolver for the (sub)goals.
5. refine π according to that resolver.
6. check for any object inconsistency and remove it.

Similar to [7], we use a lazy approach for detecting inconsistencies. Namely, we let the underlying SMT solver to extract a solution given the current constraints and, in case some inconsistency is detected we add further constraints so as to remove the inconsistency. A simple example should clarify the idea. Let us suppose in a given partial solution there are two formulas describing a state variable sv_k having two overlapping states s_i and s_j, we solve the inconsistency by adding the constraint $[\![s_i.start \geq s_j.end]\!] \vee [\![s_j.start \geq s_i.end]\!] \vee [\![s_i.scope \neq s_j.scope]\!]$ preventing further overlapping of these states on the same state variable. The core idea for solving resource inconsistencies follows a very similar schema.

5 Preliminary Results

To assess the value of our heuristic, we have endowed iLoC with the proposed MinReach (MR) heuristic and tried to compare the resulting system with different planners on different benchmarking problems. Specifically, we have selected four planners that are interesting for their features and compared them with iLoC: iLoC(AR) is the previous version of iLoC exploiting the simpler All-Reachable (AR) heuristic, VHPOP [27] shares with our planner the partial ordering approach, OPTIC [2] and COLIN (see [11]) are both based on a classic FF-style forward chaining search [19]. All the test have been executed with default configurations for every planner.

We start the comparison by solving the Blocks World domain, a workhorse for the planning community. As known, in this domain a set of cubes (blocks) are initially placed on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved. We used the 4-operator version of the classic Blocks World domain, as found on the IPC-2011 website, as a starting point. Specifically, for each block, we defined a state variable for representing what is on top of the block (i.e., either another block or the value "Clear") and a state variable for representing if the block is on the table or not. An additional state variable has been defined for modeling the robotic arm modeling values that represent either the arm holding a block or the value "Empty". Finally, we defined an "Agent" complex type for modeling

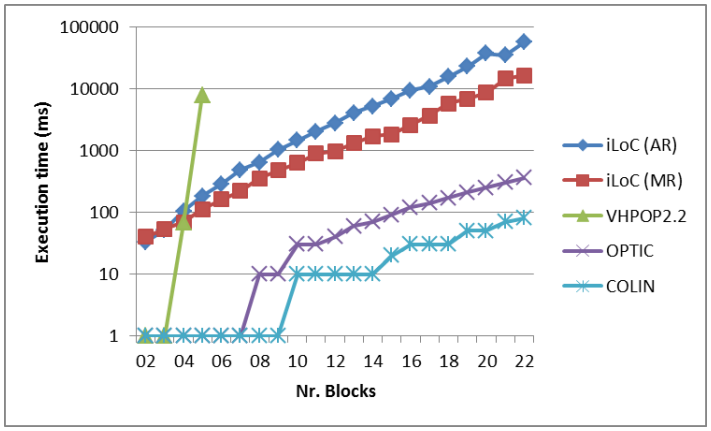


Fig. 5. Blocks world.

the agents' actions. Rules have been defined so as to have an atomic formula for each effect of the PDDL actions as head and an atomic formula for the actions as body, aside from rules having an atomic formula for each PDDL action as head and an atomic formula for their preconditions and effects as body. Temporal constraints have been conveniently added for guaranteeing that preconditions precede actions and effects follow actions.

As shown in Figure 5, despite the introduction of our heuristic planners endowed with "classical heuristics" still perform significantly better than our approach, nevertheless we were able to boost the system performance appreciably, allowing us to find solutions up to, approximately, one third of the time it was required before.

We have also checked our system with two other problems, namely the Temporal Machine Shop [13] and the Cooking Carbonara domain [22]. Both these problems are *temporally expressive* (see [12]) since they require concurrency for being solved.

The first problem is the only temporally expressive problem of the International Planning Competition (IPC) and, within the same competition, it is solved by the sole ITSAT planner (see [24]). The problem models a baking ceramic domain in which ceramics can be baked while a kiln is firing. Different ceramic types require a different baking time. While a kiln can fire for at most 20 minutes at a time (and then it must be made ready again), baking a ceramic takes, in general, less time, therefore we can save costs by baking them altogether. Additionally, similar to [24], we have slightly complicated the domain by considering the possibility for ceramics to be assembled, so as to produce different structures which should be baked again to obtain the final product. Specifically, for each kiln we defined a state variable for distinguishing either the kiln is "Ready" or "on Fire". In addition, each kiln has associated a reusable resource for representing its capacity. For each ceramic piece we defined a state

variable for representing either the piece is "Baking" (with an additional parameter for representing the kiln in which is baking), or the piece is "Baked", or the piece is "Treating", or the piece is "Treated". Similarly, for each ceramic structure we defined a state variable for representing either the structure is "Assembling", or the structure is "Assembled", or the structure is "Baking" (with an additional parameter for representing the kiln in which it is baking), or the structure is "Baked". Rules force these values to appear in time, in each state variable, in the intuitive manner (i.e., in the order in which these values have just been introduced). The interesting aspect, however, is that ceramic structures can bake concurrently with ceramic pieces both *while* (hence the temporal expressiveness) the kiln is firing.

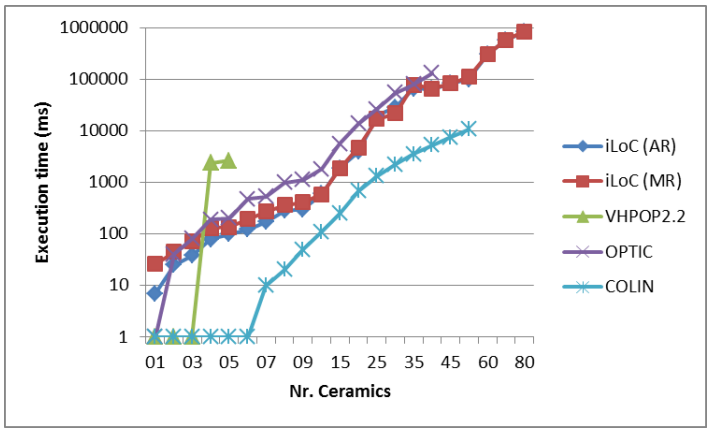


Fig. 6. Temporal machine shop.

The Cooking Carbonara domain represents another temporally expressive problem in which the aim is the preparation of a meal, as well as its consumption by respecting constraints of warmth. Problems cooking-carbonara-n allow to plan the preparation of n dishes of pasta. The concurrency of actions is required to obtain the goal because it is necessary that the electrical plates work in a way that water and oil are hot enough to cook pasta and bacon cubes. It is also necessary to perform this baking in parallel to serve a dish that is still hot during its consumption. Specifically, for each plate we defined a reusable resource for representing its (unary) capacity. For each pot we defined a state variable for distinguishing either the pot is "Boiling" (with an additional parameter for representing the plate on which is boiling) or the pot is "Hot". For each pan we defined a state variable for distinguishing either the pan is "Boiling" (with an additional parameter for representing the plate on which is boiling) or the pan is "Hot". Each portion of spaghetti has associated a state variable for distinguishing either the portion is "Cooking" (with an additional parameter for representing the pot in which is cooking) or the portion has been "Cooked".

For each bacon portion we defined a state variable for distinguishing either the bacon is "Cooking" (with an additional parameter for representing the pan in which is cooking) or the bacon has been "Cooked". Each egg has associated a state variable for distinguishing either the egg is "Being beaten" or the egg has been "Beaten". Finally, for each carbonara portion we defined a state variable for distinguishing either the portion is "Cooking" (with an additional parameter for representing the plate on which should be cooked), or the portion has been "Cooked", or someone is "Eating" the portion or the portion has been "Eaten". Again, rules force values to appear in time, in each state variable, in the intuitive manner (i.e., in the order in which these values have just been introduced). Furthermore, carbonara portions should be cooking after spaghetti, bacon and eggs have been correctly prepared, hence requiring spaghetti to be "Cooking" *while* the water in pots is "Hot" as well as bacon to be "Cooking" *while* the oil in pans is "Hot". Finally, cooking carbonara portions, boiling water in pots and oil in pans should be performed *while* plates are available.

Experimental results on these domains (figures 6 and 7) show that the heuristic does neither guarantee a substantial improvement nor the overhead produces a significant worsening (performance remains almost unchanged). Specifically, in the first problem ILOC performs almost inline with those of state-of-the-art planners. Even though COLIN performs better than ILOC, it is not able to solve problems with more than 50 ceramics since it runs out of memory (we used the default configuration for the planner). In the Cooking Carbonara domain, however, by removing the maximum duration for plate firing, the problem is reduced to a basic scheduling problem hence allowing ILOC to outperform state-of-the-art solvers. This behavior can be explained by observing that these problems are biased toward a temporal kind of reasoning rather than a causal kind of, therefore they find minimum benefit from the improvements introduced in the new heuristic which is mostly oriented toward causal aspects.

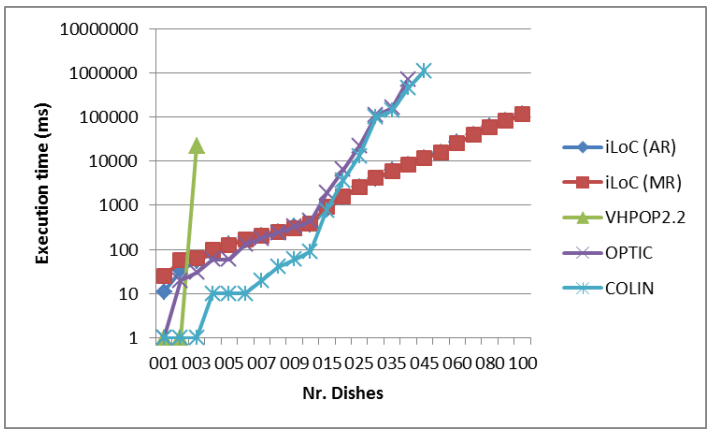


Fig. 7. Cooking carbonara.

A separate discussion it is worth doing concerns the expressiveness of iLoC. All the competing planners use the PDDL2.1 language (see [16]) for modeling their planning problems and, in general, it is quite cumbersome to impose temporal constraints among plain PDDL actions. In the Cooking Carbonara domain, for example, it is important that the cooking happens before the eating but eating should not start too late to avoid that food becomes cold. In [22] a PDDL extension is proposed to overcome this issue and to model properly the domain, however, none of the available planners supports this extension and thus they have been evaluated in a simplified domain in which the warmth constraint decays and dishes can be served anytime after they have been cooked. It is worth noting how this constraint is naturally captured in the iLoC modelling language by creating a rule having as head an action and as body a second action in conjunction with a constraint among the temporal parameters of the two actions.

6 Conclusions

This paper has introduced a new general heuristic for the iLoC planner that improves the planner performance with respect to those of a previous work. The initial heuristic was the result of a too strong simplification and therefore was probably too uninformed. With the present work we started in the direction of reintroducing parts previously neglected. In particular by introducing disjunctions we produced a heuristic that allowed us to improve the performance of the resolution algorithm, especially on those domains in which the performance were weaker.

The iLoC planner already had comparable (or even better) performance of other planners in those domains in which temporal reasoning constitutes the main reasoning requirements (i.e., temporally expressive domains). For this reason we focused on those domains in which the temporal aspects were negligible compared to the causal ones. The current results are still not competitive with respect to those of other planners, nevertheless we succeeded in improving performance on the class of problems not very suited for the timeline-based approach.

We are pursuing a domain-independent planner able to solve efficiently a wider spectrum of planning problems, therefore, work is still needed at heuristic level to reduce the differences with respect to classical approaches.

Acknowledgments. Authors work is partially funded by the Ambient Assisted Living Joint Program under the SpONSOR project (AAL-2013-6-118).

References

1. Apt, K.R., Wallace, M.G.: Constraint Logic Programming Using ECL^{i}PS^e. Cambridge University Press, New York, NY, USA (2007)

2. Benton, J., Coles, A., Coles, A.: Temporal Planning with Preferences and Time-Dependent Continuous Costs. In: Twenty-Second International Conference on Automated Planning and Scheduling (2012)
3. Bernardini, S., Smith, D.: Developing Domain-Independent Search Control for Europa2. In: Proceedings of the Workshop on Heuristics for Domain-independent Planning at ICAPS-07 (2007)
4. Blum, A., Furst, M.L.: Fast Planning Through Planning Graph Analysis. In: IJCAI. pp. 1636–1642. Morgan Kaufmann (1995)
5. Bonet, B., Geffner, H.: Planning as Heuristic Search. Artificial Intelligence 129(12), 5–33 (2001)
6. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: Developing an End-to-End Planning Application from a Timeline Representation Framework. In: IAAI-09. Proceedings of the 21^{st} Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA (2009)
7. Cesta, A., Oddi, A., Smith, S.F.: A Constraint-based Method for Project Scheduling with Time Windows. Journal of Heuristics 8(1), 109–136 (2002)
8. Chien, S., Tran, D., Rabideau, G., Schaffer, S., Mandl, D., Frye, S.: Timeline-Based Space Operations Scheduling with External Constraints. In: ICAPS-10. Proc. of the 20^{th} Int. Conf. on Automated Planning and Scheduling (2010)
9. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: An Interpolating SMT Solver. In: Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings. pp. 248–254 (2012)
10. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)
11. Coles, A.J., Coles, A.I., Fox, M., Long, D.: COLIN: Planning with Continuous Linear Numeric Change. Journal of Artificial Intelligence Research 44, 1–96 (May 2012)
12. Cushing, W., Kambhampati, S., Mausam, Weld, D.S.: When is Temporal Planning *Really* Temporal? In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 1852–1859. IJCAI'07, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007)
13. Cushing, W., Weld, D.S., Kambhampati, S., Mausam, Talamadupula, K.: Evaluating temporal planning domains. In: Boddy, M.S., Fox, M., Thibaux, S. (eds.) Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007. pp. 105–112. AAAI (2007)
14. De Benedictis, R., Cesta, A.: Integrating Logic and Constraint Reasoning in a Timeline-based Planner. In: AI*IA 2015 - XIVth International Conference of the Italian Association for Artificial Intelligence (2015)
15. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
16. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
17. Fratini, S., Pecora, F., Cesta, A.: Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. Archives of Control Sciences 18(2), 231–271 (2008)

18. Ghallab, M., Laruelle, H.: Representation and Control in IxTeT, a Temporal Planner. In: AIPS-94. Proceedings of the 2nd Int. Conf. on AI Planning and Scheduling. pp. 61–67 (1994)
19. Hoffmann, J.: FF: The Fast-Forward Planning System. AI Magazine 22(3), 57–62 (2001)
20. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in Interplanetary Space: Theory and Practice. In: AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling (2000)
21. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. JSAT 7(2-3), 59–6 (2010)
22. Maris, F., Régnier, P.: TLP-GP: Un planificateur pour la rsolution de problmes temporellement expressifs. Revue d'Intelligence Artificielle 24(4), 445–464 (2010)
23. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In: Zweben, M. and Fox, M.S. (ed.) Intelligent Scheduling. Morgan Kauffmann (1994)
24. Rankooh, M.F., Mahjoob, A., Ghassem-Sani, G.: Using Satisfiability for Non-optimal Temporal Planning. In: Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings. pp. 176–188 (2012)
25. Robinson, J.A.: A Machine-Oriented Logic Based on the Resolution Principle. Journal of the Association for Computing Machinery 12(1), 23–41 (1965)
26. Sebastiani, R.: Lazy Satisability Modulo Theories. JSAT 3, 141–224 (2007)
27. Simmons, R.G., Younes, H.L.S.: VHPOP: Versatile Heuristic Partial Order Planner. CoRR (2011)
28. Weld, D.S.: An Introduction to Least Commitment Planning. AI Magazine 15(4), 27–61 (1994)

On the Use of Landmarks in LPG

Francesco Benzi, Alfonso E. Gerevini, Alessandro Saetti, and Ivan Serina
University of Brescia, Italy
{f.benzi,alfonso.gerevini,alessandro.saetti,ivan.serina}@unibs.it

Abstract. Domain-Independent planning is notoriously a very hard search problem. In the literature, several techniques for search control have been proposed in the context of various planning formalisms. In particular, Landmark techniques have been widely used in the planning community in order to guide the search process or to define heuristic functions. A Landmark can be defined as a logical expression, consisting of facts or actions, that certainly becomes true in any solution plan for that problem. In this work, we propose the use of Landmarks for the LPG planner, considering different design choices and analysing empirically its impact on the performance of the planner. Preliminary results show that these techniques can effectively improve the performance of LPG, obtaining results comparable with the state-of-the-art planner LAMA.

Introduction

In the last two decades a number of different techniques have been proposed in the planning community in order to find a good quality solution plan for a given planning problem using a reasonable amount of CPU time. In particular, *planning through local search and action graphs* [13, 10, 12, 6, 8] has shown extremely good performances in the context of fully-automated domain-independent planning. This approach is implemented in the well-known LPG planner, which was awarded at two planning competitions [18, 14] and has been widely used by the planning community, both as a reference planner in many experimental studies comparing planner performance and as a module incorporated into other reasoning systems or applications, e.g. [5, 16, 17, 19, 20, 24, 26]. Landmark techniques have been widely used in the planning community in order to guide the search process or to define heuristic functions [25]. A *landmark* is a fact (a condition or property of the world state) or an action that certainly becomes true in any solution plan for the input planning problem. Different kinds of landmarks and orders between landmarks have been proposed and studied in literature. Finding all possible landmarks and relative orders for a planning problem is computationally very hard [15]. This has made researchers conceive only approximate solutions [15], such as methods that find only a subset of the landmarks and orders. Once landmarks have been computed, there are different ways of using them. The two main families of methods proposed in the literature consist in the use of landmarks to produce a heuristic function and the use of landmarks to decompose the planning process in sub planning tasks [25]. In this paper, we investigate the use of landmark techniques in the context of planning through local search and action graphs, and we evaluate experimentally the impact of their use on the performance of LPG. After a description of the search process in LPG we give some basic definitions regarding landmarks and their computation. Then

we describe the use of landmarks in LPG, and we present the experimental analysis we conducted with different options of our implementation of landmark techniques in LPG. Finally we give conclusions and mention future work.

Local Search in LPG

Our framework is based on a local search in the context of the "planning through planning graph analysis", an approach introduced by Blum and Furst [1]. The problem of generating a plan is a search problem where the elements of the search space are particular subgraphs of the planning graph representing partial plans. The local search method of LPG for a planning graph $\mathcal{G}$ [1] of a given problem $\mathcal{P}$ is a process that, starting from an initial subgraph $\mathcal{G}'$ of $\mathcal{G}$ (a partial plan for $\mathcal{P}$), transforms $\mathcal{G}'$ into a solution of $\mathcal{P}$ through the iterative application of some graph modifications that greedily improve the quality of the current partial plan. Each modification is either an extension of the subgraph to include a new action node of $\mathcal{G}$, or a reduction of the subgraph to remove an action node (and the relevant edges).

Adding an action node to the subgraph corresponds to adding an action to the partial plan represented by the subgraph (analogously to remove an action node). At any step of the search process the set of actions that can be added or removed is determined by the **constraint violations** that are present in the current subgraph of $\mathcal{G}$. More precisely, the search space is formed by the *action subgraphs* of the planning graph $\mathcal{G}$, where an action subgraph of $\mathcal{G}$ is defined in the following way:

Definition 1. *An* **action subgraph** $\mathcal{A}$ *of a planning graph* $\mathcal{G}$ *is a subgraph of* $\mathcal{G}$ *such that if* a *is an action node of* $\mathcal{G}$ *in* $\mathcal{A}$*, then also the fact nodes of* $\mathcal{G}$ *corresponding to the preconditions and positive effects of* a *are in* $\mathcal{A}$*, together with the edges of* $\mathcal{G}$ *connecting them to* a*.*

A *solution subgraph* (a final state of the search space) is defined in the following way:

Definition 2. *A* **solution subgraph** *of a planning graph* $\mathcal{G}$ *is an action subgraph* $\mathcal{A}_s$ *containing the goal nodes of* $\mathcal{G}$ *and such that*

- *all the goal nodes and fact nodes corresponding to preconditions of actions in* $\mathcal{A}_s$ *are supported;*
- *there is no mutually exclusive relation between action nodes.*

The first version of LPG [10, 11] was based on action graphs where each level may contain an arbitrary number of action nodes, as in the usual definition of planning graphs. The following versions of the system [6, 12, 8, 3, 9, 21] used a restricted class of action graphs, called *linear action graphs* (or extensions of them in order to support temporal and numeric information), combined with some additional data structures supporting a more expressive action and plan representation according to the language features of PDDL 2.1 and 2.2 [4, 2].

Definition 3. *A* **linear action graph** *(LA-graph) of* $\mathcal{G}$ *is an action graph of* $\mathcal{G}$ *in which each level of actions contains at most one action node representing a domain action and any number of "no-ops".*

As shown in [6], having only one action in each level of an LA-graph does not prevent the generation of parallel (partially ordered) plans.

The initial LA-graph contains only two special actions a_{start} and a_{end}, where a_{end} is the last action in any valid plan and its preconditions correspond to the goals of the planning problem under consideration; similarly the initial facts represent the effects of the special action a_{start}, which is the first action in any valid plan. Each search step identifies the neighborhood $N(\mathcal{G})$ (successor states) of the current LA-graph $\mathcal{G}$ (search state), which is a set of LA-graphs obtained from $\mathcal{G}$ by adding an action node to $\mathcal{A}$ or removing an action node from $\mathcal{A}$ in an attempt to repair the *earliest* flawed level of $\mathcal{G}$.[1]

The elements in $N(\mathcal{G})$ are evaluated using a *heuristic evaluation function* E [6, 8] consisting of three weighed terms, estimating their additional *search cost*, *execution cost* and *temporal cost*, i.e., the number of search steps required to repair the new flaws introduced, their contribution to the plan quality and their contribution to the makespan of the represented plan, respectively. An element of $N(\mathcal{G})$ with the lowest combined cost is then selected using a "noise parameter" randomizing the search to escape from local minima [6].

Landmark Techniques

Although single fact (atomic formulae) landmarks are the most studied and used kind of landmarks, also single action landmarks are receiving considerable attention nowadays. More complex kinds of landmarks (i.e. conjunction of facts) are not used because deriving them for a planning problem does not give any advantage, in terms of CPU times: the benefits do not outweigh the efforts [23].

Let us concentrate on single fact landmarks. For a given planning problem, initial and goal facts are, by definition, landmarks for that problem. Our commitment is the computation of *causal landmarks*: facts that, for any solution plan, appear as a precondition of an action in the plan [27]. Goal facts are returned too: we can see them as preconditions of a_{end}.

If we want to use landmarks in a planning problem, we do not only need to know which facts are landmarks, we also need to order the landmarks. This order is fundamental to guide the creation of a solution plan: some facts must become true before others in any solution plan. This is exactly why landmarks are really useful. The needed information is thus a graph of landmarks, called *Landmarks Graph* (LG) or *Landmarks Generation Graph* (LGG), where the nodes are the facts and the (directed) arcs are the orders between facts [15]. Depending on the convention used, arcs can go from goal facts to initial facts or vice versa. Any cycle that is eventually generated must be removed before planning starts [15]. The presence of cycles is related, for example, to the recurrence of the "arm free" fact in a "blocks world" planning domain, since "arm free" switches between TRUE and FALSE many times in any solution plan. The use of the information given by cycles in planning is still a research topic [15].

[1] LPG can use several flaw selection strategies that are described and experimentally evaluated in [7]. The strategy preferring flaws at the earliest level of the graph tends to perform better than the others, and so it is used as the default strategy in LPG. More details and a discussion about this strategy are given in the aforementioned paper.

Several different types of landmark orders are defined in literature [15]:

- *Natural* order: in every solution plan Landmark A appears before Landmark B. So A is ordered before B.

- *Necessary* order: in every solution plan, if B is true at a given step, A is true at the previous step.

- *Greedy Necessary* order: in every solution plan, if B is true for the first time in the plan at a given step, then A is true at the previous step. For every successive step where B is true, we know nothing about A.

The previous 3 orders are mandatory, in the sense that they must be satisfied in every solution plan. Instead the next 2 orders are only "suggested": they may help obtain a better solution plan, but it may be that the only way to compute a solution plan is to violate them.

- *Reasonable* order: landmark B is ordered after landmark A if, from a state in which B is true, to make A true it is necessary to destroy B, but after that B will be needed again to reach the goals. So this means that it is reasonable to make A true before B, and not vice versa.

- *Obedient Reasonable* order: if we decide to satisfy reasonable orders, i.e. treat them as mandatory, new "reasonable" orders may arise. These are called obedient reasonable orders.

Regarding the computational complexity of the problem of finding landmarks and relative orders, it has been proved that both the decision problems "Is the fact L a landmark for the given planning problem?" and "Is there an order (of any kind) O between two landmarks A and B of the given planning problem?" are PSPACE-Complete problems [15]. Since every method commonly used for finding Landmarks and orders has polynomial time complexity, these methods are often incomplete or approximate.

There are two families of methods to compute the Landmarks Graph of a given problem. The first includes the method proposed by Hoffmann [15] and its evolutions (for example LAMA). The second is the method proposed by Zhu & Givan. The method proposed by Hoffmann starts from the goal facts, which are landmarks by definition, and for each of them it looks at the first appearance of that fact in the relaxed planning graph. Then, the intersection of the preconditions of all the actions that support that fact in the relaxed planning graph is computed. The facts resulting from the intersection are landmarks, and these new landmarks are ordered before the previous fact landmark. The method continues until no more Landmarks are generated or the initial facts are reached.

The method proposed by Zhu & Givan [27], instead, starts from the initial facts and goes forward on the relaxed planning graph. The key elements of this method are labels associated to facts and actions and represent the list of facts necessary to reach those facts or actionss. The labels are propagated forward along the planning graph. The rule is: for facts, compute the intersection of the content of all the labels that are applied to the actions reaching that fact, then to the resulting list the fact itself must be added; for actions, compute the union of the content of all the labels that are applied to the facts preconditions of that action. Finally, the content of the labels applied to the goal facts is the list of landmarks for the given planning problem. The advantage of Zhu & Givan's method is that it computes all the (causal) landmarks. Furthermore, if not only

the facts but also the actions are propagated, we can compute the Action Landmarks for the given planning problem. The disadvantage is that this method does not specify how to order the computed landmarks. A naive way to order them is to look at the order of appearance of the facts in the labels. However, many superfluous orders are produced in this way. This excess of landmark ordering can in principle increase computation time, however in practice this increase is very limited.

Landmarks in LPG

We implemented two methods for computing the Landmarks Graph. The first is the method proposed by Hoffmann et al., for which we simply imported in LPG the code written by the authors. The second is the method proposed by Zhu & Givan, which we implemented by ourselves through an extension of the existing LPG code.

In the planning process, we used the Landmarks Graph as described in [22]. They proposed a method where the planning problem is divided into sub planning problems, whose concatenated solution gives a global solution to the original problem. This method works by controlling a base planner. During the planning process, when a landmark is satisfied, it is removed (with its edges) from the Landmarks Graph (obviously the Initial Facts are removed from the Landmark Graph at the first step of the planning process). The set of nodes with no other nodes ordered before them is the frontier of the current planning process and corresponds to the set of facts that the system would currently make supported. At every planning step, the Landmarks in the frontier are selected and given to the base planner as a new set of (sub)goal. Then the subplan computed is added to the end of the current (incomplete) plan. When all the landmarks in the graph have been processed, if a solution plan has not been already generated, the base planner is run with the original goals of the problem and the resulting plan is appended to the subplan computed for the processed landmarks. Differently from the work of Hoffmann et al., we do not give the base planner a disjunction of landmarks at every step; instead, we select a single landmark in the previously computed disjunction. The selection is guided by an heuristic function provided by LPG, which is based on the computation of a relaxed plan [6]. Different approaches have been implemented and tested for the selection of a landmark in the frontier, specifically, (1) selection of a random landmark, (2) selection of the landmark with a maximum heuristic value, (3) with a minimum value, and (4) a variant of the third (minimum value) where we prefer landmarks that do not destroy the portion of the relaxed plan needed to reach another landmark in the disjunction. This is done in order to try to limit the destructive interaction between landmarks. Quite interestingly this variant performs extremely well in the logistics domains. This difference with respect to Hoffmann et al.'s method was motivated by the fact that LPG does not work with disjunctive goals, and even if we modified LPG to accept this kind of goals, in the end the planning process of LPG would select a single fact anyway, or a conjunction of facts taken from the disjunctive goal, to be reached. Similarly to Hoffmann et al., if during a planning step more than a single landmark (the selected one) are reached, all of them are removed from the Landmarks Graph.

As previously said, we have implemented and tested three methods to control the insertion of actions in the action graph. The first one allows the local search of LPG to insert actions only at the end of the current partial plan; the second one also allows to insert actions inside the partial plan, instead of only adding actions at the end of the partial plan. The third one allows the local search to both insert and remove actions at any point of the partial plan. This last variant is the most robust one because it allows the search process also to remove previously inserted actions, but is characterised by a larger search space that negatively influences the planner runtime, as observed in teh experimental results section.

Figure 1 gives the pseudo code of our implementation of the landmarks control loop in LPG. In this implementation, we explicitly impose to LPG the next subgoal (chosen among the landmarks in the LG) to be reached.

```
plan = NULL;

while(TRUE)
{
  update_landmarks_graph(initial_state, plan);

  current_subgoal = select_current_landmark();

  plan = LocalSearch(initial_state, plan, current_subgoal);

  if(all_goals_reached(initial_state, plan)) break;
}

return TRUE;
```

Fig. 1. Pseudo code of the landmarks control implementation.

Every run through the loop is a search step handling the creation of the subplan needed to reach a landmark in the Landmarks Graph. First we update the Landmarks Graph to remove those landmarks that have already been reached. Then we select a landmark in the updated Landmarks Graph as the current subgoal. The control loop terminates when all the goals of the problem have been reached: all landmarks that are goals of the problem have been reached and removed from the Landmarks Graph. In this implementation, we simply run the "LocalSearch" routine of LPG to reach the selected landmark. The initial state for LocalSearch is always the initial state of the planning problem. We do this because every call to LocalSearch uses the precomputed partial plan, beginning from the initial state and reaching the previously selected landmark, as a starting point for the local search.

Experimental Analysis

Tests were performed on an Intel(R) Xeon(R) CPU E5-2620 (with an effective 2.00 GHz rating) with 8 GB of RAM. Our tests have been conducted on a series of problems mainly from IPC competitions. Problem domains tested are "Logistics" (IPC2),

Comparative Results			
Planner/Domain	% Sol.	Time (score)	Quality (score)
LLPG ZhuGivan fs3			
logistics IPC2	100.0 %	83.45 (52.17)	602.683 (50.96)
openstacks IPC5	80.0 %	261.59 (12.15)	105.208 (22.52)
storage IPC5	96.6 %	118.09 (25.07)	231.034 (24.01)
TPP IPC5	83.3 %	124.51 (21.01)	125.960 (22.75)
elevators IPC6	100.0 %	3.03 (27.81)	744.833 (23.24)
transport IPC6	83.3 %	266.73 (18.36)	3140.720 (19.61)
elevators IPC7	100.0 %	44.15 (17.66)	326.050 (17.02)
transport IPC7	90.0 %	625.58 (13.36)	423.611 (15.19)
barman IPC7	0 %	900.88 (1.00)	75.000 (0.39)
nomystery IPC7	0 %	1800.00 (0.00)	-1.0 (0.00)
visitall IPC7	25.0 %	13.24 (3.80)	365.200 (3.33)
Total	75.8 %	430.51 (220.06)	565.296 (202.41)
LLPG ZhuGivan fs2			
logistics IPC2	100.0 %	67.41 (57.77)	520.067 (58.99)
openstacks IPC5	90.0 %	49.31 (18.78)	121.407 (25.82)
storage IPC5	93.3 %	126.02 (22.45)	242.893 (22.27)
TPP IPC5	76.6 %	124.01 (16.80)	116.304 (21.20)
elevators IPC6	100.0 %	14.05 (23.79)	658.800 (27.46)
transport IPC6	93.3 %	308.57 (19.01)	3985.786 (22.68)
elevators IPC7	100.0 %	185.91 (13.25)	308.900 (18.23)
transport IPC7	90.0 %	690.72 (13.57)	411.778 (15.34)
barman IPC7	9.5 %	944.86 (1.79)	276.333 (2.11)
nomystery IPC7	10.0 %	3.09 (1.00)	21.000 (0.91)
visitall IPC7	25.0 %	16.08 (3.43)	299.400 (4.00)
Total	78.1 %	487.76 (199.23)	612.357 (227.92)
LLPG ZhuGivan fs1			
logistics IPC2	100.0 %	68.68 (56.97)	557.150 (55.03)
openstacks IPC5	100.0 %	114.47 (28.64)	159.167 (28.84)
storage IPC5	93.3 %	146.59 (21.80)	244.321 (22.09)
TPP IPC5	76.6 %	158.62 (16.31)	116.304 (21.20)
elevators IPC6	100.0 %	6.60 (26.60)	706.833 (25.40)
transport IPC6	93.3 %	301.80 (19.18)	3985.786 (22.68)
elevators IPC7	100.0 %	89.84 (15.94)	313.000 (18.06)
transport IPC7	80.0 %	605.88 (11.91)	396.500 (13.43)
barman IPC7	9.5 %	1041.76 (2.50)	178.000 (3.00)
nomystery IPC7	40.0 %	600.45 (6.46)	28.500 (6.61)
visitall IPC7	25.0 %	11.65 (3.80)	299.400 (4.00)
Total	80.3 %	447.27 (218.22)	625.469 (229.21)
LLPG ZhuGivan fs0			
logistics IPC2	100.0 %	82.10 (51.66)	625.683 (49.12)
openstacks IPC5	80.0 %	250.82 (12.37)	99.542 (23.00)
storage IPC5	93.3 %	133.75 (23.49)	202.964 (24.45)
TPP IPC5	80.0 %	120.35 (18.63)	119.375 (22.03)
elevators IPC6	100.0 %	3.37 (27.71)	703.333 (24.71)
transport IPC6	100.0 %	175.20 (24.76)	4345.367 (25.26)
elevators IPC7	100.0 %	79.29 (16.77)	378.150 (15.00)
transport IPC7	70.0 %	721.08 (10.78)	422.500 (10.17)
barman IPC7	0 %	1800.00 (0.00)	-1.0 (0.00)
nomystery IPC7	15.0 %	663.99 (1.08)	20.667 (1.95)
visitall IPC7	25.0 %	84.14 (2.18)	737.200 (1.62)
Total	76.5 %	424.08 (219.12)	745.541 (204.16)

Table 1. Comparison between landmark selection rules: average values for Coverage, Time and Quality results. IPC score in brackets.

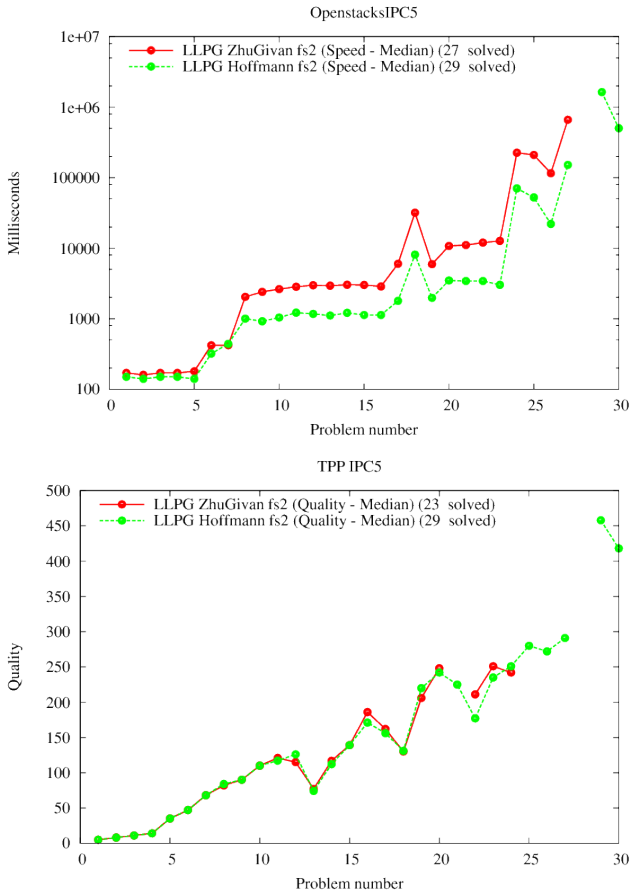


Fig. 2. Speed and Quality results for the Openstacks and TPP (IPC5) domains, respectively. Comparison of using Zhu & Givan's landmarks and Hoffmann's landmarks.

"Transport" (IPC6 and IPC7), "Elevators" (IPC6 and IPC7), "Storage" (IPC5), "Barman" (IPC7), "NoMystery" (IPC7), "Visitall" (IPC7), "Openstacks" (IPC5) and "TPP" (IPC5).

The results are shown in the tables and figures below. For each domain we show runtime (speed) and plan quality results in two separate plots. For each plot, on the x axis we have the different problems from that domain, on the y axis the results. For the "speed" results we used a log10 scale.

In the following comparison tests we will show the results of LPG without landmarks (labelled: LPG) and LPG with landmarks (labelled: LLPG). We will use the label "fs2" to indicate that the selection of a landmark in the disjunction is done using our variant of the "select the one with the minimum heuristic value" rule. The label "test" will indicate that if the landmarks control of LPG fails, the program will retry again and again, until a solution plan is found or the time available expires. Also, ex-

Comparative Results				
Planner/Domain	% Sol.	Time (score)	Quality (score)	LM
LLPG Z&G fs2				
logistics IPC2	100.0 %	67.41 (58.47)	520.1 (59.00)	224.6
openstacks IPC5	90.0 %	49.31 (19.13)	121.4 (25.91)	98.3
storage IPC5	93.3 %	126.02 (26.70)	242.9 (27.00)	0.9
TPP IPC5	76.6 %	124.01 (17.56)	116.3 (21.52)	11.6
elevators IPC6	100.0 %	14.05 (25.92)	658.8 (28.31)	30.7
transport IPC6	93.3 %	308.57 (26.48)	3985.8 (27.00)	0
elevators IPC7	100.0 %	185.91 (16.45)	308.9 (17.88)	68.1
transport IPC7	90.0 %	690.72 (16.70)	411.8 (17.00)	0
barman IPC7	9.5 %	944.86 (3.00)	276.3 (2.58)	14.7
nomystery IPC7	10.0 %	3.09 (1.00)	21.0 (1.00)	14.5
visitall IPC7	25.0 %	16.08 (4.00)	299.4 (3.94)	0
Total	78.1 %	487.76 (223.93)	612.4 (240.02)	463.4
LLPG Hoff. fs2				
logistics IPC2	100.0 %	74.85 (55.21)	520.1 (59.00)	224.6
openstacks IPC5	96.6 %	142.40 (27.98)	147.4 (28.00)	91.3
storage IPC5	93.3 %	134.15 (25.81)	244.3 (26.79)	0.9
TPP IPC5	96.6 %	151.07 (25.93)	157.5 (27.78)	14.1
elevators IPC6	100.0 %	6.87 (28.44)	717.6 (26.39)	24.3
transport IPC6	93.3 %	279.21 (26.59)	3997.6 (26.97)	0
elevators IPC7	100.0 %	121.45 (17.61)	303.4 (18.27)	49.8
transport IPC7	85.0 %	673.96 (15.96)	409.2 (16.00)	0
barman IPC7	0 %	1171.31 (0.51)	73.0 (1.00)	13.6
nomystery IPC7	25.0 %	751.96 (3.43)	25.8 (4.00)	14.8
visitall IPC7	25.0 %	19.91 (3.59)	302.0 (3.90)	26
Total	80.7 %	441.33 (239.59)	624.6 (246.98)	459.4

Table 2. Comparison between Hoffmann's landmarks and Zhu & Givan's landmarks: average values for Coverage, Time, Quality and number of landmarks. IPC score in brackets.

cept differently indicated, the control variant used is the default one: only adding new actions at the end of the partial plan.

In Table 1 we compare LPG with landmarks for all variants of the Landmarks Graph frontier selection methods: selection of a random landmark (labelled: fs0); the selection of the landmark with minimum heuristic value (labelled: fs1); our variant of fs1 designed to avoid destructive interaction between landmarks in the frontier (labelled: fs2); and selection of the landmark with maximum heuristic value (labelled: fs3). The results show that "fs1" is the best selection rule for quality and coverage results. Concerning speed the best rule is "fs3", however the advantage with respect to "fs1" is small. Moreover, "fs2" is particularly effective in Logistics.

In Figure 2 we show speed and quality plots for the comparison between the use of Zhu & Givan's landmarks and the use of Hoffmann's landmarks. As we can see, landmarks computed by Hoffmann's method give better coverage, better time results and better quality results.

Figure 3 plots speed and quality results for the comparison of the landmark selection rules we implemented. The Speed plot shows better coverage and time results for the "fs1" rule, while the Quality plot shows that our variant of "fs1", labelled "fs2", gives, as expected, better quality results in Logistics.

In Table 2 we compare LPG with landmarks in the two variants: using landmarks computed by Hoffmann's method and using landmarks computed using Zhu & Givan's method. The last column contains the average number of Landmarks for every do-

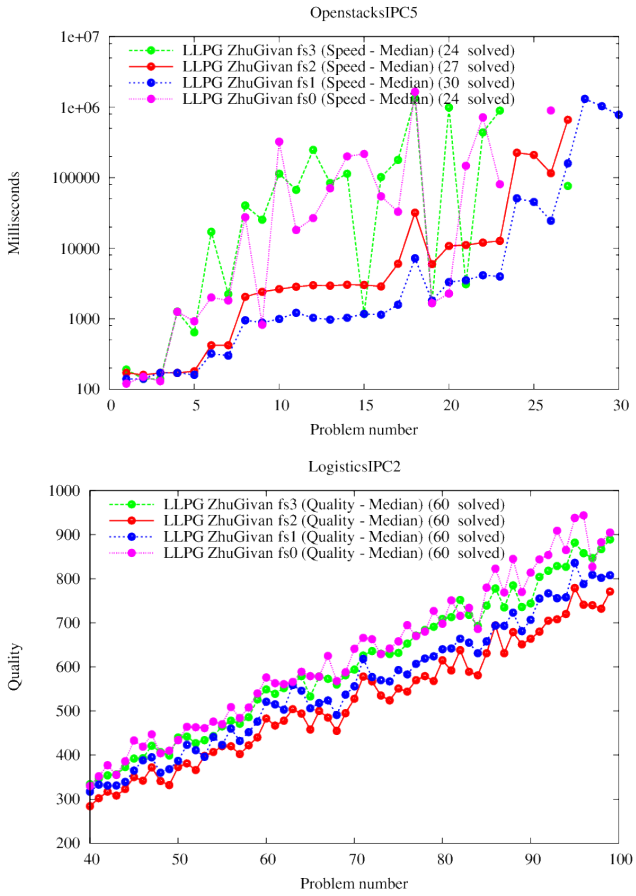


Fig. 3. Speed and Quality results for the Openstacks (IPC5) and Logistics (IPC2) domains, respectively. Comparison of different landmark selection rules.

main (initial and goal Landmarks were not counted); here we can observe that the two approaches produce a similar number of landmarks in the different domains. Unfortunately in Transport and Visitall (Zhu&Givan approach) the system cannot find new landmarks. In general, the use of landmarks computed by Hoffmann's method gives better results; however the difference is usually small.

Table 3 compares LPG with landmarks using three control variants: the one that only adds new actions at the end of the current partial plan (labelled: control 2), the one that can add new actions at any point of the current partial plan (labelled: control 1), and the one that can add as well as remove actions at any point of the partial plan (labelled: control 0). The results show that the best solution is "control 2". However, "control 1" gives best results in IPC5 "TPP" domain, and "control 0" in IPC7 "barman" domain.

Comparative Results			
Planner/Domain	% Sol.	Time (score)	Quality (score)
LLPG ZhuGivan fs2 control 2			
logistics IPC2	100.0 %	67.41 (53.41)	520.067 (56.92)
openstacks IPC5	90.0 %	49.31 (23.91)	121.407 (26.00)
storage IPC5	93.3 %	126.02 (23.00)	242.893 (21.12)
TPP IPC5	76.6 %	124.01 (15.22)	116.304 (19.34)
elevators IPC6	100.0 %	14.05 (27.13)	658.800 (28.11)
transport IPC6	93.3 %	308.57 (22.57)	3985.786 (21.68)
elevators IPC7	100.0 %	185.91 (19.00)	308.900 (19.00)
transport IPC7	90.0 %	690.72 (15.92)	411.778 (17.00)
barman IPC7	9.5 %	944.86 (1.80)	276.333 (2.07)
nomystery IPC7	10.0 %	3.09 (1.00)	21.000 (0.91)
visitall IPC7	25.0 %	16.08 (2.06)	299.400 (4.00)
Total	78.1 %	487.76 (212.33)	612.357 (225.15)
LLPG ZhuGivan fs2 control 1			
logistics IPC2	100.0 %	70.68 (53.40)	508.650 (58.17)
openstacks IPC5	90.0 %	47.75 (23.90)	121.407 (26.00)
storage IPC5	93.3 %	135.79 (22.78)	216.000 (22.98)
TPP IPC5	80.0 %	242.46 (18.30)	113.167 (22.49)
elevators IPC6	86.6 %	295.70 (14.54)	681.231 (20.61)
transport IPC6	80.0 %	385.47 (17.62)	3383.167 (18.39)
elevators IPC7	50.0 %	746.24 (4.01)	480.600 (4.14)
transport IPC7	40.0 %	950.66 (5.07)	686.500 (3.73)
barman IPC7	0 %	1800.00 (0.00)	-1.0 (0.00)
nomystery IPC7	15.0 %	453.40 (1.72)	22.333 (2.00)
visitall IPC7	25.0 %	31.98 (1.77)	428.800 (2.78)
Total	69.1 %	563.73 (192.11)	528.238 (187.22)
LLPG ZhuGivan fs2 control 0			
logistics IPC2	100.0 %	67.01 (57.90)	516.533 (57.30)
openstacks IPC5	16.6 %	.04 (4.00)	25.000 (4.00)
storage IPC5	26.6 %	490.92 (7.00)	17.250 (5.85)
TPP IPC5	56.6 %	359.72 (14.11)	103.647 (11.90)
elevators IPC6	73.3 %	489.85 (14.18)	918.455 (14.49)
transport IPC6	80.0 %	374.24 (17.04)	2811.708 (18.31)
elevators IPC7	20.0 %	666.83 (1.93)	387.250 (1.75)
transport IPC7	45.0 %	841.74 (5.66)	697.556 (3.51)
barman IPC7	14.2 %	779.49 (4.00)	193.250 (4.00)
nomystery IPC7	5.0 %	231.68 (0.00)	20.000 (0.00)
visitall IPC7	20.0 %	361.48 (4.00)	347.500 (3.80)
Total	50.4 %	941.28 (136.59)	417.421 (131.48)

Table 3. Comparison of different control methods: average values for Coverage, Time and Quality results. IPC score in brackets.

The results for domain Elevators (IPC7) in Figure 4 are shown to compare different control methods. As shown in the plots, "control 2" is the best solution, giving best coverage, speed and quality.

In Table 4 we compare LPG with landmarks (LLPG ZhuGivan fs2), LPG without landmarks (LPG) and the state-of-the-art planner LAMA2011 [23]. The experimental tests show that our planner gives the best speed and coverage, while LAMA gives the best quality. However, in assessing the results we must take into consideration that LAMA2011 in some domains (Elevators IPC6 and Transport IPC6) does not solve any problem.

The results for domain Logistics (IPC2) in Figure 5 compare LPG with and without landmarks and LAMA2011. As we can see, only LPG (with and without Landmarks)

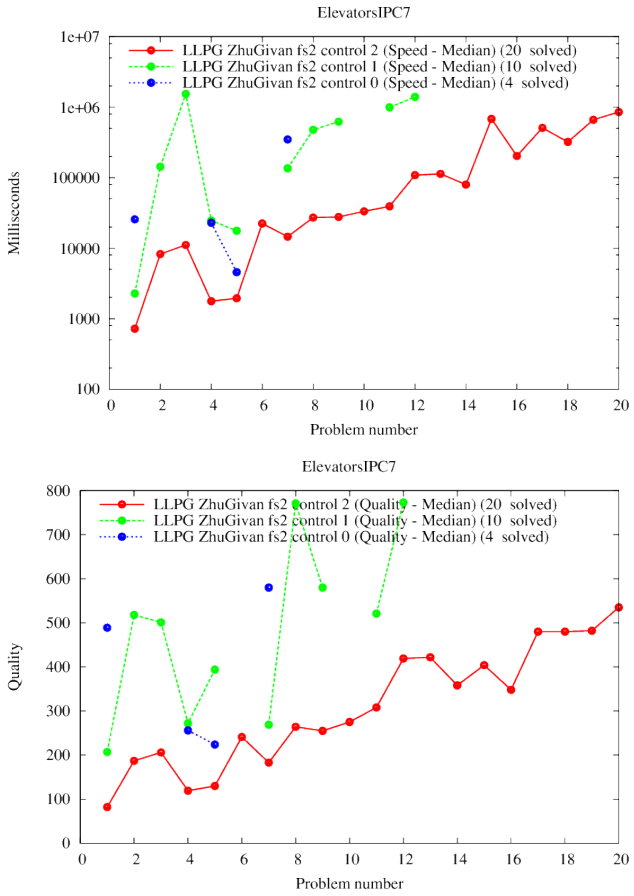


Fig. 4. Speed and Quality results for domain Elevators (IPC7): comparison of different control methods.

solved all the problems. If we consider speed, quality and coverage altogether, the best planner is LPG with landmarks.

Table 5 compares the performances of LLPG versus LPG in terms of delta values of the coverage, the IPC Speed and the IPC Quality scores. The LM column reports the average number of landmarks for every domain (initial and goal landmarks were not counted), while the "# goals" column reports the average number of goals in the different domains. We can see that a high number of landmarks is usually associated with higher performances of LLPG w.r.t. LPG, see for example the `logistics IPC2` domain (where on average we can find 224 landmarks + initial facts + the goal facts) and the `elevators IPC7` domain (where on average we can find 50 landmarks + initial facts + the goal facts). Furthermore, we can also observe performance improvements in the `transport` domains in which the LM value is equal to 0; this is related to the

Comparative Results			
Planner/Domain	% Sol.	Time (score)	Quality (score)
LPG			
logistics IPC2	100.0 %	179.33 (46.27)	666.267 (43.03)
openstacks IPC5	90.0 %	105.73 (17.38)	120.630 (24.98)
storage IPC5	96.6 %	110.74 (26.54)	189.276 (17.91)
TPP IPC5	70.0 %	177.97 (17.44)	123.952 (13.50)
elevators IPC6	90.0 %	442.08 (14.18)	509.296 (23.64)
transport IPC6	53.3 %	448.90 (12.13)	2055.062 (13.19)
elevators IPC7	50.0 %	1005.54 (2.90)	216.300 (7.69)
transport IPC7	5.0 %	1018.84 (0.48)	246.000 (0.48)
barman IPC7	0 %	900.88 (1.00)	74.000 (0.43)
nomystery IPC7	20.0 %	621.51 (1.13)	20.250 (3.00)
visitall IPC7	10.0 %	.52 (1.00)	293.000 (0.64)
Total	63.3 %	780.25 (147.17)	321.550 (155.10)
LLPG			
logistics IPC2	100.0 %	74.85 (56.84)	520.067 (54.81)
openstacks IPC5	96.6 %	142.40 (23.38)	147.379 (26.91)
storage IPC5	93.3 %	134.15 (21.25)	244.321 (13.96)
TPP IPC5	96.6 %	151.07 (18.72)	157.483 (20.46)
elevators IPC6	100.0 %	6.87 (27.54)	717.633 (25.34)
transport IPC6	93.3 %	279.21 (22.36)	3997.607 (21.74)
elevators IPC7	100.0 %	121.45 (16.33)	303.400 (14.36)
transport IPC7	85.0 %	673.96 (11.65)	409.235 (11.12)
barman IPC7	0 %	1171.31 (0.29)	73.000 (0.44)
nomystery IPC7	25.0 %	751.96 (2.42)	25.800 (3.10)
visitall IPC7	25.0 %	19.91 (2.35)	302.000 (4.00)
Total	80.7 %	441.33 (209.72)	624.611 (204.83)
LAMA2011			
logistics IPC2	83.3 %	311.93 (29.53)	441.360 (50.00)
openstacks IPC5	100.0 %	7.85 (22.76)	154.600 (28.99)
storage IPC5	63.3 %	125.06 (8.46)	21.526 (17.92)
TPP IPC5	100.0 %	14.97 (21.69)	116.900 (29.00)
elevators IPC6	0 %	1800.00 (0.00)	-1.0 (0.00)
transport IPC6	0 %	1800.00 (0.00)	-1.0 (0.00)
elevators IPC7	100.0 %	61.10 (17.91)	231.200 (18.29)
transport IPC7	70.0 %	359.93 (12.57)	212.000 (13.00)
barman IPC7	95.2 %	90.85 (20.00)	188.571 (20.00)
nomystery IPC7	60.0 %	144.41 (10.42)	31.417 (9.54)
visitall IPC7	100.0 %	94.02 (18.61)	1483.050 (18.37)
Total	69.1 %	324.14 (205.28)	286.888 (200.61)

Table 4. Percentage of problem solved, CPU time in seconds and Plan Quality (IPC scores in brackets) of LPG, LLPG and Lama 2011.

fact that the LM value does not count the goals as landmarks, although indeed they are landmarks and they are effectively used by LPG with landmarks (LLPG).

Conclusions

In this paper, we have presented some new techniques for planning with landmarks that have been implemented in LPG; the experimental results show significant improvements in terms of both number of problems solved and CPU time. In particular, the use of landmarks for dividing the planning problem into sub planning problems, whose concatenated solution gives a global solution to the original problem, gives extremely interesting results.

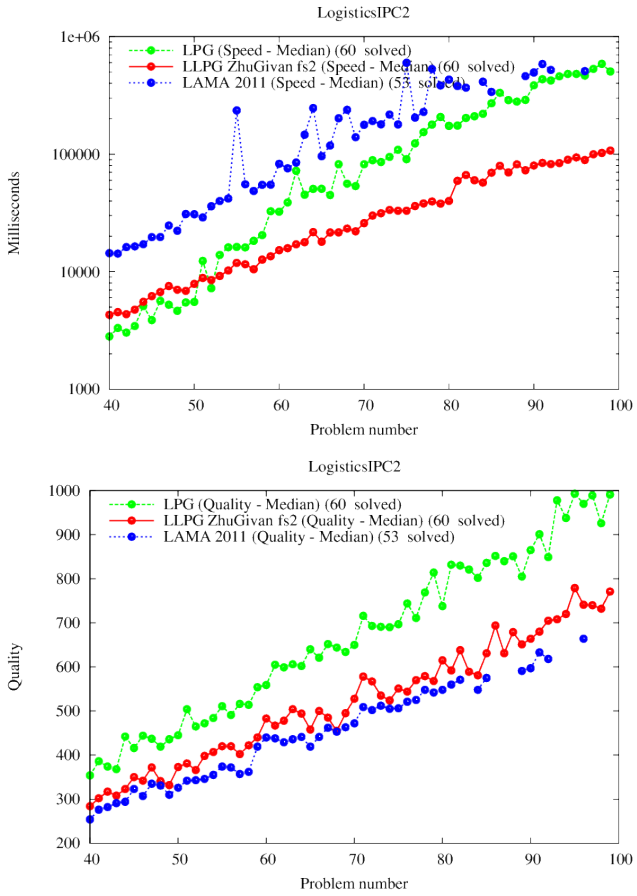


Fig. 5. Speed and Quality results for domain Logistics (IPC2): comparison of different planners.

As future work, we plan to extend LLPG in order to compute landmarks for temporal and metric domains. Moreover, we plan to compute also Action Landmarks and use them to effectively initialize the search process. Exploiting action landmarks seems to be very natural and promising in the context of LPG. Finally, we are developing a new idea about *quasi-landmarks*: facts that appear in almost every solution plan. We expect quasi-landmarks to be useful in domains where the only landmarks that are computed by the existing methods are the initial and goal facts. These domains include, for example, the two domains Transport IPC6 and Transport IPC7 that we used in our experiments.

References

1. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence 90, 281–300 (1997)

	LLPG vs LPG				
Planner/Domain	LM	# goals	△ % Sol.	△ Time score	△ Quality score
logistics IPC2	224.6	69.5	+ 0.0 %	+ 10.6	+ 11.8
openstacks IPC5	91.3	31	+ 6.6 %	+6	+ 1.9
storage IPC5	0.9	7.7	-3.3 %	- 5.3	-4
TPP IPC5	14.1	8.7	+ 26.6 %	+1.3	+7
elevators IPC6	24.3	17	+ 10.0 %	+13.4	+1.7
transport IPC6	0	10.4	+ 40.0 %	+10.2	+8.6
elevators IPC7	49.8	37.6	+ 50.0 %	+13.4	+6.7
transport IPC7	0	18.8	+ 80.0 %	+11.2	+10.6
barman IPC7	13.6	9.3	0 %	-0.7	0
nomystery IPC7	14.8	8.4	+5 %	+1.3	+0.1
visitall IPC7	26	263	+ 15.0 %	+1.3	+3.4
Total	459.4	481.4	+ 17.4 %	+62.6	+49.7

Table 5. LLPG vs LPG in terms on delta values for the coverage, IPC Speed and Quality scores.

2. Edelkamp, S., Hoffmann, J.: PDDL2.2: The language for the classic part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Freiburg, Germany (2004)
3. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: Proceedings of the 16th International Conference on Automated Planning and Scheduling. AAAI Press, Cumbria, UK (2006)
4. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research (JAIR) 20, pp. 61–124 (2003)
5. Gerevini, A., Kuter, U., Nau, D., S., A., S., Waisbrot, N.: Combining domain-independent planning and HTN planning: The Duet planner. In: Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI-08) (2008)
6. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs. Journal of Artificial Intelligence Research (JAIR) 20, 239–290 (2003)
7. Gerevini, A., Saetti, A., Serina, I.: An empirical analysis of some heuristic features for local search in LPG. In: Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04). pp. 171–180. AAAI Press, Menlo Park, CA, USA (2004)
8. Gerevini, A., Saetti, A., Serina, I.: An approach to temporal planning and scheduling in domains with predictable exogenous events. Journal of Artificial Intelligence Research (JAIR) 25, 187–231 (2006)
9. Gerevini, A., Saetti, A., Serina, I.: An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artificial Intelligence 172(8-9), 899–944 (2008)
10. Gerevini, A., Serina, I.: LPG: A planner based on local search for planning graphs with action costs. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02). pp. 281–290. AAAI Press/MIT Press (2002)
11. Gerevini, A., Serina, I.: Planning as propositional CSP: from Walksat to local search for action graphs. CONSTRAINTS 8(4) (October 2003)
12. Gerevini, A., Serina, I., Saetti, A., Spinoni, S.: Local search techniques for temporal planning in LPG. In: Proceedings of the 13th International Conference on Automated Planning & Scheduling (ICAPS03). pp. 62–71. AAAI Press (2003)
13. Gerevini, A., Serina, I.: Fast plan adaptation through planning graphs: Local and systematic search techniques. In: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems. AAAI Press, Breckenridge, CO (2000)
14. Hoffmann, J., Edelkamp, S.: The deterministic part of IPC-4: An overview. Journal of Artificial Intelligence Research (JAIR) 24, 519–579 (2005)

15. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. J. Artif. Int. Res. 22(1), 215–278 (Nov 2004), http://dl.acm.org/citation.cfm?id=1622487.1622495
16. Jimẽnez, S., Fernãndez, F., Borrajo, D.: The PELA architecture: integrating planning and learning to improve execution. In: Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI-08) (2008)
17. Kolobov, A., Mausam, Weld, D., S.: Determinize, solve, and generalize: Classical planning for MDP heuristics. In: ICAPS-09 Workshop on Heuristics for Domain-independent Planning (2009)
18. Long, D., Fox, M.: The 3rd International Planning Competition: Results and analysis. Journal of Artificial Intelligence Research (JAIR) 20, 1–59 (2003)
19. Morales, L., Castillo, L., Fernandez-Olivares, J., Gonzalez-Ferrer, A.: Automatic generation of user adapted learning designs: An AI-planning proposal. In: Proceedings of the Fifth International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH-08) (2008)
20. Nguyen, T., A., Do, M., B., Kambhampati, D., Srivasta, B.: Planning with partial preference models. In: Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09) (2009)
21. Nguyen, T.A., Do, M.B., Gerevini, A., Serina, I., Srivastava, B., Kambhampati, S.: Generating diverse plans to handle unknown and partially known user preferences. Artificial Intelligence 190, 1–31 (2012)
22. Porteous, J., Sebastia, L., Hoffmann, J.: On the extraction, ordering and usage of landmarks in planning. In: European Conference of Planning (ECP'01). pp. 37–48 (2001)
23. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. CoRR abs/1401.3839 (2014), http://arxiv.org/abs/1401.3839
24. Srivastava, B., Nguyen, T.A., Gerevini, A., Kambhampati, S., Do, M.B., Serina, I.: Domain independent approaches for finding diverse plans. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (2007)
25. Vernhes, S., Infantes, G., Vidal, V.: Problem splitting using heuristic search in landmark orderings. In: IJCAI'13. pp. –1–1 (2013)
26. Vrakas, D., Hatzi, O., Bassiliades, N.and Anagnostopoulos, D., Vlahavas, I.: A visual programming tool for designing planning problems for semantic web service composition. In: Visual Languages for Interactive Computing: Definitions and Formalizations (2008)
27. Zhu, L., Givan, R.: Landmark Extraction via Planning Graph Propagation. In *Printed Notes of ICAPS'03 Doctoral Consortium* (June 2003), trento, Italy

Automated Planning for Urban Traffic Control: Strategic Vehicle Routing to Respect Air Quality Limitations

Lukáš Chrpa[1], Daniele Magazzeni[2], Keith McCabe[3], Thomas L. McCluskey[1], and Mauro Vallati[1]

[1] PARK research group, University of Huddersfield, United Kingdom
{n.surname}@hud.ac.uk
[2] Department of Informatics, King's College London, United Kingdom
daniele.magazzeni@kcl.ac.uk
[3] KAM futures, United Kingdom

Abstract. The global growth in urbanisation increases the demand for services including road transport infrastructure, presenting challenges in terms of mobility. These trends are occurring in the context of concerns around environmental issues of poor air quality and transport related carbon dioxide emissions. One out of several ways to help meet these challenges is in the *intelligent routing* of road traffic through congested urban areas. Our goal is to show the feasibility of using automated planning to perform this routing, taking into account a knowledge of vehicle types, vehicle emissions, route maps, air quality zones, etc. Specifically focusing on air quality concerns, in this paper we investigate the problem where the goals are to minimise overall vehicle delay while utilising network capacity fully, and respecting air quality limits. We introduce an automated planning approach for the routing of traffic to address these areas. The approach has been evaluated on micro-simulation models that use real-world data supplied by our industrial partner. Results show the feasibility of using AI planning technology to deliver efficient routes for vehicles that avoid the breaking of air quality limits, and that balance traffic flow through the network.

Keywords: Automated Planning, Urban Traffic Control, Pollution Limits, Micro Simulation

1 Introduction

In the 21st Century the worlds population is expected to increase from 5.9bn in 2013 to 9.6bn by 2100. At the same time the worlds population is expected to become more urbanised, in 2005 there was a 50/50 split between urban and rural population, by 2050 this is predicted to change to a 70/30 split. This huge growth in urbanisation increases the demand for housing, associated utilities and services including transport infrastructure, public transport, vehicle parking and better interchanges between the urban and national transport networks. In turn, this leads to increased pressure on land use and development within or around urban areas contributing to land shortages and urban sprawl. These trends present significant challenges in terms of mobility.

There are expected to be key developments in terms of growth of big data in transport, with increased deployment of roadside and vehicle sensors, and increased automation with the introduction of semi-autonomous and fully autonomous vehicles into the urban environment in the next decade. These trends are occurring in the context of concerns around environmental issues of poor air quality and transport related carbon dioxide emissions.

Current urban transport management systems help to minimise delay within day to day traffic flows using controls such as self-tuning traffic light clusters, but are not designed to take into account varying vehicle types, or regional demands such as avoiding poor air quality, utilising the network fully, or working adequately in the face of unplanned exceptional events. Our goal is to show the feasibility of using automated planning to intelligently route all vehicles collectively through Urban Areas, taking into account a knowledge of vehicle types, vehicle emissions, route maps, air quality zones etc. This contrasts with the current situation where personal navigation devices produce routes for a vehicle in isolation of other travellers, and traffic controls can only react to local flows of traffic. Our approach anticipates future developments in route generation for road users, which is likely to be based on a balance between minimising the average delay of all traffic in an urban region, and observing regional constraints such as area and weather dependent air quality limits.

Whereas there has been a long record of the use of AI techniques in road transportation [14, 11], with some notable exceptions, for example [16, 9, 1], there has been little application of AI Planning and Scheduling techniques to urban traffic control. The novelty of this paper lies in the application of automated planning techniques for strategic urban traffic control, in order to calculate routes for vehicles entering an urban area to satisfy regional-level constraints.

Micro-simulation models of road traffic are models that consider the traffic system at the level of the individual vehicle [13]. We create a micro-simulation model for an urban area, and show how planning can, by varying the vehicle flows through roads, or re-routing the highest polluting vehicles around poor air quality zones, ensure that pollution limits are respected while maximising the utilisation of network capacity, without a marked effect on the goal of delay minimisation. We test our method using various scenarios within a real urban area, and a generic urban area, incorporating poor air quality zones.

2 Application Background

2.1 Traffic Control in Urban Areas

Urban Traffic Control (UTC) is normally the responsibility of local authorities whose aims include reducing congestion, improving journey times, increasing the reliability of the road network, safety regulation compliance and traffic pollution limitation. In a recent investigation by some of the authors involving UTC of two major European conurbations [10], it was found that the predominant goal of UTC centres is to minimise delay to the traveller. UTC is distinct from Freeway traffic management, where the number of junctions is much lower, and average speeds much higher. It is also distinct

from traffic controls in rural areas, where the concentration of traffic and sensors is much lower.

UTC forms a key component of *intelligent mobility* (the safe and expeditious movement of people and goods) in urban areas. It is made up of four different levels, namely:

- **operational level**, this is the domain of the commands to individual traffic control devices;
- **tactical level**, this involves the control of individual intersections or small groups of intersection;
- **strategic level**, this involves the management of routes in and around urban areas;
- **external level**, this is the interface to control centres controlling adjacent urban, rural or interurban areas.

On a day to day basis, most UTC happens at an operational or tactical level, with the main controls available to operators being traffic light plans, advisory / warning messages displayed on variable message signs, variable speed limits and information distributed on social media and applications based on open data platforms. Notable traffic lights scheduling systems using linked or self-optimising delay reduction techniques include SCOOT[4], MOVA[5] and SCATS[6] [12, 15, 3]. These control traffic light clusters, changing their own phase lengths depending on current traffic conditions, have embedded triggers which can instantly adjust the behaviour of a set of lights. Such adjustment to "fixed plans" can be made at peak times of the day, usually to cope with the vagaries of rush hour. However, given several weeks notice, if operators want to increase flow in a certain direction, due to for example a large football match, they can act at a strategic level by composing a fixed plan to increase the green light phase through a particular corridor of traffic lights. These plans are typically constructed manually, and can be quite complex to create and validate.

Even the most advanced urban traffic management centres face major challenges, and have shortfalls in what they can deliver. One example of such a challenge is unbalanced distribution of traffic in the road network. In particular, one might observe that in rush hours some roads might be overcrowded by traffic while some other roads are nearly empty. This is mainly caused by the fact that traffic is navigated via the same route between given way-points. Therefore, alternative routes that might not be shorter but faster because of little traffic are not or only rarely exploited, and hence the full capacity of the Network is not used. Another shortfall, as reported in [9], is that there are no automated methods to deal with significant unexpected events such as the sudden loss of a road. Self tuning traffic lights can deal with changes in traffic flow, but fail in the face of serious congestion caused by an incident.

Apart from the manual creation of fixed, connected traffic light plans (e.g. enabling a "corridor" effect as mentioned above) UTC can do little to influence traffic at a strategic or external level, such as flow balancing through different parts of a region. In particular, there are currently little or no controls that an operator can enact on a day by day basis to change traffic behaviour to reduce the traffic through specific areas of a

[4] www.scoot-utc.com

[5] http://www.jstsm.com/mova.html

[6] www.scats.com.au

network, or to select different types of vehicle to travel using different routes. A recent review [2] identified areas where the operation of UTC could be enhanced, highlighting the issues above. In particular, it emphasised the lack of air quality considerations in UTC technologies, as discussed in the next section.

2.2 UTC in the Future

Looking towards the future, UTC is often considered within the "Smart City" initiatives, as its function is to make better use of the physical infrastructure of a city, and potentially reduce usage of "environmental capital". With the use of personalised information services widespread, vehicle route choice is becoming increasingly dependent on automated route finding software, especially for commercial vehicles. Currently these routes are enacted by the driver, but in future transport vehicles will follow calculated routes autonomously. Commercial fleets have a clear motivation [7]; they are planning to install software for their vehicles that will take into account the characteristics of their vehicle and local authority constraints on routes, which in combination with minimising delay criteria will be used to decide routes. In other words, important regional criteria such as air quality will be taken into account in future route navigation systems.

2.3 Air Quality in Urban Areas

In many areas of the world there are in force strict controls on air quality, and local authorities can be fined if an area exceeds the fixed threshold[8]. Naturally, it is within the centre of urban areas where the air quality is poorest, with vehicles by far being the biggest polluter. For example, in 2012 there were 2720.9 km of UK roads that exceeded the EU air quality standards. While this is expected to decrease to 501.2 km by the deadline of 2020, 501.2 km of road exceeding health standards for air quality still represents a significant health issue for the UK.

The parameters that govern air quality in an urban area due to traffic, are:

- area specific: certain areas have poor air quality intrinsically, mainly because they are shielded from the prevailing wind, hence the pollution cannot be dispersed easily;
- vehicle specific: the level of emissions depends on the amount and type of traffic. For example diesel buses, large delivery vans and trucks are about 15 times more polluting than cars;
- weather dominated: if there is little wind in the direction that will disperse the pollutants, or in particular if the day is still, then the problems arise. Wind usually disperses pollutants, but it is also possible that it could acculmulate it into certain areas.

[7] http://analysis.telematicsupdate.com/fleet-and-asset-management/present-and-future-connected-fleets

[8] For example, the European standards are available on http://ec.europa.eu/environment/air/quality/standards.htm

Hence, an area would only likely have air quality problems on certain days when the weather was of a certain type and the volume of traffic was high, and of the wrong sort. Potential management controls at a strategic level include routing higher level polluting vehicles away from the pollution. This is very locality-dependent, e.g., assuming we have a north - south corridor, and a line of high buildings going north - south. Then on days of dangerous air quality, a potential strategy would be to route higher polluting vehicles on the west side of the buildings, since the prevailing wind may disperse them. The east side will be shielded by the buildings, hence will be more likely to be a problem area.

3 Problem Requirements

We performed the study for this paper with an ITS (intelligent transport system) consultant, who is also a co-author of the paper, and were helped by members of, and financially supported by, a European Network in future transport systems[9]. We visited and investigated the current capability, and future needs, of two major conurbations, both with over 1 million citizens. We gathered that in the near future there will be a requirement for software that can work at a UTC strategic level, by producing routes for vehicles through urban areas in partnership with the traveller, the urban traffic management authority, and, in the case of fleet vehicles, the fleet-owning company. The routes produced would need to make full use of the road network, and be aimed at minimising delay. In certain cities where pollution is a problem, it would be necessary, on weather-determined days, for the software to route vehicles in order to avoid breaking the pollution limits.

Whereas fixed plans have been used at a strategic level in UTC, they are created only for expected, well specified disturbances such as large city events, and require much preparation. Hence, it is recognised that the problem outlined is too complex for manual solution. Further, the requirements demand that the solution is flexible enough for potential day-to-day changes in goals and initial state characteristics, e.g., changes to road topology, or goal optimisation constraints, and unexpected events such as sudden road closures. Also, the solution method needs to reason at a strategic level, using the whole of the network within the area, and using knowledge-based features such as types of vehicle and their emission characteristics. To show the feasibility and advantages of using automated planning to generate individual routes, we have produced domain models which embodied a micro-simulation model of traffic, as specified in the next section.

An example scenario in which our approach will fit, is as follows. Vehicles travelling in a region obtain a route for their whole journey, departing from some origin O to some destination D, using a conventional navigation algorithm, but where the calculated routes will be visible by the UTC centres in the region. UTC will filter the routes on whether or not they intersect one of their controlled regions. Assuming the vehicle enters a controlled region at point A, and exiting B, then the UTC centre's software will have the authority to change the route. The part of the route from A to B is recalculated

[9] *Towards Autonomic Road Transport Systems*, EU COST Action TUD 1102

by UTC's planning software, to take into account the authority's regional goals and constraints (in our case air quality, network capacity, vehicle type), and replaces the old section of the route from A to B. Thus, re-routing causes minimal disruption as it focuses only on re-routing vehicles that traverse the urban area. To deal with the delay between a route being requested and the actual travelling of that route, a "plan - validate - replan" approach can be taken [5]. After a route is given, when vehicles arrive at the edge of the UTC centre region, if the time is different from the expected one a checker such as VAL [8] can be used to check all constraints are still satisfied, otherwise the route can be replanned for that vehicle.

4 Problem Formulation

4.1 The Static and Dynamic Model of a Region of the Road Network

We present a formulation of a micro-simulation model of road traffic [13], that is one that considers traffic flow at the level of the individual vehicle (this is necessary as routes need to be calculated for each vehicle). It is worth pointing out that such models in the area of ITS, and Transport Studies in general, are designed to be executed and their behaviour observed. Here we are using such a model *as the input to a reasoning system* (i.e. a planner). In the former, the behaviour of individual traffic is already determined by the model and its semantics, whereas in the latter we are expecting the planner to supply a key part of the picture - the vehicles' routes.

A region of the road network can be represented by a directed graph, where edges stand for road sections and vertices stand for either junctions, entry or exit points. Intuitively, vehicles enter the network in entry points, and exit the network in exit points. Each road section has a given length and capacity (i.e. a maximum number of vehicles it can serve). In junctions, we must consider that vehicles cannot go through it in some directions simultaneously, otherwise they moght collide. In our case, we will consider a simplistic case that at most one vehicle can pass through the junction at once. Notice that such a case reflects 4-stop junctions that are very common in US urban areas.

Definition 1. *Let (V, E) be a directed graph such that $\forall v \in V : (indeg(v) = 0 \Rightarrow outdeg(v) = 1) \wedge (outdeg(v) = 0 \Rightarrow indeg(v) = 1)$[10]. Edges in E represent one-way (or one direction of two-way) road sections. A vertex $v \in V$ represents:*

- *an entry point if $indeg(v) = 0$*
- *an exit point if $outdeg(v) = 0$*
- *a junction otherwise*

Let $C : E \to \mathbb{N}$ be a function representing road section capacity and $l : E \to \mathbb{R}^+$ be a function representing road section length.
Then $\mathcal{N} = \langle V, E, C, l \rangle$ is a Road Network.

[10] $indeg(v)$ represents the number of incoming edges to v and $outdeg(v)$ represents the number of outgoing edges from v

This structure for the road network represents the static part of the environment.

To capture a current traffic situation which forms part of the dynamic part of the environment, we have to consider time. Let T be a set of time-stamps. Then, we can define functions referring to positions of vehicles in the road network as well as usage of the roads. Let $use : E \times T \rightarrow \mathbb{N}_0$ be a function referring to a number of vehicles on the road section in a given time-stamp. Clearly, it must hold that $\forall t \in T, \forall r \in E : use(r,t) \leq C(r)$. For capturing the position of vehicles on road sections in a given time-stamp, we define relations $head$ and $tail$ such that $head \subseteq X \times E \times T$ and $tail \subseteq X \times E \times T$ (X is a set of vehicles). Lastly, we have to capture situations when in a give time-stamp vehicles are ready to enter the network in a given entry point or have just exited the network in a given exit point. For this purpose we define relations $ready$ and $exited$ such that $ready \subseteq X \times V \times T$ and $exited \subseteq X \times V \times T$.

We define four planning operators that are used to navigate vehicles through the network. We assume the operator to be applied in a time-stamp t and its application to last Δt. Notice that Δt might vary even for different instances of the same operator (e.g driving through different road sections may take different amount of time). For the *release-vehicle* and *exit-vehicle* operators we assume instantaneous effects (Δt=0).

- An operator *release-vehicle*(x, v, r) releases a vehicle x at the entry point v to the head of the road section r. As a precondition it must hold that r is an outgoing edge from v, $ready(x, v, t)$ is true and $C(r) > use(r, t)$. The effect of this operator is that $use(r, t) = use(r, t) + 1$ and $head(x, r, t)$ becomes true.
- An operator *drive-through-junction*(x, v, r_1, r_2) navigates a vehicle x from the tail of the road section r_1 through the junction v to the head of the road section r_2. As a precondition it must hold that r_1 is an incoming edge and r_2 an outgoing edge from v, $tail(x, r_1, t)$ is true, $\forall t' \in (t; t + \Delta t) : C(r_2) > use(r_2, t')$, and no instance of operator *drive-through-junction*(x', v, r'_1, r'_2) such that $x \neq x'$ is being executed in $\langle t, t+\Delta t\rangle$. The effect of this operator is that $tail(x, r_1, t)$ becomes false, $head(x, r_2, t + \Delta t)$ becomes true, and $use(r_1, t) = use(r_1, t) - 1$ and $use(r_2, t + \Delta t) = use(r_2, t) + 1$.
- An operator *drive*(x, r) moves a vehicle x from a head of r to its tail. As a precondition it must hold that $head(x, r, t)$ is true. The effect of this operator is that $head(x, r, t)$ becomes false, $tail(x, r, t + \Delta t)$ becomes true.
- An operator *exit-vehicle*(x, v, r) allows a vehicle x to leave the network in the exit point v. As a precondition it must hold that r is an incoming edge to v and $tail(x, r, t)$ is true. The effect of this operator is that $tail(x, r, t)$ becomes false, $exited(x, r, t)$ becomes true, and $use(r, t) = use(r, t) - 1$.

We consider frame axioms that, informally speaking, keeps the same value of a function (or a truth value of a relation) between consecutive time-stamps unless some operator changes it. For instance, $use(r, t) = use(r, t')$ ($t' > t$) if no corresponding instance of an operator modifying the use function is executed in between t and t'.

A planning problem is specified by a road network, which captures the static part of the problem, initial and goal positions of vehicles (the use function is appropriately initialised), which captures the dynamic part of the problem. Timed Initial Literals [6] are used to represent situations when a vehicle is ready to enter the network later. For

example, a vehicle x is ready to enter the network at the entry point e in a time-stamp 5. Then, we represent it (in PDDL2.1) as `(at 5 (ready x e))`. Solution plans are in our case optimised for "make-span", i.e., we minimise a cumulative time needed to navigate vehicles from their initial to their goal positions.

4.2 Extending the Problem Formulation for Pollution Constraints

The problem formulation introduced in the previous subsection can be extended in order to comply with pollution constraints. We divide an urban region into zones, depending on the geography of the region, and taking into account daily weather (wind speed and direction) we assume a limit on the amount of emissions from vehicles can be calculated each day, for each zone - we call this the pollution limit. Further, we assume that each road section belongs to exactly one zone. Formally speaking, given a road network $\mathcal{N} = \langle V, E, C, l \rangle$ we define a set of zones $Z = \{z_1, z_2, \ldots, z_k\}$ such that $\bigcup_{i=1}^{k} z_i = E$ and $\forall i, j, i \neq j : z_i \cap z_j = \emptyset$. For each zone we define a pollution limit as a function $polLimit : Z \rightarrow \mathbb{N}$. For each vehicle we define its emission as a function $emis : X \rightarrow \mathbb{N}$ (X is a set of vehicles). Then, we define a function $polLevel$ that represents a current pollution level in a given zone in a given time-stamp, i.e., $polLevel : Z \times T \rightarrow \mathbb{N}_0$. Clearly, it must hold that $\forall t \in T, \forall z \in Z : polLevel(z,t) \leq polLimit(z)$.

Given the additional constraints, some of the planning operators have to be amended by incorporating additional preconditions and/or effects in order to fulfill the constraints. Again, we assume the operators to be applied in a time-stamp t and its application to last Δt time.

- *release-vehicle*(x, v, r, z) is modified such that in precondition it must hold that $r \in z$ and $polLimit(z) \geq polLevel(z,t) + emis(x)$. The effects are extended by adding $polLevel(z,t) = polLevel(z,t) + emis(x)$.
- *exit-vehicle*(x, v, r, z) is modified such that in precondition it must hold that $r \in z$. The effects are extended by adding $polLevel(z,t) = polLevel(z,t) - emis(x)$.
- *drive-through-junction*$(x, v, r_1, r_2, z_1, z_2)$ is modified such that in precondition it must hold that $r_1 \in z_1$, $r_2 \in z_2$ and if $z_1 \neq z_2$, then $\forall t' \in \langle t; t + \Delta t)$: $polLimit(z_2) \geq polLevel(z_2, t') + emis(x)$. If $z_1 \neq z_2$ the effects are extended by adding are extended by adding $polLevel(z_1,t) = polLevel(z_1,t) - emis(x)$ and $polLevel(z_2, t + \Delta t) = polLevel(z_2,t) + emis(x)$

The change in the pollution level in a given zone is approximated in these operators by increasing it when a vehicle enters a road in that zone, and decreasing it when the vehicle leaves the zone, to approximate the pollution dispersion.

5 Strategic Re-Routing through Planning

5.1 A Two Step Approach

As we introduced in Section 3, we consider a scenario where vehicles already have a planned route for their journey, using conventional routing software. For all vehicles

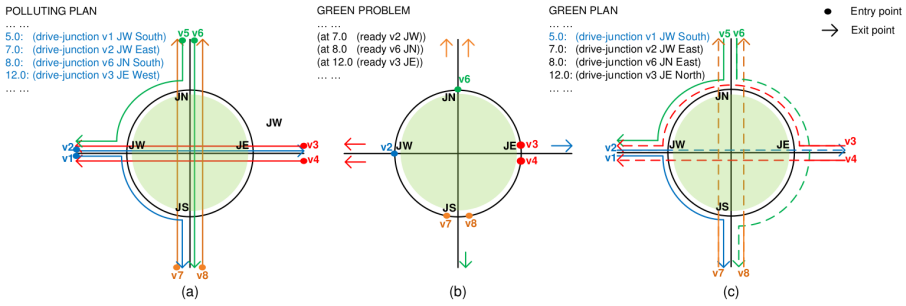


Fig. 1. Example of re-routing: eight vehicles have to reach their exit points, either passing through the city center of using the external ring road.

whose routes are detected to intersect the controlled urban region, their routes are re-planned for that region, in order to utilise the available road network (which would respect any problems resulting from sudden changes in capacity due to e.g. a road incident leading to blockages). It is the responsibility of the UTC to monitor the level of pollution emissions of vehicles traversing the urban area, and, if necessaary, to re-route some of these vehicles in order to maintain the pollution level in the area within the safety limit. In this case a new planning problem is dynamically created by considering the expected entry points of all vehicles into the urban area. Timed Initial Literals are used for this purpose, by modelling both the exact entry point and the expect time of arrival of each vehicle. A new plan is then generated for these vehicles, but this time pollution levels in the urban area must be kept below the safety thresholds, therefore some vehicles may be re-routed.

Finally, the two plans need to be merged, by replacing actions in the initial plan referring to the vehicles that have been re-routed with the corresponding actions in the re-routing plan. It is easy to observe that the final plan is guaranteed to satisfy the pollution limit constraint, as all the vehicles traversing the urban area have been routed while taking the pollution constraints into account. In order to guarantee that at most one vehicle passes through each junction at the same time, and that road capacities are respected, we adopt a delay strategy for the vehicles that have been re-routed. A plan and validate approach is used [5] to dynamically check that the solutions are valid. In summary, dynamic problem formulation and planning are used to accomplish the UTC task of strategic re-routing.

A simple example scenario is provided in the following section to illustrate the method.

5.2 Illustrative Example

Figure 1 shows an example scenario, where eight vehicles `v1...v8` have to reach their exit points, and can choose to traverse the urban area (highlighted in green) or use the external ring road. Figure 1 (a) shows the initial plan containing the best path for each vehicle. Given the entry and exit points of each vehicle, and without taking pollution

emissions into account, for vehicles `v1` and `v5` it is better to use the ring road, while for the other vehicles the shortest path traverses the urban area. The fragment of the corresponding plan (`POLLUTING PLAN`) focuses on vehicles `v1`, `v2`, `v3` and `v6` and shows the time points at which they traverse the junctions at the borders of the urban area[11].

In the next phase, shown in Figure 1 (b), from `POLLUTING PLAN` a new problem is generated, with the objective to re-route the vehicles traversing the urban area in case their pollution emissions exceed the safety limit. To this end, only the vehicles entering the urban area are considered in the new problem, while vehicles `v1` and `v5` are ignored. As shown in the fragment of the new problem (`GREEN PROBLEM`), for vehicles `v2`, `v3` and `v6`, TILs are used to set their new entry points, while the exit points remain the same. The pollution limit constraint is added to the new problem and a new plan is generated, as shown in Figure 1 (c). Here, dotted lines represent the new generated paths. Note that not all vehicles need to be re-routed, as for example vehicles `v2` keeps traversing the urban area. On the other hand, vehicles `v3` and `v6` are re-routed to the ring road in order to keep the pollution emissions in the urban area below the safety limit. The new plan can then be merged with the initial plan, as shown in the fragment of `GREEN PLAN`. Note that the new plan is guaranteed to satisfy the pollution constraint after the merging with the `POLLUTING PLAN`, as from the latter only the vehicles not entering the urban area are considered.

6 Experimental Evaluation

The aim of the experimental evaluation is to test whether the current state of the art of domain-independent planning approaches can perform the required vehicle routing in both real and generic urban scenarios. The data used in these scenarios (the relative percentages of different types of vehicles in the urban area, the relative pollution emissions of vehicles, the road topology of a urban area) has been extracted from openly available and published data. Our experimental evaluation tests whether the current state of the art of domain independent planning approaches can handle the proposed two-step approach, and the possibility to effectively plan the routes of vehicles in a micro-simulation urban context. In particular, the objectives of the particular tests are: i) testing the feasibility of the two-step approach, in terms of CPU time needed; ii) assess the distribution of the traffic on the network, according to road capacities and pollution limits; iii) demonstrate that pollution limits in critical areas can be kept under control, without major delays for the urban traffic.

In our experiment we used the well-known temporal planner POPF [4]. The planner has been run on a cluster with computing nodes equipped with 2.5 Ghz Intel Core 2 Quad Processors, 4 GB of RAM and Linux operating system. A cutoff of 3600 seconds (one hour) was imposed for solving each problem.

[11] For brevity, the action parameters are replaced by the traversed junction and the next direction.

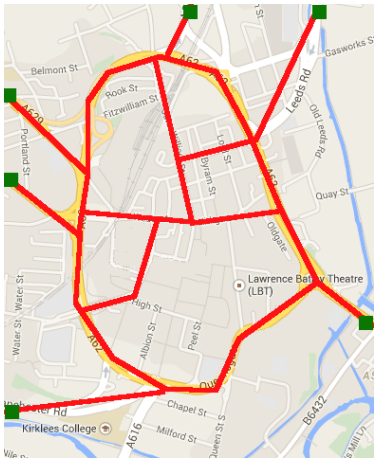


Fig. 2. The map of the scenario 2, which encodes an actual European town. In red the modelled network. Green squares indicate entry/exit points.

6.1 Scenarios

In our experimental evaluation we considered the structure of the typical European city. There is a ring road around the city centre, that allows vehicles to quickly move between different areas of the city, as well as all the intercity traffic. The ring road is elliptic, thus navigation on the north-south direction takes more time than the corresponding west-east navigation. There are four entry points to the city, which is navigable through the ring road and through the roads that cross city centre. The area within the ring road, which represents the densely populated area, as strict pollution limits, in order to limit the access for heavy vehicles, and avoid a very dense traffic.

In a second set of experiments, we encoded in our framework the structure of a real European town. Connections, as well as road length and capacity have been modelled by considering the real data. The map is shown in Figure 2. There are six entry/exit points to the network, which consists of a ring road, and six road sections that can be used for crossing or reaching the town centre.

6.2 Results

In the considered scenarios, we generated problems with the number of vehicles ranging between 20 and 50. Vehicles include cars, vans (6-10%) and heavy vehicles (7-20%), in proportion to the published averages within urban areas of these vehicle groups. They are randomly distributed among entry points, and are released at different times. In testing instances, we considered a rush hour situation, in which many vehicles are released at entry points in a very short time interval, and a more “relaxed” condition, in which vehicles appear less frequently at entry points. Considered problems in both scenarios have been designed for stressing the pollution limits in the urban area, by setting vehicles entry points on one side of the centre, and exit points on the opposite side of the network.

We followed the two-phase approach in both scenarios. We first generated a plan for navigating vehicles throughout the network while ignoring pollution limits. We then generated a new problem file, taking into account the vehicles traversing the urban area and generated a new plan for them, while considering the pollution constraint. All plans were then validated using VAL.

In scenario 1, both finding the first plan, and re-routing vehicles is done efficiently; both steps are usually solved in less than 60 CPU time seconds. We also observed that makespan is not significantly increased when re-routing occurs: average makespan rise is 19.9%, and the number of re-routed cars ranges between 8 (40%) and 14 (28%). On the other hand, we have a large reduction of pollution emissions in the urban area, up to 54%.

In scenario 2, we observed very interesting behaviour of our planning framework. Only problems which include between 20 and 30 vehicles are solved; on the rest, POPF runs out of memory. On solved instances, the first step is computationally expensive, as it requires an average of 451 CPU time seconds to be solved. On the other hand, re-routing vehicles took at most 12 CPU time seconds. Number of vehicles that were re-routed ranged between 5 and 9. Furthermore, re-routing never increased the total makespan, while pollution emissions were reduced in the urban area by between 29% and 36%.

We noticed that in plans produced by both steps, vehicles are effectively distributed across the whole network. This reduces the delay of vehicles, since junctions and roads are less congested. Also, the good distribution of vehicles on the network, permits to spread pollution in different areas of the map, without seriously affecting a specific small area of the network.

7 Discussion

7.1 Directions for Future Work

The experiments demonstrate the extent to which our approach is able to efficiently route vehicles, and re-route them when necessary to respect pollution limits. In scenario 2 at least, we get an idea of the complexity level that current planning technology can reach. Also, the model developed in this paper requires some features (e.g., temporal reasoning, numeric fluents, timed initial literals) that are not supported by many planning engines. Therefore, there is a need for developing advanced planning engines that are able to handle such features. Furthermore, in the current discrete model we make some assumptions on how the pollution level of different zones changes over time. Namely, the pollution level in a given zone is increased (decreased) when the vehicle enters (leaves) that zone. In reality, the pollution level is subject to continuous change, that depends on several factors, such as the amount of time each vehicle is in a given zone, or the current wind velocity and direction. A more realistic modelling of this scenario requires the use of PDDL+ features [7] to model such a continuous change. In particular, PDDL+ continuous processes can be used to model the pollution dispersion due to the wind, or the flows of vehicles through a given road. How to enrich the current model is an interesting direction for future work. Indeed, we believe that lessons learnt

Table 1. Number of cars, makespan extension after re-routing, number of redirected cars, pollution level reduction after re-routing, and total CPU time of problems from scenario 1 and 2, in rush hours or "normal" time.

Scenario 1 – rush hour				
# cars	Δ M	Redir	Δ Poll.	CPU time
20	+42.9%	7	-53%	5.5
30	+0.0%	10	-17%	12.7
40	+27.5%	14	-34%	26.1
50	+0.0%	14	-54%	46.6
Scenario 1 – normal				
# cars	Δ M	Redir	Δ Poll.	CPU time
20	+33.3%	7	-53%	5.8
30	+0.0%	10	-22%	13.6
40	+0.0%	11	-46%	28.3
50	+54.7%	14	-34%	106.2
Scenario 2 – rush hour				
# cars	Δ M	Redir	Δ Poll.	CPU time
20	+0.0%	5	-29%	247.3
Scenario 2 – normal				
# cars	Δ M	Redir	Δ Poll.	CPU time
20	+0.0%	5	-29%	230.7
30	+0.0%	9	-36%	890.8

in this work will help us to improve the current domain model. For instance, we might encode some sort of guidance for vehicles into the domain model (e.g which road the vehicle should take).

Alternatively, there is a possibility to develop and use domain-dependent planning techniques that can be tailored to our application. This might be an efficient approach, but on the other hand, such an approach may be less flexible to changes in requirements.

Decentralised (multi-agent) planning might be another option. Agents (vehicles) might be capable to plan their own routes taking into account the air-quality constraints. This involves some control units responsible for controlling pollution restricted zones that collect data from vehicles passing through the zones and communicate these data with vehicles willing to enter these zones. A possible issue for such an approach might be collision avoidance in junctions. While, the centralised approach can deal with this issue easily, the decentralised approach will have to consider communication between vehicles willing to pass through the same junctions. It might cause some hardly expectable delays and thus it might be hard to determine when a vehicle leaves the zone.

7.2 Related Work

Jimoh et al. [9] recently introduced the idea of using automated planning in urban traffic control as a planning aid to be used in exceptional circumstances, e.g. in situations where roads within a network of roads become blocked due to some unanticipated incident. This demonstrated that it was feasible to use current planning technology to produce re-routing of vehicles using a simple micro-simulation model. The aim of the

work, however, was to provide a supplement to existing tactical UTC, rather than providing a strategic solution at a regional level.

The work of Xie et al. [16] is also aimed at urban traffic control, though again works at a tactical level. Their research has led to the deployment within an urban area of a real-time, adaptive traffic control system called SURTRAC which utilises scheduling techniques to synchronise a group of traffic light clusters, by viewing intersection control optimisation as a scheduling problem. Botea et al. [1] modelled multi-modal journey planning (i.e. generating plans using more than one mode of transport) as a non-deterministic planning problem, and created a heuristic planner for generating contingent plans to advise a user on using combinations of transport (e.g. walking, getting a bus). While this work is relevant to urban areas, it is aimed more at personalised planning rather than strategic vehicle routing.

8 Conclusion

In this paper we have examined the current situation in UTC, where management of traffic is largely done at an operational or tactical level, with personal vehicle route planning performed in isolation of other travellers and of the regional UTC centre.

We investigated the feasibility of applying automated planning to UTC at the strategic level, where individual vehicles are taken into account at a regional level, a development foreseen in the near future for organised (potentially autonomous) vehicle routing. This is seen as having a range of advantages for traffic management, such as the enforcing of regional constraints, and the even distribution through, and utilisation of, network capacity. We formulated the problem in temporal planning, and created a two stage solution which calculated routes for vehicles in an initial pass, re-planning when air quality limits dictates. The results demonstrated the feasibility of the approach on problems of the complexity demonstrated by the real urban area, though considering larger areas / amounts of traffic will clearly challenge current planners.

For the future, we propose to further validate our proposed solution in collaboration with our industrial partners. We also plan and explore promising approaches which will extend the method's scope, like the exploitation of decentralised strategies for planning vehicles routes. Furthermore, we are already working on extending the current approach by considering flows of cars modelled with PDDL+ processes.

References

1. Botea, A., Nikolova, E., Berlingerio, M.: Multi-modal journey planning in the presence of uncertainty. In: Borrajo, D., Kambhampati, S., Oddi, A., Fratini, S. (eds.) Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS. AAAI (2013), http://dblp.uni-trier.de/db/conf/aips/icaps2013.htmlBoteaNB13
2. Cenamor, I., Chrpa, L., Jimoh, F., McCluskey, T.L., Vallati, M.: Planning & scheduling applications in urban traffic management. In: Proceedings of PlanSIG 2014, Annual Workshop of UK Planning & Scheduling Special Interest Group (2014)
3. Chong-White, C., Millar, G., Shaw, S.: SCATS and the environment study: definitive results. In: Proceedings of the 19th World Congress on ITS, Vienna, Austria (2012)

4. Coles, A.J., Coles, A.I., Clark, A., Gilmore, S.T.: Cost-sensitive concurrent planning under duration uncertainty for service level agreements. In: Proceedings of the Twenty First International Conference on Automated Planning and Scheduling (ICAPS-11) (2011)
5. Della Penna, G., Magazzeni, D., Mercorio, F., Intrigila, B.: UPMurphi: A tool for universal planning on PDDL+ problems. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS (2009), http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/707
6. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. J. Artif. Intell. Res.(JAIR) 20, 61–124 (2003)
7. Fox, M., Long, D.: Modelling mixed discrete-continuous domains for planning. J. Artif. Intell. Res. (JAIR) 27, 235–297 (2006), http://dx.doi.org/10.1613/jair.2044
8. Howey, R., Long, D., Fox, M.: VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004). pp. 294–301 (2004), http://doi.ieeecomputersociety.org/10.1109/ICTAI.2004.120
9. Jimoh, F., Chrpa, L., McCluskey, T., Shah, M.M.S.: Towards application of automated planning in urban traffic control. In: 2013 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), pp. 985–990. Institute of Electrical and Electronics Engineers (IEEE) (2013)
10. McCluskey, T.L., Vallati, M.: Extracting information from urban traffic data to improve road traffic management. In: Final Report for Transport Catapult Innovation Voucher (2014)
11. Miles, J. C., W.A.J.: The potential application of artificial intelligence in transport. Journal of Intelligent Transport Systems, Volume 153, Institution of Engineering and Technology (3) (2006)
12. Taale, H., Fransen, W., Dibbits, J.: The second assessment of the SCOOT system in Nijmegen. In: IEE Road Transport Information and Control. No. 21-23 (1998)
13. Treiber, M., Kesting, A.: Traffic flow dynamics. Springer (2013)
14. Various: Artificial intelligence in transportation. Transportation Research Circular E-C113, Transport Research Board (2007)
15. Vincent, R., Pierce, J.: Self-optimising signal control for isolated intersections. In: Crowthorne: Transport and Road Research Laboratory Research Report. No. 170 (1988)
16. Xie, X.F., Smith, S., Barlow, G.: Schedule-driven coordination for real-time trafc network control. In: Proceedings 22nd International Conference on Automated Planning and Scheduling (ICAPS) (2012)

A discrete differential evolution algorithm for multi-objective permutation flowshop scheduling

M. Baioletti, A. Milani, V. Santucci

Dipartimento di Matematica e Informatica
Università degli Studi di Perugia
Via Vanvitelli, 1
Perugia, Italy

Abstract. Real-world versions of the permutation flowshop scheduling problem (PFSP) have a variety of objective criteria to be optimized simultaneously. Multi-objective PFSP is also a relevant combinatorial multi-objective optimization problem. In this paper we propose a multi-objective evolutionary algorithm for PFSPs by extending the previously proposed discrete differential evolution scheme for single-objective PFSPs. The novelties of this proposal reside on the management of the evolved Pareto front and on the selection operator. A preliminary experimental evaluation has been conducted on three bi-objective PFSPs resulting from all the possible bi-objective combinations of the criteria makespan, total flowtime and total tardiness.

Introduction

The Permutation Flowshop Scheduling Problem (PFSP) is an important type of scheduling problem which has many applications in manufacturing and large scale product fabrication. In this problem there are n jobs $J_1, \ldots, J_n$ and m machines $M_1, \ldots, M_m$. Each job J_i is composed by m operations $O_{i1}, \ldots, O_{im}$. The generic operation O_{ij} can be executed only by the machine M_j and its given processing time is p_{ij}. Moreover, the execution of any operation cannot be interrupted (no pre-emption) and job passing is not allowed, i.e., the jobs must be executed using the same order in every machine. The goal of PFSP is to find the optimal job permutation $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$ with respect to a given objective function. Three important criteria are to minimize the total flowtime (TFT), the makespan (MS) and the total tardiness (TT) defined as follows:

$$TFT(\pi) = \sum_{i=1}^{n} C(\pi(i), m) \tag{1}$$

$$MS(\pi) = \max_{i=1,\ldots,n} C(\pi(i), m) = C(\pi(n), m) \tag{2}$$

$$TT(\pi) = \sum_{i=1}^{n} \max\{C(\pi(i), m) - d_{\pi(i)}, 0\} \tag{3}$$

where $C(h, j)$ is the completion time of the operation O_{hj} and is computed by the following recursive equation

$$C(\pi(i), j) = p_{\pi(i),j} + \max\{C(\pi(i-1), j), C(\pi(i), j-1)\}$$

for $i, j \geq 1$, while the terminal cases are $C(\pi(0), j) = C(i, 0) = 0$. In equation (3), for each job h, also a given delivery date d_h is considered.

The minimization of each one of these criteria is computationally hard. Indeed, both the TFT and TT problems are NP-hard for $m \geq 2$, while the MS minimization becomes NP-hard when $m > 2$.

Many single-optimization algorithms exists, either exact or approximate, for instance: heuristic techniques, local searches or evolutionary algorithms [2]. Anyway, in this paper we investigate the PFSP problem as a multi-objective optimization problem, in which the goal is to find a set of job permutations which are good enough with respect to two or more contrasting criteria, i.e. a set of Pareto optimal solutions.

Given k objective functions $f_1, \ldots, f_k$, a solution x dominates a solution x' (denoted by $x \prec x'$) if $f_i(x) \leq f_i(x')$ for $i = 1, \ldots, k$, and there exists at least an index $j \in \{1, \ldots, k\}$ such that $f_j(x) < f_j(x')$. A solution x is Pareto optimal if there exist no other solution x' such that $x' \prec x$. The Pareto optimal set is the set of all the Pareto optimal solution. If two solutions x and x' are such that neither $x \prec x'$ nor $x' \prec x$, then x and x' are incomparable.

Since the Pareto set is in general very large, the goal is to find an approximation of this set, i.e., a set composed by incomparable solutions which is as close as possible to the Pareto optimal set. One of the most promising approaches to solve multi-objective optimization problems is to use evolutionary algorithms [1].

In the context of multi-objective PFSP, many approaches have been proposed. The surveys [3, 7] describe and compare many algorithms for PFSP with all the three possible combinations of two objectives among TFT, MS and TT.

In this paper we describe an algorithm for multi-objective optimization which is based on Differential Evolution for Permutation (DEP) [6]. DEP is a discrete differential evolution algorithm which directly operates on the permutations space and hence is well suited for permutation optimization problems like PFSP. Indeed, in [6] and in [5], it was shown that DEP reaches state-of-the-art results with respect to total flowtime and makespan single objective optimization. Here, DEP has been extended in order to handle multi-objective problems. The preliminary experimental results show that its performances are comparable with state-of-the-art algorithms.

The rest of the paper is organized as follows. The second section describes the classical Differential Evolution algorithm. Its extension to the permutations space and multi-objective PFSP is introduced in the third section. An experimental investigation of the proposed approach is provided in the fourth section, while conclusions are drawn at the end of the paper.

The Differential Evolution algorithm

In this section we provide a short introduction to Differential Evolution (DE) algorithm. For more detail see [4]. Differential Evolution (DE) is a powerful population-based evolutionary algorithm for optimizing non-linear and even non-differentiable real functions

in $\mathcal{R}^n$. The main peculiarity of DE is to exploit the distribution of the solutions' differences in order to probe the search space.

DE initially generates a random population of NP candidate solutions $x_1, \ldots, x_{NP}$ uniformly distributed in the solutions space. At each generation, DE performs mutation and crossover in order to produce a trial vector u_i for each individual x_i, called target vector, in the current population. Each target vector is then replaced in the next generation by the associated trial vector if and only if the produced trial is fitter than the target. This process is iteratively repeated until a stop criterion is met (e.g., a given amount of fitness evaluations has been performed).

The differential mutation is the core operator of DE and generates a mutant vector v_i for each target individual x_i. The most used mutation scheme is "rand/1" and it is defined as follows:

$$v_i = x_{r_0} + F \cdot (x_{r_1} - x_{r_2}) \tag{4}$$

where r_0, r_1, r_2 are three random integers in $[1, NP]$ mutually different among them. x_{r_0} is called base vector, $x_{r_1} - x_{r_2}$ is the difference vector, and $F > 0$ is the scale factor parameter.In [4] it is argued that the differential mutation confers to DE the ability to automatically adapt the mutation step size and orientation to the fitness landscape at hand.

After the mutation, a crossover operator generates a population of NP trial vectors, i.e. u_i, by recombining each pair composed by the generated mutant v_i and its corresponding target x_i. The most used crossover operator is the binomial one that builds the trial vector u_i taking some components from x_i and some other ones from v_i according to the crossover probability $CR \in [0, 1]$.

Finally, in the selection phase, the next generation population is selected by a one-to-one tournament among x_i and u_i for $1 \leq i \leq NP$.

Discrete Differential Evolution for Multi-Objective Optimization

In this section we describe the proposed Multi-Objective Differential Evolution for Permutation (MODEP) which directly evolves a population of NP permutations $\pi_1, \ldots, \pi_{NP}$. With respect to the classical DE, important variations have been made to the genetic operators of mutation, crossover and selection. Moreover, an additional archive of solutions is introduced to maintain the evolved Pareto front.

To simplify our description, let us restrict to the case of two objective functions f_1 and f_2. A population of NP permutations $\pi_1, \ldots, \pi_{NP}$ is randomly generated at the beginning. At each iteration, a secondary population of trial elements $\upsilon_1, \ldots, \upsilon_{NP}$ is generated by means of the mutation and crossover operators. Then, a selection operator selects, for $i = 1, \ldots, NP$, which element among υ_i and π_i should be part of the population for the next iteration.

The pseudo-code of MODEP is depicted in Alg. 1.

Differential Mutation

The mutation operator used is the same of DEP [6]. It produces a mutant ν_i for each population element π_i using some algebraic concepts related to the symmetric group of permutations. Here we briefly recall its structure:

Algorithm 1 MODEP

```
1: Initialize Population
2: Update ND
3: while num_fit_eval ≤ max_fit_eval do
4:     for i ← 1 to NP do
5:         ν_i ← DifferentialMutation(i)
6:         υ_i^(1), υ_i^(2) ← Crossover(π_i, ν_i)
7:         Update ND
8:         υ_i ← SelectChild(υ_i^(1), υ_i^(2))
9:     end for
10:    for i ← 1 to NP do
11:        π_i ← Selection (π_i, υ_i)
12:    end for
13: end while
```

1 Find r_0, r_1, r_2 different to i and to each other
2 $\delta \leftarrow \pi_{r_2}^{-1} \circ \pi_{r_1}$
3 $S \leftarrow RandBS(\delta)$ (S is a sequence of adjacent swaps)
4 $L \leftarrow Length(S)$
5 $k \leftarrow \lceil F \cdot L \rceil$
6 $\nu_i \leftarrow \pi_{r_0}$
7 for $j = 1, \ldots, k$ apply S_j to ν_i

where $\circ$ is the ordinary permutation composition operator, $\cdot^{-1}$ denotes the inverse of a permutation, and $RandBS$ is the randomized bubble sort procedure which allows to decompose a permutation in a sequence of adjacent swaps (that are themselves simple permutations). For more details, see [6].

It is worth to notice that this operator works directly with permutations, simulating from an algebraic point of view, the expression of equation (4).

Crossover

The crossover operator for permutation representations is the same of DEP and produces two children $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$ from π_i and ν_i. The details are described in [6].

The two permutations $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$ are compared with respect to both f_1 and f_2. If $\upsilon_i^{(1)}$ dominates $\upsilon_i^{(2)}$, then the trial υ_i is $\upsilon_i^{(1)}$. Analogously, if $\upsilon_i^{(2)}$ dominates $\upsilon_i^{(1)}$, then the trial υ_i is $\upsilon_i^{(2)}$. When $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$ are incomparable, then one of them is randomly selected to become the trial υ_i.

Selection

The selection operator chooses the new population element π_i' between the old element π_i and the trial ν_i. If $\pi_i \prec \nu_i$, then π_i' becomes π_i, i.e., π_i remains in the population. Otherwise, if $\nu_i \prec \pi_i$ or it is equal to π_i, then π_i' becomes ν_i, that is ν_i replaces π_i in

the next generation population. However, if π_i and ν_i are incomparable, then we use a probabilistic method somehow similar to the α-selection described in [6].

Suppose first that $f_1(\nu_i) < f_1(\pi_i)$ but $f_2(\nu_i) \geq f_2(\pi_i)$. Then, π'_i becomes ν_i with probability $\max\{0, \alpha_2 - \Delta_i^{(2)}\}$, otherwise it retains the old element π_i, where

$$\Delta_i^{(2)} = \frac{f_2(\nu_i) - f_2(\pi_i)}{f_2(\pi_i)}$$

is the relative worsening of ν_i with respect to π_i according to f_2.

Analogously, if $f_1(\nu_i) \geq f_1(\pi_i)$ and $f_2(\nu_i) < f_2(\pi_i)$. Then, π'_i becomes ν_i with probability $\max(0, \alpha_1 - \Delta_i^{(1)})$, where

$$\Delta_i^{(1)} = \frac{f_1(\nu_i) - f_1(\pi_i)}{f_1(\pi_i)}.$$

The rationale behind this selection operator is that ν_i enters the population if it dominates or is equal to π_i or, with a small probability, if it is not too worse than π_i in one of the objective functions, while it is better than π_i in the other objective function. Moreover, note that the probability of accepting a slightly worsening population element linearly shades from α_h, when $\Delta_i^{(h)} = 0$, to 0, when $\Delta_i^{(h)} = \alpha_h$, for $h = 1, 2$.

Therefore, the parameters α_h regulates how worse ν_i can be in order to be accepted in the new population: if $\alpha_1 = \alpha_2 = 0$ only better elements (in the Pareto sense) can replace old elements in the population.

Pareto Front

The algorithm keeps updated the approximated Pareto front ND, which contains all the non-dominated elements ever generated and evaluated. Initially ND contains all the non-dominated population elements created during the random initialization. Then, at each generation, all the couples of children $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$ are used to update ND. A new element υ enters ND if it is not dominated by any element of ND. Moreover, all the elements of ND which are dominated by υ are removed.

Experimental Results

In this section we report some preliminary experimental results obtained with an implementation of MODEP.

The experiments have been performed by solving the well known Taillard's instances with the additional due times given in [3]. These instances are divided in 11 groups of 10 instances with the same values of n and m. The values of n are in the set $\{20, 50, 100, 200\}$, while m lies in $\{5, 10, 20\}$. The combination $(n = 200, m = 5)$ is not considered. The processing time p_{ij} of each instance are randomly generated in $\{1, \ldots, 99\}$, while the due date of each job J_i are generated by multiplying the value $\sum_{j=1}^{m} p_{ij}$ for a random factor in $[1, 4]$. MODEP has been run 10 times for each instance and the adopted stopping criterion is the maximum number of evaluations, which

has been set to $2000 \cdot n \cdot m$. Three combinations of objectives have been considered: (MS, TFT), (MS, TT), and (TFT, TT). For each execution the obtained Pareto front (corresponding to ND) has been analyzed by computing two performance indices: the hypervolume I_H and the unary multiplicative epsilon I_ϵ^1. I_H is computed as the area delimited by the solutions of ND and a reference point. I_ϵ^1 compares ND with the best known Pareto front B and is computed as

$$I_\epsilon^1 = \max_{x \in B} \min_{y \in ND} \max_{j=1,2} \frac{f_j(y)}{f_j(x)}.$$

The indices have been computed by averaging over the multiple executions and instances for every combination of $n \times m$.

The value for the parameter NP has been set to 100 after some preliminary experiments. The parameter F used in the mutation operator is, as in [6], self-adapted during the evolution. Instead, the values for the selection parameters α_1 and α_2 have been set after a calibration phase according to Table 1.

Table 1. Calibration values for α_1 and α_2

Opt.	$\boldsymbol{\alpha_1}$	$\boldsymbol{\alpha_2}$
(MS, TFT)	0.025	0.015
(MS, TT)	0.01	0.01
(TFT, TT)	0.01	0.01

The results of the optimization of (MS, TFT) are shown in Table 2. MODEP works well on this problem and the values of the second index I_ϵ^1 (whose optimal value is 1) are quite good, while the values for I_H (whose optimal value is 1.44) are however good, compared to those reported in [3].It is worth to notice that, fixing n, I_H seems to have a decreasing behavior as m increases (except when $n = 20$).

Table 2. Results for (MS, TFT)

$\boldsymbol{n}$	$\boldsymbol{m}$	$\boldsymbol{I_H}$	$\boldsymbol{I_\epsilon^1}$
20	5	1.089	1.015
20	10	1.185	1.014
20	20	1.188	1.013
50	5	1.248	1.045
50	10	1.149	1.050
50	20	1.119	1.042
100	5	1.238	1.065
100	10	1.140	1.072
100	20	1.067	1.057
200	10	1.143	1.083
200	20	1.058	1.073

The results of the optimization of (MS, TT) are shown in Table 3 and are similar to those for (MS, TFT), even if the decreasing behavior of I_H with respect to m is not so apparent.

Table 3. Results for (MS, TT)

n	m	I_H	I_ϵ^1
20	5	1.241	1.048
20	10	1.136	1.162
20	20	1.071	1.038
50	5	1.219	1.078
50	10	1.140	1.175
50	20	1.195	1.237
100	5	1.221	1.086
100	10	1.137	1.119
100	20	1.138	1.167
200	10	1.138	1.105
200	20	0.976	1.117

Finally, the results of the optimization of (TFT, TT) are shown in Table 4. Here, while the performances as measured by I_ϵ^1 are still satisfactory, the results of I_H are slightly worse than in the previous cases.

Table 4. Results for (TFT, TT)

n	m	I_H	I_ϵ^1
20	5	1.081	1.026
20	10	1.088	1.193
20	20	1.032	1.038
50	5	0.6525	1.024
50	10	0.899	1.083
50	20	1.021	1.206
100	5	0.540	1.027
100	10	0.660	1.055
100	20	0.799	1.103
200	10	0.588	1.051
200	20	0.641	1.077

Conclusion and Future Work

In this paper we have described an algorithm for optimization of multi-objective permutation flowshop scheduling problems. Some preliminary experimental results show that this approach is promising and reaches results which are comparable to the state-of-the-art algorithms. As a future line of research, we would like to add to our algorithm

some method to enhance the diversity of the population, as done in other evolutionary multi-objective algorithms, like crowding distance or niching techniques.

References

1. Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler. Evolutionary multi-objective optimization. *European Journal of Operational Research*, 181(3):1617–1619, 2007.
2. J. Gupta and J.E. Stafford. Flowshop scheduling research after five decades. *European Journal of Operational Research*, (169):699–711, 2006.
3. Gerardo Minella, Rubén Ruiz, and Michele Ciavotta. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20(3):451–471, 2008.
4. K.V. Price, R.M. Storn, and J.A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, Berlin, 2005.
5. Valentino Santucci, Marco Baioletti, and Alfredo Milani. Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm. *submitted to AI Communication*.
6. Valentino Santucci, Marco Baioletti, and Alfredo Milani. A differential evolution algorithm for the permutation flowshop scheduling problem with total flow time criterion. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*, pages 161–170, 2014.
7. M.M. Yenisey and B. Yagmahan. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, (45):119–135, 2014.

Web Services and Automated Planning for Intelligent Calendars

George Markou, Anastasios Alexiadis, Ioannis Refanidis

Department of Applied Informatics, University of Macedonia
Thessaloniki, Greece

gmarkou@uom.gr, talex@java.uom.gr, yrefanid@uom.gr

Abstract. This paper promotes the automated creation of hybrid personal plans, comprising web services and real human activities, to be supported by the next generation of intelligent calendar applications. A prototype work is present-ed, utilizing both atomic web services and composite ones, the latter having been previously generated from a contingent web service composition module, aiming at in-creasing the likelihood of achieving their intended goal. A metric planner generates a plan, giving priority to web service calls over human activities. Then, a scheduler schedules the human activities into the user's calendar, taking into account the ordering constraints that result from the plan. The resulting schedules substitute human activities with web services, thus increasing the user's capacity, his free time, as well as the scheduling options. As a proof of concept we present a case study implementation utilizing existing state-of-the-art components.

Keywords: web services · intelligent calendar · personal activities

1 Introduction

Although in the recent past the popularity of paper calendars has steadily declined in favor of web-based calendar applications like Google Calendar, such applications still do not provide any means for automated activity scheduling. In that way, users are forced to decide for themselves whether a particular activity has enough time to be scheduled alongside another, or whether the distance between the places where the two activities occur is a prohibiting factor.

This problem was efficiently tackled in our previous work, by SELFPLANNER [1], a web-based calendar application that combines a rich problem model with a fast domain-specific scheduler and automatically produces optimized schedules. SELFPLANNER also uses Google Maps to compute the necessary travelling times between the activities' locations, as well as Google Calendar to present the outputted schedules.

The obvious next step in intelligent calendar applications is to employ planning, instead of pure scheduling, to achieve the user's goals. Existing systems, like SELFPLANNER, rely on the user to select the activities to be included in his plan, with

the system providing only scheduling functionalities. We envision a situation where intelligent calendar applications (a) support action ontologies, with action descriptions containing pre-conditions and effects, (b) allow the user to set his goals, (c) know the user's state, and (d) employ auto-mated planning to create plans that achieve the user's goals.

An even further step would be to extend intelligent calendar applications so as to take web services into account. Web services can be considered as normal actions that can be used to achieve users' goals or other actions' preconditions. Incorporating web services into the action ontology enables the substitution of human activities by web service calls, thus allowing for more flexible plans, more goals to achieve or just more free time. As an example of such a setting that is very common in our daily routine consider this: a person may wish to attend a concert, and for that reason he may insert a personal activity in his schedule so as to reserve time and remember to buy tickets for it. To achieve this goal the user is required to physically go to a brick-and-mortar shop or buy his tickets online: both options re-quire some of his time (obviously, buying the tickets online requires less time). A more efficient electronic calendar, on the other hand, would have searched for an alternative - automated - way of achieving such goals, so as to relieve the user from the burden of manually executing the necessary actions.

In this work we propose such an approach; it is based on the integration of web services with an existing metric planner and an intelligent calendar application, namely SELFPLANNER. Web services may be simple or composite; in the latter case, a contingent web service composition module has been used, trying to reduce the nondeterminism underlying their execution. From the planning perspective, though, web services are considered deterministic. As a result of the above process, the user's schedule contains only the human activities that cannot be completed through the use of web services, thus relieving him of the extra burden of achieving the rest.

The rest of the paper is structured as follows: First we present related work; next, we give a motivating example; then we present our approach and, finally, we conclude the paper and pose future directions of research.

2 Related Work

SELFPLANNER was the first system to tackle the problem of automated scheduling personal activities into electronic calendars, using a combination of greedy optimization algorithm, namely a modified version of the Squeaky Wheel Optimization (SWO) [2], and stochastic local search. SELFPLANNER employs a rich model supporting temporal domains and preferences, locations, interruptible and periodic activities, binary constraints and preferences, etc.

Bank et al. [3] build directly upon this work, using SWO in addition to a set of calendar entity types that they propose; in specific, they discriminate between simple events, multiple choice events, floating events and tasks. Moreover, they incorporate elements of psychology into the generation of the schedules, by defining preferences

such as that there should be no wasted travel time between events, or that creative tasks should be split into multiple segments.

La Placa, Pigot and Kabanza [4] follow a different approach, by utilizing Hierarchical Task Network (HTN) Planning [5] and focusing on a specific user target group. Their approach is directed towards people with cognitive impairments, e.g., Alzheimer's disease, and as such, HTN planning, which decomposes tasks into subtasks, is well suited as it resembles the way medical professional actually plan for their patients. Moreover, this degree of granularity is dependent on the specific patient, with information such as his impairment or personality being taken into account.

Finally, Berry et al. [6] present Emma, a personalized calendar management tool, which simultaneously manages calendars from multiple sources, with the main aims of facilitating the coordination of groups of people, the negotiation of their meeting times and the (re)scheduling of various events.

In regard to non-deterministic web service composition, Kuzu and Cicekli [7] present a conversion schema from OWL-S to PDDL and utilize an existing PDDL planner, namely Simplanner [8], to tackle non-determinism, through interleaving planning and execution. Zou et al. [9] follow a similar approach, albeit to generate a distributed plan; first, a web service choreography problem that contains explicit user defined contingencies is translated into a deterministic planning one and then, either FF [10] or SatPlan06 [11] are employed to solve it.

Dacosta et al. [12] on the other hand, opt for a stratified method so as to produce robust plans, which allow for semantic web services that achieve the same tasks. The approach generates a graph that contains all the possible contingency plans, with each path in it being a possible execution path, and each child node comprising an alternative execution possibility. Redundant operations have been removed from the graph, and the set of paths is ordered from the best – the one containing the smaller number of web services – to the worst.

In our previous work [13] we implemented MAPPPA, a cost sensitive probabilistic contingent planning approach specifically targeted for automated semantic web service composition. MAPPPA produces a contingent plan by integrating alternative deterministic plans previously computed in an anytime fashion from a determinized version of the original problem. It does so, however, without disregarding the information that each web service contains in relation to its execution cost and probability of alternative outcomes, thus, generating considerably more informed plans than other determinization approaches.

3 Motivating Example

Let us imagine a usual week of a Bob, who uses a web-based calendar application to organize his time. Due to being self-employed, Bob needs to spend all working hours at his office, to which he commutes every day from his home. Once every week, he watches a movie, and although he prefers to go to the cinema, sometimes he watches the movie at home, depending on his work schedule. However, if the movie's duration is such that it will end later than 11 pm, he does not desire to watch a movie at

all, as he has to sleep early. Moreover, this week, he will travel on a business trip abroad; a day before the trip, the day he usually watches a movie, he has to book his airplane tickets and hotel, as well as to buy a travel guide for the city he will visit.

In order to schedule these activities and insert them into a calendar, the user has to define their temporal domain and, if they are interruptible, the minimum and maximum allowed duration for their parts. Moreover, for each activity, the user has to declare whether it is periodic or not, as well as if it is bound to specific locations. For example, the user should define his daily work as a periodic task, with a temporal domain from 8 a.m. to 5 p.m., and a minimum and maximum duration of its parts – as he cannot work continuously, e.g., 30 minutes and 2 hours respectively. Watching a movie is also periodic but non-interruptible, and has a temporal domain set late in the evening, with a minimum and maximum duration of 90 minutes and 180 minutes respectively.

Since the user's work requires his physical presence, only a human activity can be placed in his calendar. However, for the rest of the aforementioned tasks, a combination of human and web services' activities can be inserted. The user may have to drive to the cinema, choose among the available movies there and buy the tickets himself. Alternatively, he may ask for a list of the available movies to be emailed to him, book the tickets online and then, having saved considerable time, travel to the cinema later. Another option altogether would be to rent a movie online and watch it at home. As to the user's business trip, again, there are various alternative activities; the user may visit a single tourist agent to book his tickets and hotel; or prefer an online reseller. He may also require another trip to a bookstore to purchase his travelling guide or purchase it online and have it sent at home.

Since these options create a complex problem, containing a multitude of constraints and preferences, a schedule manually created by a human is usually highly inefficient. Applications such as SELFPLANNER tackle this problem; however, they only deal with human activities and, as such, they cannot take advantage of the opportunities that are offered by the use of web services. In the aforementioned scenario, since a human activity requires the user to first purchase a movie ticket himself, in certain situations he may not have had the time to do so, and he would have had to watch it at home instead. Even worse, if he had to visit a travel agent and a bookstore, he may not have even had the time to watch a movie at home.

With the introduction of web services, the user saves the time needed to perform the actual action of purchasing these services and to travel between locations.

4 Proposed Approach

This work assumes that web services are semantically annotated and that their descriptions are present in an online registry; in previous work, we presented such a registry [14], which, furthermore, provides a translation of each web service to a PDDL action. Moreover, this registry also contains composite web services fortified against non-determinism, having been generated by MAPPPA prior to the start of the scheduling process. Thus, the composite web services used comprise multiple execu-

tion paths achieving the same result, and for this reason can only fail when all the execution paths fail.

Fig. 1 presents the proposed system's architecture. Initially, we employ an existing metric planner, namely LPG-td [15]; this step is necessary in order to automate the planning process as the activities in SELFPLANNER are normally entered by the users. Instead, in this case, in order to obtain the set of activities that achieve the desired goal a planner has to be utilized. LPG-td receives as input a planning problem containing both web services and human activities.

This problem comprises of the translation of the web services from the registry, as well as a simplified version of the human activities; that is, the locations, durations and temporal domains of the activities, along with any preferences and constraints in regard to them are removed. The metric planner does not differentiate between the two types of activities; as a result, it also treats all web services as deterministic ones, i.e., as if their intended output (the most probable one) is always outputted. Moreover, the planner has to take into account that web services are preferred to their human activities' counterpart. This is achieved by setting the cost of human activities higher than that of web services.

LPG-td is capable of generating a sequence of alternative plans, each being an improvement – in terms of the specified metric – compared to the previous one; the plans generated by LPG-td allow for parallel actions and, thus, are very similar to partial order ones.

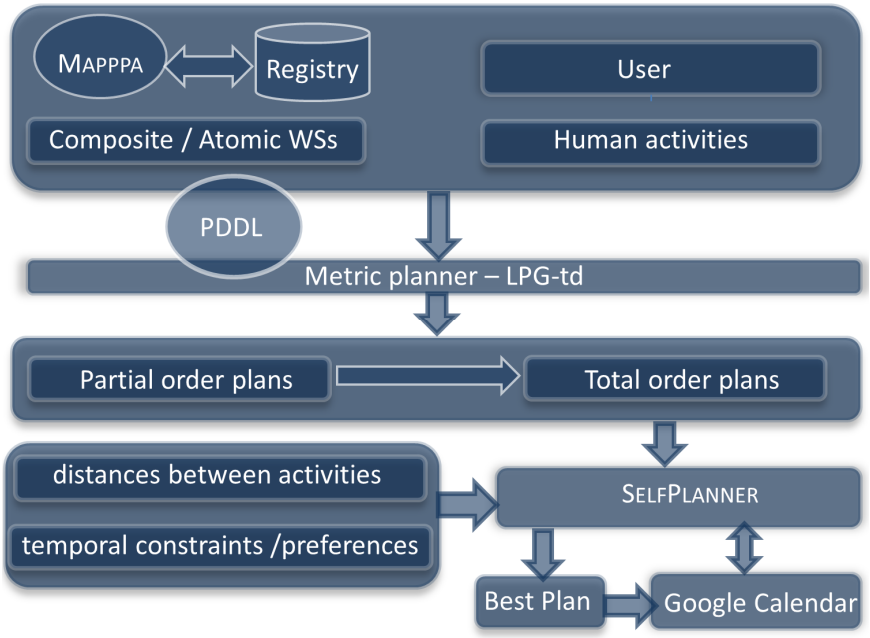


Fig. 1. Proposed system's architecture overview.

We feed the best generated plan by LPG-td to SELFPLANNER so as to create a detailed schedule. In case a feasible plan does not exist, the rest of the previously generated alternative partial order plans are attempted to be scheduled in a similar process. In order to schedule the activities of the partial order plan, SELFPLANNER employs the information concerning temporal constraints and preferences (loaded from a separate activity definition file for a given problem instance), as well as the actual distances between the activities' locations as returned by the Google Maps Distance Matrix API.[1] Web services are considered to have an open temporal domain and are not related to a specific location, as they can be executed at any time and place. Moreover, they can be scheduled in parallel to human activities. SELFPLANNER can generate multiple alternative plans; the human activities of the best schedule are uploaded into the user's Google calendar.

5 Conclusions and Future Work

This paper presents the first steps towards the next generation of intelligent calendar applications. We propose an approach comprising a contingent web-service composition system, a partial order metric planner and a scheduler to insert human activities into a user's web calendar. The contingent planner is used to create composite web services, able to achieve complex goals with high probability of success; the metric planner is used to select an optimized set of activities to achieve a user's goal, favoring web service calls than real activities; and, finally, the scheduler automatically produces an optimized plan based on the user's constraints and preferences.

The paper also presents a motivating example along with a – still under work – implementation. We aim to further improve this implementation, first by providing a graphical interface, as well as by integrating it with a web service execution platform. Also, we propose to perform an exploratory evaluation, by providing the - users with two schedules, one consisting solely of human activities and an equivalent one comprising both human and web service activities, and having them rate each one online.

Finally, various research and implementation issues are still open; for example, we assume that the user's schedule should only contain the human activities that cannot be substituted by web services; this is achieved through setting the cost of human activities higher than that of web services. However, in some cases this may not be true; employing a web service may incur a (financial) cost that the user does not prefer over the benefit of not employing the respective human activity. Moreover, a web service activity may not always provide exactly the same functionality or satisfaction as its human counterpart; such a case is present in our motivating example, in which a user prefers to go to the cinema to watch a movie, than watching it at home through streaming. In this case, however, if the problem is modeled so as to favor the human activity, it could be impossible to include it in the plan, thus providing the use of a web service as an alternative plan. Such problems require further investigation.

[1] https://developers.google.com/maps/documentation/distancematrix/

Acknowledgment. This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

References

1. Refanidis, I., Alexiadis, A.: Deployment and evaluation of SELFPLANNER, an automated individual task management system. Comput. Intell. 27(1), 41-59 (2011)
2. Joslin, D., Clements, D.: Squeaky wheel optimization. J. Artif. Intell. Res. 10, 353-373 (1999)
3. Bank, J., Cain, Z., Shoham, Y., Suen, C., Ariely, D.: Turning personal calendars into scheduling assistants. In: Extended Abstracts of the 30th CHI Conference on Human Factors in Computing Systems (2012)
4. La Placa, M., Pigot, H., Kabanza, F.: Assistive planning for people with cognitive impairments. In: Proceedings of the Workshop on Intelligent Systems for Assisted Cognition hosted by the 21st International Joint Conference on Artificial Intelligence (2009)
5. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
6. Berry, P. M., Donneau-Golencer, T., Duong, K., Gervasio, M., Peintner, B., Yorke-Smith, N.: Evaluating user-adaptive systems: Lessons from experiences with a personalized meeting scheduling assistant. In: Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, pp. 40–46 (2009)
7. Kuzu, M., Cicekli, N.: Dynamic planning approach to automated web service composition. Appl. Artif. Intell. 36(1), 1-28 (2012)
8. Onaindia, E., Sapena, O., Sebastia, L., Marzal, E.: SimPlanner: An execution-monitoring system for replanning in dynamic worlds. In: Proceedings of the 10th Portuguese Conference on Artificial Intelligence, pp. 393-400 (2001)
9. Zou, G., Chen, Y., Xu, Y., Huang, R., Xiang, Y.: Towards automated choreographing of web services using planning. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (2012)
10. Hoffmann, J.: FF: The Fast-Forward Planning System. AI Mag. 22(3). 57-62 (2001)
11. Kautz, H., Selman, B., Hoffmann, J.: SatPlan: Planning as Satisfiability. In: Proceedings of the 5th International Planning Competition (2006)
12. Dacosta, L.A.G., Pires, P.F., Mattoso, M.: Automatic Composition of Web Services with Contingency Plans. In: Proceedings of the 2nd International Conference on Web Services Workshop (2004)
13. Markou, G., Refanidis, I.: Anytime planning for web service composition via alternative plan merging. In: Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence, pp. 91-98 (2014)
14. Markou, G., Refanidis, I.: Composing semantic web services online and an evaluation framework. Int. J. on Adv. in Internet Tech. 6(3-4), 114-131 (2013)
15. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: Fast Planning in Domains with Derived Predicates: An Approach Based on Rule-Action Graphs and Local Search. In: Proceedings of the 20th AAAI National Conference on Artificial Intelligence (2005)

Social Continual Planning in Open Multiagent Systems

Matteo Baldoni, Cristina Baroglio, Roberto Micalizio

Università degli Studi di Torino — Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)
firstname.lastname@unito.it

Abstract. We describe a Multiagent Planning approach, named Social Continual Planning, that tackles *open* scenarios, where agents can join and leave the system dynamically. The planning task is not defined from a global point of view, setting a global objective, but we allow each agent to pursue its own subset of goals. We take a *social* perspective where, although each agent has its own planning task and planning algorithm, it needs to get engaged with others for accomplishing its own goals. Cooperation is not forced but, thanks to the abstraction of *social commitment* stems from the needs of the agents.

1 Introduction

The ability to plan one's own activities, even in dynamic and challenging scenarios such as Multiagent Systems (MAS), represents a key feature in many real-world applicative domains (see e.g., logistics, air traffic control, rescue missions, and so on). Not surprisingly, planning in MAS is drawing the attention of an ever growing number of researchers, as witnessed by the new series of *Distributed and Multi-Agent Planning Workshops* hosted by ICAPS.

The term Multiagent Planning (MAP) refers to a planning task in which a set of planning agents, each equipped with its own tools and capabilities, has to synthesize a *joint solution* (i.e., a joint multiagent plan). The planning task usually involves a number of interdependent subgoals, so that some form of coordination among the agents is necessary to solve the problem. Different methodologies have been proposed in the literature. Besides centralized approaches (e.g., [2]), which fall outside the above notion of MAP, the other distributed solutions can be categorized into three main families, depending on when the coordination among the agents is actually performed. First of all, coordination can be performed after that each agent has completed its own planning process [8, 9]: each agent works on a portion of the problem, and then coordinates with others to resolve conflicts; this requires that agents exchange with each other their partial solutions, and that they are willing to revise their plans to overcome problems. Second, coordination can be interleaved with the planning process [10, 11, 15]: agents continuously exchange information to achieve a joint solution. Finally, coordination can occur before the planning process. In such a case domain dependent coordination structures are given to the agents as a further input. For instance, in [6] a hierarchical decomposition of tasks and their dependencies is given to the agents in order to guide their local planning processes.

In all the above approaches, the planning task defines a global objective to be achieved by means of a "joint solution" involving the capabilities of the agents. Moreover, the set of agents to be involved is known in advance and cannot change during the planning process; the system is therefore closed. In this paper we deal with a *different planning problem*, and propose a methodology named *Social Continual Planning* (SCP), to tackle it. We consider the planning problem of an agent situated in an open multiagent system. The agent may resort on other agents for solving a task of its own interest. The agent plans both its own actions, and its interactions with others whenever it is not capable, or it deems as not convenient, to execute certain steps in autonomy. The focus is not on negotiation, but on the framework through which an agent seeks the help by the others, and on the *engagements* that bind agents to supporting each other. Interaction is not limited to communication but it is a process through which the involved agents progress each in the solution of its own task. Engagements are binary social relationships, that are established dynamically and that create expectations on the involved agents behavior. An agent autonomously decides (plans) when to bind to another one to do something.

More precisely, we take a *social* perspective in the sense that, even though each agent has its own planning task and uses its own planning algorithm, the agent has still to get engaged with others in order to accomplish its own goals. The interactions that an agent has with others will, in general, allow both parties to get closer to their own goals. Cooperation is not forced to the agents just because they are part of the system, but rather cooperation stems from the needs of the agents within the system, and endures as far as the parties take advantage of it. In other terms, we propose a form of (agent) planning which is *situated* in a multiagent system, where an agent not only has to plan its own actions, but has also to plan its social relationships with other agents. Since the coordination has to be planned, it must be supported by a proper abstraction that enables one agent to create expectations about the behaviors of others. To this end, in this paper we adopt *social commitments* [16]. Interestingly, a recent work by Telang et al. [20] shows how goals and commitments are strongly interrelated by means of a set of practical rules. This supports our intuition that commitments may play a central role, together with beliefs and goals, in the synthesis of a plan in a multiagent setting.

The paper is organized as follows. Section 2 overviews the most relevant literature. Section 3 introduces the necessary background. Section 4 explains our proposal. Section 5 describes the implementation, and exemplify the approach in the logistic scenario. Conclusions and a discussion end the paper.

2 Related Work

To the best of our knowledge, the SCP problem has not been tackled in the literature, so far. It is, however, worthwhile to report the main approaches that are currently discussed in multiagent and distributed planning. Since the seminal work by Boutilier et al. [2], the multiagent planning problem has taken on the perspective of finding a coordinated, joint solution to a given planning task. Agents are therefore seen as resources to be managed so as to achieve the global goal. For instance, in [2] a centralized extension to the Partial-Ordered Planning (POP) approach for dealing with multiple plan

executors was proposed. The solution took into consideration possible concurrency and non-concurrency constraints on the execution of actions that had to be satisfied by any feasible plan.

More recently, distributed approaches have emerged within the planning community. However, even though the planning search can be distributed among the agents, the definition of the planning task is still centralized in most of them. See for instance the MA-STRIPS formalization [4], in which, despite each agent has its own set of (private) actions, the initial state and the goal state of the planning task are globally defined. Similarly, in [21] a multiagent plan is seen as a solution of a coordination problem where constraints on resources and tasks are defined globally.

Distributed approaches can be distinguished on how the planning and coordination phases are actually carried on. First attempts to coordinating plan *after* the planning phases [8, 9] suffered from a sever drawback: whenever conflicts were detected between any two plans, the agents had to revise their plans accordingly. Thus, the domain knowledge about conflicts and constraints was not used actively during the planning phase, but only *a posteriori* to verify the correctness of the joint solution.

This drawback is overcome by approaches (see e.g., [10, 11, 15]) in which the coordination and planning phases are *interleaved*. These approaches rely on the exchange of various kinds of information, such as partial plans, or states inferred during the search, so that conflicts are discovered as soon as possible, and corrections can be made while the planning phase is still in progress. The planning phase is carried on by means of a distributed algorithm, and this implies a form of coupling of the agents. The heterogeneity of the agents is limited to the set of actions they can perform, but the planning strategy must be the same for all the agents in the team. Another implicit assumption in distributed approaches is that agents be cooperative and possibly disclose sensible data, e.g. their internal states and resources, and their local goals. Notably, SECURE-MAFS [3], a recent version of the very efficient MAFS algorithm [15], guarantees, to some extent, the privacy of the cooperating agents; yet, it still requires that each agent knows at least the "public interface" of the other agents' actions. A last family of approaches set the coordination phase *before* the planning one (see e.g., [6]). Such solutions, however, assume that all the possible conflicts are known in advance and globally defined.

For what concerns commitments, only recently there have been some attempts to integrate them in planning problems, see [13, 19, 14] which as well as our work rely on the rules proposed in [20]. The idea of translating pragmatic rules into a planning language is first proposed in [19], where the Hierarchical Task Network (HTN) formalization is used. HTNs, however, are used at design time to model and verify commitment protocols [13]; thus, the point of view of these works is still centralized. In this work we will consider a STRIPS-like representation of the pragmatic rules, and use them for generative planning in a context where a centralized point of view is missing. In other terms, in this paper the interactions via commitments are not outlined within predesigned HTNs, but have to be discovered at execution time by the planning search.

3 Background

3.1 Commitments

The proposal presented in this paper strongly relies on *social commitments* (simply commitments below), as first introduced in [17]. Commitments arise, exist, are satisfied, revoked, or otherwise manipulated, all in a social context. They not only rely on the social structure of the groups in which they exist, but also help create that structure. They are revokable. They overcome the subjectivist bias of traditional AI, so generally the conditions associated to a commitment are evaluated in the world, not in the mind of any agent. More specifically, a commitment $\mathsf{C}(x, y, s, u)$ formalizes a relationship between an agent x, playing the role of *debtor*, and another agent y, playing the role of *creditor*: the debtor is committed towards the creditor to bring about a consequent condition u, whenever an antecedent condition occurs s. Antecedent and consequent conditions are conjunctions or disjunctions of events and commitments and they concern only the *observable behavior* of the agents.

Commitments have a *life cycle*, and we adopt the one proposed in [20], and reported in Figure 1. Briefly, a commitment is *Null* right before being created; *Active* when it is created. Active has two substates: *Conditional* (as long as the antecedent condition did not occur), and *Detached* (when the antecedent condition occurred). In the latter case, the debtor is now engaged in the consequent condition of the commitment. An Active commitment can become: *Pending* if suspended; *Satisfied*, if the engagement is accomplished; *Expired*, if it will not be necessary to accomplish the consequent condition; *Terminated* if the commitment is canceled when Conditional or released when Active; and finally, *Violated* when its antecedent has been satisfied, but its consequent will be forever false, or it is canceled when Detached (the debtor will be considered liable for the violation).

Commitments are manipulated by commitment operations such as: *create* (an agent creates a commitment toward someone), *cancel* (a debtor withdraws an own commitment), *release* (an agent withdraws a commitment of which it is the creditor), *assign* (a new creditor is specified by the previous one), *delegate* (a new debtor is specified by the previous one), *discharge* (the commitment is resolved). In particular, when the

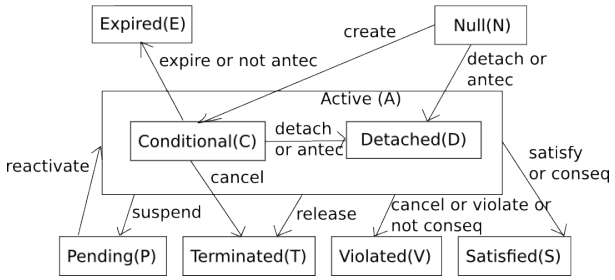


Fig. 1. Commitments life cycle.

consequent condition u holds, the commitment is *discharged*. Notably, only an agent playing the role of debtor can create a commitment.

The reason for relying on commitments stems by the fact that commitments have a *normative* power: Since debtors are expected to behave so as to satisfy their engagements, commitments create social expectations on the agents' behaviors [7]. From a practical reasoning point of view, this means that an agent is expected to behave so as to achieve the consequent conditions of an Active commitment of which it is the debtor. Instead, when the agent is creditor of a commitment, it will set as goal the antecedent condition of the same commitment when it deems it needs the debtor to pursue the consequent condition. Therefore, commitments can be used by agents in their practical reasoning together with beliefs, intentions, and goals for taking into account other agents and the conditions the latter committed to have achieved.

There is a vast literature on social commitments. A comprehensive starting point for the interested reader is [18].

3.2 Goal Formalization

The notion of *goal* plays an important role not only from the point of view of planning, but also in general whenever one has to design and develop intelligent agents. In this paper, we take advantage of the formalization initially proposed in [22], and subsequently revised in [20]; specifically, a goal G is a tuple $\mathsf{G}(x, p, r, q, s, f)$, where x is the agent pursuing G, p is a precondition that must be satisfied before G can be considered active, r is an invariant condition that holds until the achievement of G, q is a post-condition (effect) that becomes true when G is successfully achieved, and finally, s and f are the success and failure conditions, respectively. In other terms, one can think of p as the context in which an attempt to achieve G can be pursued, r as the set required resources, and q as the effects of reaching G. Note that q and s need not to coincide (see [22]). This formalization includes both the specification of an internal procedure (i.e., p, r, and q) that agent x can adopt to satisfy G, and an abstract, declarative success condition s, which is an agent-independent description of G. This complementarity turns out to be fundamental for an agent to decide (1) when cooperation is required (e.g., when no local procedure is applicable), and (2) how to carry through cooperation (e.g., by asking others what success condition bringing about). In the rest of the paper, we will mainly exploit the declarative formalization since we are interested in those cases in which an agent has to cooperate with others.

As well as commitments, goals have a life cycle in which state transitions are triggered by the execution of proper goal actions. Figure 2 shows the goal life cycle [20].

3.3 Pragmatic Rules

In [20] the relation between goals and commitments has been studied, and it has been formalized in terms of practical rules, which capture patterns of pragmatic reasoning. We review in this section a few pragmatic rules through their operational semantics. We first recall the notion of *agent configuration*: the configuration of an agent x is the tuple $S_x = \langle \mathcal{B}, \mathcal{G}, \mathcal{C} \rangle$ where $\mathcal{B}$ is its set of beliefs about the current snapshot of the world, $\mathcal{G}$ is the set of agent's goals, and $\mathcal{C}$ its set of commitments; i.e., commitments in which x

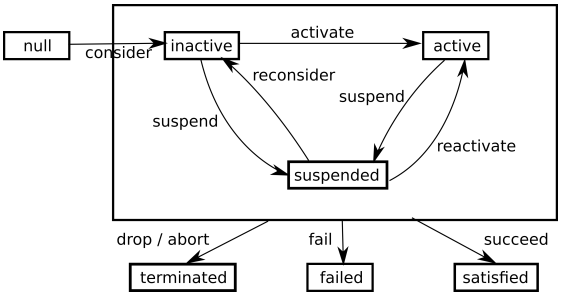


Fig. 2. Goal life cycle

is involved either as debtor or as creditor. We use subscripts from Figure 1 to denote the state of commitments and goals.

The operational semantics of pragmatic rules is given via guarded rules in which S_i are configurations:

$$\frac{guard}{S_1 \longrightarrow S_2}$$

Where $guard$ is a condition over the current agent's beliefs and commitments; whereas $S_1 \longrightarrow S_2$ is a state transition involving a change in the state of commitments or goals; usually it corresponds to an operation on goals or commitments. Pragmatic rules are distinguished into: (1) rules from goals to commitments, they involve commitments that are used as a means to achieve some goal; and (2) rules from commitments to goals, they involve goals that are used as a means to achieve either the antecedent (if the agent at issue is debtor) or the consequent (if creditor) condition of a commitment.

Rules From Goals to Commitments. These rules address situations in which an agent manipulates (e.g., create, or cancel) a commitment to achieve a goal it cannot obtain alone, or to drop a goal is no longer required. Let $G \in \mathcal{G}$ be an agent goal $\mathsf{G}(x, p, r, q, s, f)$, and $C \in \mathcal{C}$ be a commitment $C(x, y, s, u)$ (note that the success condition s of G appears as antecedent in C):

– ENTICE: (Only) by creating the commitment can the agent satisfy its goal. If G is active and C is null, x creates an offer to another agent

$$\frac{\langle G^A, C^N \rangle}{\mathsf{create}(C)} \quad \text{ENTICE}$$

– WITHDRAW OFFER: The commitment is of no utility once the end goal for which it is created no longer exists. If G fails or is terminated, then x cancels C.

$$\frac{\langle G^{T \vee F}, C^A \rangle}{\mathsf{cancel}(C)} \quad \text{WITHDRAW OFFER}$$

Rules from Commitments to Goals. The general idea of these rules is that an agent manipulates a goal to satisfy the antecedent or the consequence of a commitment in

which the agent is involved. Let C be a commitment $\mathsf{C}(x, y, s, u)$; and let us consider two goals $G_1 = \mathsf{G}(x, p, r, q, u, f)$ and $G_2 = \mathsf{G}(y, p', r', q', s, f')$

- DELIVER: The agent activates a goal that would lead to discharging its commitment. If C becomes detached (i.e., goal G_2 has been satisfied), then debtor x activates a goal G_1 to bring about the consequent:

$$\frac{\langle G_1^N, C^D \rangle}{\mathsf{consider}(G_1) \wedge \mathsf{activate}(G_1)} \quad \text{DELIVER}$$

- DETACH: The creditor brings about the antecedent hoping to influence the debtor to discharge the commitment. If G_2 is null and C is conditional, agent y considers and activates G_2:

$$\frac{\langle G_2^N, C^C \rangle}{\mathsf{consider}(G_2) \wedge \mathsf{activate}(G_2)} \quad \text{DETACH}$$

4 The Social Continual Planning Problem

A *Social Continual Planning (SCP) system* is an open environment inhabited by heterogeneous and independent agents. Each agent has its own planning task, and can perform a specific set of actions. A *Social Continual Planning Problem* is a planning problem of an agent, situated within an SCP system, which, for being solved, requires the agent to plan also a set of engagements, realized as social commitments, with other agents in the system. Agents can join and leave the system dynamically; however, we assume that no agent leaves the system as long as there are active commitments involving it either as debtor or as creditor.

More formally, an SCP system is a tuple $\langle \mathcal{U}, \mathcal{A}, \mathcal{S} \rangle$ where:

- $\mathcal{U}$ is a finite set of propositional atoms, whose truth value can be observed by all the agents in the SCP; $\mathcal{U}$ represents a sort of common language through which agents can interact. Atoms in this set are used to describe the state of the environment shared by the agents. In addition, these are the atoms that can appear as antecedents and consequents of the commitments.
- $\mathcal{A}$ is a set of agents; each agent $i \in \mathcal{A}$ is associated with a configuration which extends the agent configuration we have already introduced. Specifically, the agent configuration for agent i is a tuple $\langle \mathcal{B}^i, \mathcal{G}^i, \mathcal{C}^i, Acts^i, Socs^i \rangle$: $\mathcal{B}^i$, $\mathcal{G}^i$, and $\mathcal{C}^i$ are as before; whereas:
  - $Acts^i$ is a set of actions agent i can perform; it is partitioned into:
    - Φ^i is a set of "physical" actions; as usual, these actions are defined in terms of preconditions and effects, which can be both conditions on environment atoms (i.e., in $\mathcal{U}$) or on internal (agent-dependent) atoms that are not globally traced (i.e., the internal state of an agent is private).
    - Σ^i is a set of *social actions*; preconditions and effects are defined in terms of goals in $\mathcal{G}^i$ and commitments in $\mathcal{C}^i$. More precisely, each social action corresponds to a pragmatic rule from goals to commitments. Indeed,

we consider these pragmatic rules as actions because, as we discuss below, they can be used by an automated planner to plan interactions with other agents. Note that while goals in $\mathcal{G}^i$ are *private* (only agent i can see and manipulate them), commitments in $\mathcal{C}^i$ have a *social value*: whenever i changes the state of a commitment in $\mathcal{C}^i$, this change becomes visible to all the other agents in the system (see $\mathcal{S}$).

- $Socs^i$ is a set of pragmatic rules from commitments to goals adopted by an agents; from our point of view these rules define the *social strategy* of agent i. Thus, these rules are not used during the planning search, but rather to decide which goals should be pursued.

– $\mathcal{S}$ is the social state shared by all the agents in the SCP system at hand. The social state can be partitioned into two subsets:
  - $\mathcal{S}^{\mathcal{C}}$ is the set of all the active commitments defined between any two agents in $\mathcal{A}$; in particular, for each agent $i \in \mathcal{A}$, $\mathcal{C}^i \subseteq \mathcal{S}$, $\mathcal{C}^i$ is the projection of $\mathcal{S}$ over all the commitments in which i appears either as debtor or as creditor.
  - $\mathcal{S}^{\mathcal{E}}$ is the set of all the propositional atoms describing the environment that hold at a given time; in particular, $\mathcal{S}^{\mathcal{E}} \subseteq \mathcal{U}$.

Given an SCP system $\langle \mathcal{U}, \mathcal{A}, \mathcal{S} \rangle$, let $i \in \mathcal{A}$ be an agent, that is described by the tuple $\langle \mathcal{B}^i, \mathcal{G}^i, \mathcal{C}^i, Acts^i, Socs^i \rangle$. An SCP problem for i amounts to finding a plan, composed by $Acts^i$ and $Socs^i$, to achieve $\mathcal{G}^i$ starting from $\mathcal{B}^i$. In particular:

– $\mathcal{B}^i$ is the initial state of the planning task i is responsible for; such a state is a set of atoms possibly occurring in $\mathcal{U}$, but also occurring in a private set of atoms describing the internal state of i, and hence these atoms are not traced within the SCP system. We only assume that i joins the SCP system iff $\mathcal{S} \cup \mathcal{B}^i \not\models \bot$.
– $\mathcal{G}^i$ is a list of goals the agent has to achieve; each goal can be an atom or a conjunction of atoms in $\mathcal{U}$ and possibly in the private set of agent's atoms. Note that, differently from classical planning, it is not required that all the goals in $\mathcal{G}^i$ hold in a unique system state.
– $\mathcal{C}^i$ is initially empty.
– Φ^i is a set of domain-dependent actions agent i can directly perform whenever their preconditions hold. For instance, in a logistic domain, a truck-agent can perform action `drive`, whereas a plane-agent can `fly`.
– Σ^i can be initialized in different ways; in fact, differently from Φ^i, this set needs not to be static; on the contrary, it could change over time according to contextual conditions. In our preliminary implementation, we have adopted a very simple solution. Let us consider the ENTICE rule above[1]. The objective of this rule is to create a commitment of the form $C(i, j, s, u)$, in order to "entice" another agent j to bring about s, which is of interest for i. At this initial stage, however, i cannot know which condition u is of interest for j. Surely enough, i knows which atoms it can directly achieve by performing its physical actions. Thus, for each atom $s \in \mathcal{U}$ such that s never appears as an effect of any action in Φ^i, agent i creates a template `entice-`s whose effect is the creation of a commitment $C(i, \_, s, u)$, where $\_$ denotes any agent willing to satisfy s, and u is any atom in $\mathcal{U}$ that appears in

[1] Other rules are treated consequently.

Algorithm 1 Social Continual Planning Strategy

SCP-Strategy$(\mathcal{B}^i, \mathcal{G}^i, \mathcal{C}^i, Acts^i, Socs^i)$
1. **while** $\mathcal{G}^i \neq \emptyset \vee \mathcal{C}^i \neq \emptyset$ **do**
2. **on** $\mathcal{S}$ **change** update $\mathcal{G}^i$ using $Socs^i$
3. $g \leftarrow$ pick up a goal from $\mathcal{G}^i$
4. $\pi \leftarrow$ plan to g
5. $status \leftarrow$ execute π
6. **if** $status$ equals $success$ **then**
7. $\mathcal{G}^i \leftarrow \mathcal{G}^i \setminus \{g\}$
8. **end if**
9. **end while**

the effects of at least one physical action in Φ^i. Of course, since the `entice-`s template can be instanced in different ways, depending on the actual u condition, agent i will offer first the conditions, that from its point of view, are the cheapest to achieve.

– $Socs^i$ is a static set of rules, decided at design time, that defines the social behavior of i; namely, how an agent is reliable for bringing about the consequent and antecedent conditions of the commitments in $\mathcal{C}^i$.

4.1 Social Continual Planning: the Strategy

Basically, the SCP strategy we propose, sketched in Algorithm 1, is a form of continual planning (see e.g., [5]) in which generative planning is interleaved with plan execution. The main difference with other approaches is that to achieve a goal, an agent plans not only its own actions, but also its engagements with others, and depending on how these interactions carry through, the agent may decide to perform some replanning or to pursue a different goal.

An agent i follows the SCP strategy as far as there are goals in $\mathcal{G}^i$ to be achieved or $\mathcal{C}^i$ is not empty. This second condition assures that an agent does not leave the system when it is still involved in some active commitments.[2] At each iteration, the agent checks for updates in the social state $\mathcal{S}$ (line 2); any change occurring in $\mathcal{S}$, in fact, can have an impact on the set $\mathcal{G}^i$ of goals. For instance, a new commitment $C(j, \_, s, u)$ appearing in $\mathcal{S}^{\mathcal{C}}$ could draw the attention of agent i when u is a condition that i needs but it cannot achieve on its own, and at the same time i knows how to obtain s. In such a case, i could accept to be the *creditor*: s is added to $\mathcal{G}^i$ (i will eventually bring about s). On the other hand, the occurrence of a new atom in $\mathcal{S}^{\mathcal{E}}$ could make the achievement of a goal g in $\mathcal{G}^i$ no longer necessary, so g is dropped. Of course, these agent's decisions are driven by the $Socs^i$ behavioral rules.

[2] In principle, an agent may remaining situated within the system indefinitely, waiting for agents to cooperate with. For example, in a logistic domain, a shipper has the high-level objective of earn money by offering its transportation facilities. This objective does not immediately translate into an initial goal $\mathcal{G}$, but rather it is better modeled in terms of pragmatic rules (i.e., both social actions in Σ, and behavioral rules in $Socs$), so as the shipper is willing to accept requests from other agents, but also offers itself shipment services to others.

Once $\mathcal{G}^i$ has been updated, agent i selects one goal g from $\mathcal{G}^i$ (line 3); and synthesizes a plan π reaching g (line 4). It is worth noting that any off-the-shelf planner can be used to synthesize π since from the point of view of the planner there is no distinction between social and physical actions (both kinds of actions are translated into PDDL, see below). We only assume that in case the used planner produces a partial-order plan (POP), π is one of the possible linearizations of such a POP.

After the planning step, the agent can start the execution of π (line 5), which contains both physical actions in Φ^i, and social actions in Σ^i. The execution of π proceeds one action a at a time and in the order. If a is a physical action, it is immediately executed, and its effects on atoms in $\mathcal{U}$ are made available to all the other agents via $\mathcal{S}^{\mathcal{E}}$. If a is a social action, e.g., an `entice-`s action, the action execution affects $\mathcal{S}^{\mathcal{C}}$ with the addition of a new commitment $C(i, \_, s, u)$, which has to be picked up by some other agent. The execution of π is therefore suspended; indeed, the `entice-`s action is part of π only in case the atom s is a precondition for some subsequent action, and hence the plan execution cannot proceeds without s. In case an agent j is interested in u, it accepts the offers by finalizing the commitment in $C(i, j, s, u)$, and eventually it will bring about s. As soon as s is satisfied, i proceeds with the execution of its plan (u will be added to $\mathcal{G}^i$ the next time i checks for changes in S). When all the actions in π are performed, the execution phase terminates in *success* state (i.e., g has been achieved), and hence g is removed from $\mathcal{G}^{\rangle}$ (line 7).

However, it is also possible that no agent is interested in the service u offered by i. To avoid an indefinite wait, i sets up a timer. As soon as the time runs out, the commitment is canceled from $\mathcal{S}^{\mathcal{C}}$, and the plan execution terminates with a *failure* state. Since g has not been achieved, it is not removed from $\mathcal{G}^i$. At the next iteration of the strategy, i first checks whether g is still required (line 2), and then tries to find an alternative plan reaching it (line 4) that may require a different instantiation for the `entice-`s action (i.e., with a different condition offered as consequent of the commitment).

Intuitively, the correctness of the approach relies on the coherence and convergence properties discussed in [20]. In particular, the goal convergence property states that in the situation in which agent i has a goal G_1=$\mathsf{G}(i, p_1, r_1, q_1, s, f_1)$, another agent j has a goal G_2=$\mathsf{G}(j, p_2, r_2, q_2, s, f_2)$, and there exists a commitment $C_1 = C(i, j, s, u) \in \mathcal{S}^{\mathcal{C}}$, then, there is a finite sequence of pragmatic rules that leads to $G2$'s state equaling G_1's state. This means that whenever agent j brings about s, satisfying its internal goal G_2, then, also agent i has its own goal G_1 indirectly satisfied. This demonstrates the correctness of the SCP strategy in the sense that whenever a plan π, synthesized by i, contains an entice action `entice-`s, which actually creates the commitment C_1, then the plan is:

1. *feasible*: no action a in π has open preconditions (i.e., atoms that are neither provided by the initial state nor by any previous action); this implies that the preconditions that agent i cannot directly produce, are obtained via cooperation with others;
2. *correct*: if each action is performed successfully, g holds in $\mathcal{S}^{\mathcal{E}}$ at the end of π; as noted above, the execution of social action implies the cooperation with other agents.

5 Implementation and Proof of Concept

To verify the feasibility of the SCP strategy, we implemented a proof-of-concept in C and SICStus Prolog 4.3.2. More precisely, an agent is implemented as an independent C program that embeds a simple Prolog planner. Each agent takes in input three files:

1. the description of the planning domain (in PDDL 2.2) from the point of view of a single agent, e.g., the templates of the physical actions the agent is capable to perform;
2. the definition of the planning problem (in PDDL 2.2) that this specific agent has to accomplish;
3. the list of environment objects, namely, those objects that constitute the ontological backbone shared by all the agents in the SCP system. All the possible atoms about these objects must be known by the all the agents.

Note that the parsing of the three files creates a number of initial structures within the Prolog planner, as for instance the list of action templates. This list is subsequently completed with a list of social actions (i.e., actions in Σ), following the procedure sketched above. This step consists in instantiating the predefined PDDL templates encoding the pragmatic rules from goals to commitments. For instance, a number of entice actions are created by instantiating the following template:

```
(:action entice
  :parameters (?deb ?cre - agent
               ?ant ?cons - goal)
  :precondition (and
    (active-G ?deb ?ant)
    (not (achieved ?ant))
    (cando ?deb ?cons)
    (null-C ?deb ?cre ?ant ?cons)
   )
  :effect (and
    (not (null-C ?deb ?cre ?ant ?cons))
    (active-C ?deb ?cre ?ant ?cons)
    (increase (plancost) 10)
   )))
```

where a dummy symbol is used to refer to the creditor agent `?cre`, not known at this time. `active-G` and `null-C` are terms used to define goals and commitments, respectively, with their current state. `?ant` and `?cons` goals are unique identifiers associated to atoms such as `at(pkg1, E)`. These identifiers are therefore shared by the agents. This workaround was necessary to overcome the current limits of the PDDL expressiveness for which nested terms are not admissible. This instantiation is realized as a C procedure. The generated instances are injected within the Prolog planner by using the C/SICStus bi-directional interface. Finally, note that the entice action is associated with a cost, this is used to avoid the planner resorts to them even when they are not strictly required (e.g., a "lazy" agent could ask others to obtain goals it can achieve by itself).

The social state is implemented by relying on the Inter-Process Communication facilities offered by the Unix operating system. In particular, two segments of shared memory are used, one for $\mathcal{S}^{\mathcal{C}}$ and one for $\mathcal{S}^{\mathcal{E}}$. Initially both segments are empty, but active processes take turns filling up the $\mathcal{S}^{\mathcal{E}}$ segment by publishing the public facts they know about the environment objects (Unix semaphores are used to guarantee the consistent access to the shared memory segments). If a contradiction arises at this step, the agent terminates without joining the SCP system (i.e., without attempting to solve its task). So far, behavioral rules that specify the social strategy $Socs$ are implemented directly in C. Indeed, only a subset of the rules discussed in [20] have been implemented. For simplicity, we have implemented only progressive rules that any "honest" agent should follow; so our agents never cancel a commitment, and always pursue the goals appearing in the antecedent and consequent conditions of commitments in which they are involved.

After this preliminary steps, each agent can start solving its task by invoking the embedded planner. Note how the facts maintained by the planner correspond to the private belief state $\mathcal{B}$, whereas facts that are maintained in the shared memories correspond to the social state. (For efficiency reasons, though, each planner replicates in its own working memory the facts in the social state.)

5.1 Example: a Logistic Domain

Let us exemplify the SCP strategy via a simple problem from the well-known logistic planning domain. Figure 3 shows the current state of the SCP system at hand: `A` through `F` are cities; `C` and `D` have airports, and plane $pln1$ can fly between them; trucks $trk1$ and $trk2$ can drive along the solid edges connecting the cities.

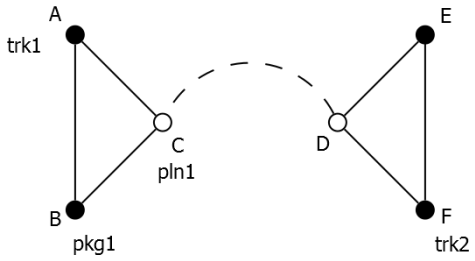


Fig. 3. Initial state of the logistic example.

In such a scenario, the only shared object in the environment is represented by the package $pkg1$, currently located at `B`. Thus, $\mathcal{U}$ contains all the atoms about such an object, which in the logistic domain concern its position, specifically: $\mathcal{U}$=$\{at(pkg1, A)$, $at(pkg1, B)$, ..., $at(pkg1, E)$, $at(pkg1, trk1)$, $at(pkg1, trk2)$,$at(pkg1, pln1)\}$.The set of agents is therefore $\mathcal{A}$=$\{trk1$, $trk2$, $pln1\}$; whereas, for the sake of simplicity, we suppose the social state $\mathcal{S}$ is currently empty.

Note that, since $\mathcal{U}$ refers only to the package position, the position of each agent is a private piece of information. Moreover, also the domain knowledge is partitioned: $trk1$

Table 1. The first four steps of the SCP strategy

steps	goal	plan	social state
1	tkr1: at(pkg1, E)	trk1: drive(A,B); load(pkg1, B); entice-at(trk1, _, at(pkg1, E), $100)	$\mathcal{S}^{\mathcal{E}}$:{at(pkg1,trk1)}, $\mathcal{S}^{\mathcal{C}}$:{$C$(trk1, _, at(pkg1,E), $100)}

2	trk2: at(pkg1, E)	trk2: drive(F, D);
		entice-at(trk2, _, at(pkg1, D), at(pkg1, E));
		load(pkg1, D); drive(D, E);
		unload(pkg1, E)
			$\mathcal{S}^{\mathcal{E}}$:{at(pkg1,trk1)},
			$\mathcal{S}^{\mathcal{C}}$:{$C$(trk1, trk2, at(pkg1,E), $100),
			C(trk2, _, at(pkg1, D), at(pkg1, E))}

3	pln1: at(pkg1, D)	pln1:
		entice-at(pln1, _, at(pkg1, C), at(pkg1, D));
		load(pkg1, D);
		fly(C, D); unload(pkg1, C)
			$\mathcal{S}^{\mathcal{E}}$:{at(pkg1,trk1)},
			$\mathcal{S}^{\mathcal{C}}$:{$C$(trk1, trk2, at(pkg1,E), $100),
			C(trk2, pln1, at(pkg1, D), at(pkg1, E)),
			C(pln1, _, at(pkg1, C), at(pkg1, D))}

4	trk1: at(pkg1, C)	drive(B,C); unload(pkg1)	$\mathcal{S}^{\mathcal{E}}$:{at(pkg1,C)}, $\mathcal{S}^{\mathcal{E}}$:{$C$(trk1, trk2, at(pkg1,E), $100), C(trk2, pln1, at(pkg1, D), at(pkg1, E)), C(pln1, trk1, at(pkg1, C), at(pkg1, D))}
....	...	...	...

just needs to know how to move among A, B, and C, whereas it needs not to know how the other cities are connected.

Let us consider the problem from the point of view of truck $trk1$. Its local planning task consists in delivering package $pkg1$ to E. This is formalized as follows:

- $\mathcal{B}^{trk1}$=$\{at(pkg1, B), at(trk1, A)\}$,
- $\mathcal{G}^{trk1}$=$\{at(pkg1, E)\}$,
- $\mathcal{C}^{trk1}$=$\emptyset$,
- Φ^{trk1}=$\{drive(A, B), drive(A, C), \ldots, load(pkg1, A), unload(pkg1, A), \ldots\}$
- Σ^{trk1}=$\{entice - at(trk1, \_, at(pkg1, E), \$100)$, $entice{-}at(trk1, \_, at(pkg1, D), \$100), entice{-}at(trk1, \_, at(pkg1, F), \$100), entice{-}at(trk1, \_, at(pkg1, E), \$1000), \ldots\}$

Note that Φ^{trk1} contains those actions that are typically defined in the logistic domain for a truck whereas Σ^{trk1} contains all the social actions agent $trk1$ is willing to use during its planning search. In particular, there are more entice actions sharing the same antecedent but differing in the consequent. This is the result of the instantiation procedure of the entice template. For instance, $trk1$ can either pay $100 or $1000 to have $pkg1$ at E. Of course, a proper usage of action costs would drive the planner in creating "cheaper" commitments first.

Agent $trk1$ starts the solution of its local planning task by finding a plan reaching the goal $at(pkg1, E)$. The execution of such a plan starts a course of engagements that will involve all the agents. Table 3 summarizes the first four runs of the SCP strategy followed by the agents. Each row in the table shows which goal a specific agent is pursuing, the plan that has been synthesized by the agent, and how the execution of the plan (until the first social action) changes the social state.

The first row of the table shows the plan inferred by $trk1$: after having picked up package $pkg1$, the agent, that cannot physically deliver it to E, offers $ 100 to any agent

willing to take $pkg1$ to E. Agent $trk2$ accepts the offer, see row 2, and plans how to achieve the goal. Also the $trk2$'s plan contains an offer towards any other agent which is willing to take $pkg1$ to D, which is accepted (see row 3) by agent $pln1$. To keep the discussion simple we assume that the agents are cooperative, and omit the fact that $pln1$ could ask $trk2$ to do something specific in exchange (e.g., pay for the shipment service). In row 4 we have that agent $trk1$ brings $pkg1$ to C where $pln1$ is waiting for loading and flying it to D, from which $trk2$ will take it to E. Note that the execution of the agents' plans will progressively satisfy goals and commitments that will be removed from the social state.

5.2 First Experiments

We have used the logistic domain as a test-bed for a preliminary experimental analysis. At this stage of development our main objective was to study the feasibility of the approach, and to highlight possible bottlenecks and shortcomings of the SCP strategy, in particular as concerns the instantiation of the entice action.

In a domain involving 2 trucks and 2 planes, we prepared 10 problems. In each problem, every agent was assigned with one package to be delivered. In all the cases, the agent could not deliver the package without the help of at least one other agent. Although the implementation is yet to be engineered, the first data we collected are very encouraging. On average, the instantiation of the entice action produces 68 instances for each agent. Such a large number of action could represent a burden for the planner, but in practice the planner we used (implementing a simple A* search) worked very efficiently; in fact, the planning times are on average 1500 ms, with a pick to 8000 ms only in one case. The length of the synthesized plans is 15 actions, on average, with a pick to 24.

These results show that, even though the instantiation of the entice action can produce a relatively large number of actions, the planner is not significantly burdened by them as in general only a few of them are applicable at the same time. In other words, the number of entice instances is not directly related to the search branching factor.

6 Discussion and Conclusions

In this paper we addressed the SCP problem, and proposed the SCP strategy as a possible solution. Differently from MAP approaches, where a predefined team of agents has to find a *joint* plan solving a given planning task, here we deal with situations in which each agent is given a planning task which is independent of the others' ones. The challenge, thus, is not to find a joint plan, but to find a plan for each agent that solves the agent's planning task taking advantage of the cooperation with other agents. Moreover, agents are free to join and leave the system dynamically.

The novelties of our proposal are not limited to the openness of the agent team. While in approaches to MAP agents can be thought of as resources used for solving the given planning task, in SCP agents are seen as *autonomous* entities. This change implies that an agent cannot order another agent to do a job, but the agent can just make an offer, and as we have seen, social commitments come at handy to model this kind

of relations. More importantly, however, we have to observe that an agent receiving an offer, being an autonomous entity, can accept or reject the offer depending on its contextual conditions and its local goals. A rational agent, in fact, should accept an offer only if the offer brings along some advantages, otherwise the offer should be put aside.

It is worth noting how the SCP strategy supports the *decoupling* of agents, that just share environment objects, whereas they are independent for all the other respects. In particular, each agent can implement its social strategy (i.e., pragmatic rules in $Socs$ and Σ) according to local criteria. Moreover, the planning algorithm each agent uses can be tailored to meet optimization functions that are relevant for the agent itself. Note also how the cooperation among the agents do not require that an agent knows the action templates of others (as for instance happens in [15]), and, hence, also the agents' privacy is preserved.

Many lines of research and improvement are possible. In the near future we aim at engineering the implementation of the SCP strategy by exploiting one of the many agents platforms available. In particular, the JaCaMo+ platform [1] seems to be a good candidate since it naturally supports the notions of commitments and social states. In addition, the social behavioral rules in $Socs$ could find an easy implementation as Jason plans (used to program JaCaMo+ agents). Also the integration with a planner does not seem to raise to much troubles; as demonstrated in [12] where Jason plans have been integrated with generative planning.

Acknowledgments

We would like to thank Lorenzo Pierini for his contribution to the implementation.

References

1. Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Programming with Commitments and Goals in JaCaMo+. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*, pages 1705–1706. International Foundation for Autonomous Agents and Multiagent Systems, 2015.
2. C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.
3. Ronen I. Brafman. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of Distributed and Multi-Agent Planning Workshop ICAPS 2015*, pages 1–8, 2015.
4. Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pages 28–35, 2008.
5. Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.
6. Pieter Buzing, Adriaan Ter Mors, Jeroen Valk, and Cees Witteveen. Coordinating self-interested planning agents. *Autonomous Agents and Multi-Agent Systems*, 12(2):199–218, 2006.

7. Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous norm acceptance. In *Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th International Workshop, ATAL '98, Paris, France, July 4-7, 1998, Proceedings*, pages 99–112, 1998.
8. Jeffrey S Cox and Edmund H Durfee. Discovering and exploiting synergy between hierarchical planning agents. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 281–288. ACM, 2003.
9. Jeffrey S Cox, Edmund H Durfee, and Thomas Bartold. A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 821–827. ACM, 2005.
10. Edmund H Durfee and Victor R Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(5):1167–1183, 1991.
11. Victor Lesser, Keith Decker, Thomas Wagner, Norman Carver, Alan Garvey, Bryan Horling, Daniel Neiman, Rodion Podorozhny, M Nagendra Prasad, Anita Raja, et al. Evolution of the gpgp/taems domain-independent coordination framework. *Autonomous agents and multi-agent systems*, 9(1-2):87–143, 2004.
12. Felipe R. Meneguzzi and Michael Luck. Leveraging new plans in agentspeak(pl). In *Declarative Agent Languages and Technologies VI, 6th Int. Workshop, DALT 2008, Revised Selected and Invited Papers*, volume 5397 of *Lecture Notes in Computer Science*, pages 111–127. Springer, 2008.
13. Felipe R. Meneguzzi, Pankaj R. Telang, and Munindar P. Singh. A first-order formalization of commitments and goals for planning. In *Proc. of the 27th AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.
14. Felipe R. Meneguzzi, Pankaj R. Telang, and Neil Yorke-Smith. Towards planning uncertain commitment protocols. In *Proc. of the 2015 Int. Conf. on Autonomous Agents and Multiagent Systems, AAMAS*, pages 1681–1682. ACM, 2015.
15. Raz Nissim and Ronen I. Brafman. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research (JAIR)*, 51:293–332, 2014.
16. Munindar P. Singh. An ontology for commitments in multiagent systems. *Journal of Artificial Intelligence in Law*, 7(1):97–113, 1999.
17. Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.
18. Munindar P. Singh. Commitments in multiagent systems some controversies, some prospects. In *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, chapter 31, pages 601–626. College Publications, London, 2011.
19. Pankaj R. Telang, Felipe R. Meneguzzi, and Munindar P. Singh. Hierarchical planning about goals and commitments. In *Int. conf. on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, pages 877–884. IFAAMAS, 2013.
20. Pankaj R. Telang, Munindar P. Singh, and Neil Yorke-Smith. Relating Goal and Commitment Semantics. In *Post-proc. of ProMAS*, volume 7217 of *LNCS*. Springer, 2011.
21. Adriaan ter Mors, Chetan Yadati, Cees Witteveen, and Yingqian Zhang. Coordination by design and the price of autonomy. *Autonomous agents and multi-agent systems*, 20(3):308–341, 2010.
22. Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In Dieter Fensel, Fausto Giunchiglia, Deborah L. Mc Guinness, and Mary-Anne Williams, editors, *Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 470–481. Morgan Kaufmann, 2002.