KI 2013
FROM RESEARCH TO INNOVATION AND PRACTICAL APPLICATIONS
36TH GERMAN CONFERENCE ON ARTIFICIAL INTELLIGENCE
16TH-20TH SEPTEMBER 2013: KOBLENZ

27. PuK-Workshop: Planung/Scheduling und Konfigurieren/Entwerfen

http://www.puk-workshop.de

Stefan Edelkamp[1], Jürgen Sauer[2], Bernd Schattenberg[3] (eds.)

1 Universität Bremen
Technologie-Zentrum Informatik
D-28357 Bremen
edelkamp@tzi.de

2 Universität Oldenburg
Department für Informatik
D-26111 Oldenburg
juergen.sauer@uni-oldenburg.de

3 Universität Ulm
Institut für Künstliche Intelligenz
D-89069 Ulm
bernd.schattenberg@uni-ulm.de

Koblenz, September 2013

CALL FOR PAPERS

KI 2013 Workshop

27. Workshop Planen/ Scheduling und Konfigurieren / Entwerfen

16. September 2013
Koblenz

The PuK workshop is the regular meeting of the special interest group on planning, scheduling, design and configuration within the AI section of the GI. It has an amazing long tradition and goes into its 27th edition.
As in previous years the PuK workshop brings together researchers and practitioners of the areas of planning, scheduling, design and configuration. It provides a forum for the exchange of ideas, evaluations and experiences especially in the use of AI techniques within these application and research areas.

Topics of Interest

The general topics of interest of the PuK community include but are not limited to:
* Practical applications of configuration, planning or scheduling systems
* Architectures for planning, scheduling or configuration systems
* Knowledge representation and problem solving techniques, e.g.,
    - domain specific techniques:
    - heuristic techniques
    - distributed problem solving
    - constraint-based techniques
    - iterative improvement
    - integrating reaction and
    - user-interaction.
* Learning in the context of planning, scheduling and design.

Special Focus: System View

As we have done in earlier workshops, we intend to focus on a specific area of interest: This time it is the systemic view on the use of the tools and techniques developed in the area of planning, scheduling and configuration. The complete systems and application areas in which the approaches are used shall be in the focus of the interest. This focus shall also help to attract the workshop to practitioners in the field, who are invited to present practical problems and to discuss their experiences, concepts, and ideas. It is also intended to stimulate a mutual exchange with the researchers on our common field's future directions. Thus, a second main goal of this part of the workshop is the support of research planning. Besides this, further submissions from the general topics mentioned above are welcome.

Submission and Contribution Format

The workshop will be held in English. Papers are subject to regular peer review and subsequent publication within the workshop proceedings (4-8 pages).

Submission is by eMail to one of the organizers.

Important Dates

17.06.2013 Papers/ Abstracts Due Date
24.06.2013 Notification of Acceptance
01.07.2013 Submission of Camera-Ready Papers (PDF)

Program Committee

Susanne Biundo, Universität Ulm, D
Stefan Edelkamp, University of Bremen, D
Malte Helmert, Universität Basel, CH
Joerg Hoffmann, Universität Saarbrücken, D
Joachim Hertzberg, Universität Osnabrück, D
Lothar Hotz, University of Hamburg, D
Ulrich John, SIR Plan GmbH, Berlin. D
Peter Kissmann, Universität Saarbrücken, D
Thorsten Krebs, Encoway, Bremen, D
Jürgen Sauer, University of Oldenburg, D
Bernd Schattenberg, University of Ulm, D
René Schumann, University of Valais, CH

Organizers and Contact

Prof. Dr. Stefan Edelkamp
Technologie-Zentrum Informatik und Informationstechnik
Universität Bremen
e-mail: edelkamp@tzi.de

Prof. Dr.-Ing. Jürgen Sauer
Universität Oldenburg
Fak II, Department Informatik
e-mail: juergen.sauer@uni-oldenburg.de

Dr. Bernd Schattenberg
Universität Ulm
Institut für Künstliche Intelligenz
e-mail: bernd.schattenberg@uni-ulm.de

Programm

Title	*Authors*
Experience Based Task and Environment Specification for Plan-based Robotic Agents	Hartmut Messerschmidt, Jan Winkler, Asil Kaan Bozcuoglu, and Michael Beetz
Planning for Multiple Robots in a Smart Environment	Maurizio Di Rocco, Federico Pecora, Alessandro Manzi, Raffaele Limosani,Giancarlo Teti, Filippo Cavallo, Paolo Dario, and Alessandro Saffiotti
Search Strategies for Partial-Order Causal-Link Planning with Preferences	Pascal Bercher, Fabian Ginter, Susanne Biundo
PUK goes Industry	Bernd Schattenberg
Agent-based Multimodal Transport Planning in Dynamic Environments [KI-Paper]	Christoph Greulich, Stefan Edelkamp, Max Gath
Gamer -- Past, Present, Future Symbolic Search in Planning" [Invited Talk]	Peter Kissmann
Combining Symbolic with Bitvector Graph Search for Solving Connect Four	Stefan Edelkamp, Martha Rohte, Peter Kissmann
Power plant scheduling at long term scenarios with GAMS	Maik Günther
Decentralized, cooperative agv control – first results	Christoph Schwarz
Scheduling 4Green	Jürgen Sauer

Experience Based Task and Environment Specification for Plan-based Robotic Agents

Hartmut Messerschmidt, Jan Winkler, Asil Kaan Bozcuoğlu, and Michael Beetz

Universität Bremen, 28359 Bremen, Germany

Abstract. Planning for autonomous robots differs from classical planning, because actions are not behaving deterministically. In a real world environment there are too many parameters and too many cases to handle before the plan execution. Nevertheless planning is needed to give robots the flexibility to work in unknown or not completely known environments. Plan execution can be seen as a flexible way of handling unknown and/or changing environments. Here, during runtime, sensors and reasoners can be asked to give necessary information. To cover the uncertainty in the real world, there is need for more than pure logical reasoning . In order to close this gap, we want to extend the CRAM Plan Language (CPL) so that it can ask probabilistic world models which is the best way to continue under the current conditions and beliefs. These world models should be ultimately learned and trained autonomously by the robot. For this the robot needs to log data in an appropriate way as well as use this data to infer a world model and train it. But it is not feasible to learn one model of the whole world. It looks much more promising to use a combined approach of defining and learning a hierarchical model of the world. To start with small parts during a plan execution and learning methods to increase the success rate for these parts can be a starting point. In this paper a first proof of concept of how such a model can automatically be found and learned is presented.

1 Introduction

A fundamental problem during plan execution is a strict parameterization even before the execution starts [5]. Thus, such plans can fail to accomplish goals if any of the used assumptions change even marginally. Therefore today's cognition-enabled robotic agents are expected to competently and autonomously perform complex tasks with as little information as possible. The information given by humans is mostly partial and vague, leaving room for interpretation and misunderstanding. As robotic agents cannot fall back to common-sense knowledge, they must be equipped with extensive reasoning mechanisms to interpret the input data and act accordingly. I.e., they need to find the correct parameters for a task or give up as the task description is too ambiguous and there are no knowledge sources to reduce its ambiguity.

For example, consider a *fetch and place* scenario, where the robot needs to find, grasp and bring an object to a given location. The task seems rather easy at first; take a specified object and place it at a specified location. But there are many unknown parts before and during execution of the plan. Depending on the object its most probable location in the environment has to be determined, after the object is found it has to be picked up, so some reasoning about possible grasping points has to be done and so on and so forth. The task description might be something like "Place my cup on the table". The first step is then to get knowledge about the user issuing this task, to find out to whom the cup belongs, and finally where it is. After finding the cup it has to be picked up. For this a good place to stand and a grasping point on the cup have to be found.

CRAM[1] offers a solution to some of these issues by introducing vague parameter descriptions in the form of *designators*. The description might change over time (when the world changes or more knowledge is discovered), and so the designator carries the current belief about its assigned object. In the CRAM framework, the plans consist of an ultimate goal, high-level subtasks with their own subgoals, and low-level predicates for describing the environmental properties and self-information such as joint angles. In this paper, we propose an integrated learning scheme for CRAM that even goes beyond this approach by gaining “experiences” from previous plan executions. By doing that, we aim at a more adaptive and less parametrized, i.e., less environment specific, plan execution in the future. In other words, we want the robot to infer environment-specific parameters by using the perception data, the learned model, knowledge bases, and the supplied plans to include only high-level information about the task at hand.

In order to achieve this, we log the low-level predicates (i.e., events and plan decisions) during the execution. These predicates, as the feature set for our learning scheme, describe the respective state of the environment and the performance of the robot when executing any given task. For learning we label these samples as positive or negative samples either manually or by getting the result of the corresponding task from CRAM.

An additional advantage of our scheme is the integration with the powerful robotic knowledge tool KNOWROB[2], which supplies us with high-level information such as which objects are needed for a particular task or which results should be expected after executing an action.

In the *fetch and place* example, the CRAM system can query the knowledge base or the learned probabilistic model to improve its plan execution success rate. That reasoning is a main part in plan execution is shown in Figure 1.

This paper is devided into two parts: In Sections 2 and 3 the CRAM and KNOWROB systems are explained, respectively. Then an idea on how to extend this current structure with learning from experience, that is logged data, is given. The second part of this paper shows in an example that automatic preprocessing and learning from logged data works and how it can be done. It closes with a conclusion and an outline of further steps to include this mechanisms into the CRAM system.

2 Robot Plan Language in the CRAM System

The CRAM plan execution engine is being developed as an abstraction layer between generic, parameterizable high-level plans, and robot-specific driver modules. One of its goals is to supply comprehensive language features for defining these high-level plans, transparently incorporating capabilities from external components such as perception or knowledge processing systems.

[1] Cognitive Robot Abstract Machine, `www.github.com/cram-code`
[2] `www.knowrob.org`

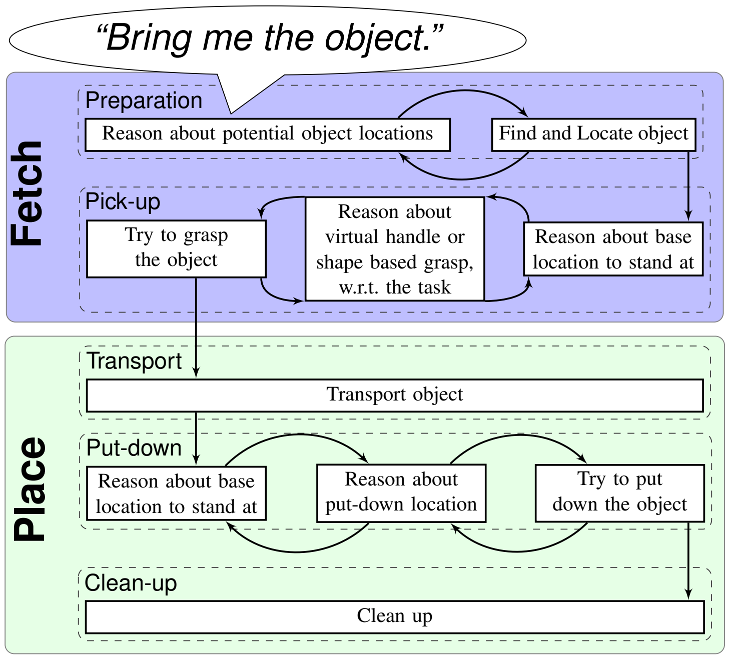

"Bring me the object."
Fetch
Preparation
Reason about potential object locations
Find and Locate object
Pick-up
Try to grasp the object
Reason about virtual handle or shape based grasp, w.r.t. the task
Reason about base location to stand at
Place
Transport
Transport object
Put-down
Reason about base location to stand at
Reason about put-down location
Try to put down the object
Clean-up
Clean up

Fig. 1. Flow of control when executing a generalized *fetch and place* plan. The fetch and place plans are separated and show the five different phases during plan execution.

2.1 Plan Execution with Vague Task Descriptions

In the CRAM system, two major factors are taken into account when trying to execute any given task. First, all plans consist of plan primitives, i.e. goals, from a common plan library. These goals include for example perceiving objects (`perceive-object`), and grasping an object (`object-in-hand`). As these goals picture very complex behavior, their exact execution strongly depends on the objects acted on, the environment, and the given task. This leads to the second major factor, i.e. vague information, which we introduce as the designator concept [3] [4] [1]. Designators are descriptive data structures that transport partial, or complete information about objects, tasks, and locations. In the case of the goals introduced earlier, this concept can be applied for perceiving unknown objects as `(perceive-object (an object ((on table) (color red))))` which will then detect all red objects on a given table. Here, the `perceive` goal interprets the given description and triggers a perception system based on the parameters inferred from the object description. The same concept applies to grasping and placing objects, taking mass (`(an object ((mass 2.0)))` $\rightarrow$ its heavy, use both arms to grasp) or semantic grasping information (`(an object ((handle ...)))` $\rightarrow$ grasp at handle) into account.

2.2 Reasoning about Plan Parameters

When inferring the missing parameters for any task to perform, two knowledge sources are currently queried. One is the built-in situational intelligence inside the CRAM system, defining standard behavior for all plan primitives in the plan library. These include what steps need to be taken when grasping, which perception module to trigger, and when to give up on a task due to frustration limits (retry counts or serious failures such as losing an object from the gripper). This also includes robot-architecture dependent behavior, such as correct grasping poses for a robot-specific gripper or the correct cartesian margins for navigation to not hit furniture in the environment.

The second source of information is an external knowledge base, which holds semantic and task-specific information about objects. Information stored here are CAD models for identification and pose estimation, semantic handle information, the dominant color of the object, or functional properties, such as the mass, or parts of the object to avoid when touching it.

A third very promising channel for behavioral information and decision triggering is the machine learning approach. As most decisions in CRAM are based on the resolution of Prolog patterns and have a set of parameters that must be interpreted, trained decisions based on the current set of parameters can be treated just as the introduced information sources and integrated in the same way.

3 KnowRob: A Knowledge Processing and Inference Tool for Robots

As robots are envisioned to take a role in everyday life as we, human actors do, they must process the knowledge flow of the outer world and reason about it using this flow so that they can accomplish complex tasks in an adaptive manner. To gain processing and reasoning abilities, a robot should be able to match the knowledge with its inner symbolic representations. In [7], Tenorth et al. proposed the KnowRob tool to address this need.

KnowRob is based on SWI Prolog and its API for semantic web which is used for accessing the Web Ontology Language-based ontologies using Prolog. The knowledge bases consisting of these ontologies can either be queried by the robot control program or by the user via the command line input.

The CRAM system exploits KnowRob as a source of knowledge. In the *fetch and place* scenario it could ask whether there is an instance of a cup that belongs to the human who issued the command. When there is such an instance, the designator for this cup is updated such that it contains the information from KnowRob. This might be a description, a point-cloud or object size. To find this cup, KnowRob can have information about probable places to find a cup, like a cupboard or the dishwasher, and places for this specific cup like the office desk of the human.

4 Using Probabilistic Models to Improve Planning Success

The example task *fetch and place* cannot be solved solely by logical information sources. The best location to stand while grasping a cup or the best grasping point can be saved in the knowledge base, but even small changes in the environment will then lead to a re-computation of the location to grasp and for each cup grasping points have to be predefined.

A more general way is to use machine learning to learn probabilistic models to give good estimates of grasping positions or grasping points. In this way the robot can get more experienced after several attempts of performing a task. This "experience" is given by logging data while performing a task, and aligning this data with the belief state of the CRAM system. This labeled data is used to learn and train a world model.

This world model is hybrid, because it consists of discrete values like the belief state of the CRAM system, the task the robot is currently performing, and continuous parts like the robots joint configuration over time. While the discrete parts can be seen as contexts, the continuous part is used to learn a probabilistic model w.r.t. this contexts.

5 Description of the Use-Case: Pancake Baking

In the second part of the paper we give an example use-case which illustrates how the proposed methodology works. We concentrate on the learning from logged data and the preprocessing needed to get good results. For this purpose, we choose pancake making as a more simple scenario. In the task of pancake making we concentrate on the subtask *flipping a pancake with a spatula*, which is fundamental in baking a pancake.

In this example, we collect the training data for the proposed learning scheme using the physics simulation Gazebo and a computer game environment in which human testers do various flipping actions with a Hydra joystick. The pancake dough is represented by small spheres and these spheres are glued together (determined by a signal from the human) after the dough was poured on the oven. After this the tester has to take the virtual spatula and use it to flip the pancake around.

6 From Plan Logging to Training Data

Within CRAM and KNOWROB learning and reasoning with symbolic data is already done. On the other hand, continuous sensor data offers much more learnable data, but it might be too much to learn anything useful from it.

Our approach is to use the symbolic data available to annotate the continuous data. From the task at hand, e.g., flipping a pancake, it follows that the pancake has to be moved and that a tool, e.g., a spatula, is needed for this.

The recorded data is in the form of a trajectory. For each timestep the position and orientation of the tool is recorded. When dealing with different time steps, a normal procedure is to align them using dynamic time warping [2]. During this, two different trajectories are considered similar when they have performed roughly the same movement, even if one was slower or started at a different time. Unfortunately, this method cannot be used in our case, because the success of pushing the spatula under the pancake depends on the speed of the movement. Performing this task too slow pushes the pancake off the oven.

But without this alignment, it is difficult to learn anything from these trajectories. Another way are Hidden Markov Models to couple time steps together [6]. Our solution is to take small subsets of the trajectory and learn these critical parts of the task. This approach is more accurate than Hidden Markov Models. A manual selection of time spans makes the problem easier, but in contrast to automatic selection doesn't scale well and is task dependent. To automatically find critical subtasks or time spans we can use knowledge that is already being inferred during plan execution. With the robotic knowledge tool KNOWROB the objects considered relevant for this task can be inferred. For flipping the pancake it is the pancake itself and the spatula. The robot needs to detect these objects in order to perform the flipping, so its locations over time can be logged. The relevant objects given by the knowledge base can always be logged, in any plan execution.

When we consider the distance between the relevant objects and the (relative) movements of pairs of relevant objects, we can detect periods in the task execution. A period is for example a time frame where the relative movement of two objects stays the same. These slices are not manually assigned and this method works for any tasks, where we have relevant objects, either beforehand or by a reasoning mechanism started from the CRAM system to infer the best plan.

The sequence of subtasks can then be (manually) annotated or critical subtasks can be learned.

In some tasks even the outcome of the subtasks can be determined automatically: A flipping must contain a movement of the pancake and in order to get the upper side down it has to be lifted. We can take the positions and orientations of the spatula and the pancake and derive from that if the lifting of the pancake was successful.

7 Evaluation

As mentioned in Chapter 5, we tested the proposed learned scheme in the physics simulation Gazebo and a computer game environment in which human testers do various flipping actions with a Hydra joystick. In this environment, the pancake was modeled as a circular colony of 80 small spheres.

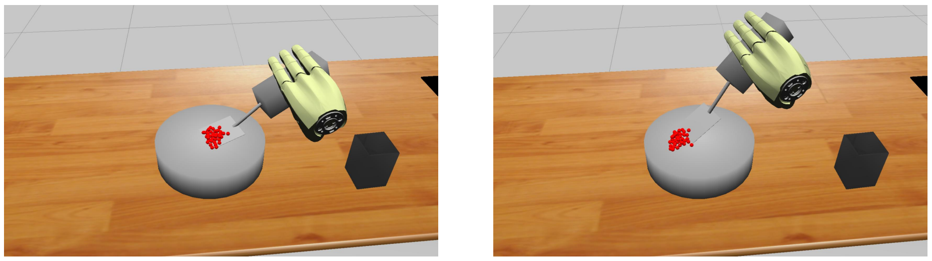


Fig. 2. Successful (left) and unsuccessful (right) attempts of pushing the spatula under the pancake.

In the experiments, the human tester made ten flipping actions out of which five were successful while the rest failed for various reasons (Figure 2). During each try, the pose of the spatula and all spheres involved in the pancake model were recorded with a frequency of 10 Hz, for a duration of about 40 seconds. Prior to learning, we process this information and compute the mean of the spheres' positions (which corresponds to the position of the pancake), the velocity of the spatula, the mean velocity of spheres (which corresponds to the velocity of the pancake) and the relative speed between the spatula and the pancake. In the end, we have the spatula's position and velocity, the pancake's position and velocity, and the relative velocity between two objects as the feature set for each time frame (Table 1).

Table 1. Features created and used in learning part. Each feature consists of three coordinates.

Feature Name
Spatula position
Mean of sphere positions
Relative speed of spatula w.r.t. pancake
Spatula velocity
Mean of sphere velocity

Now there are two ways to learn from these examples: Either each time step is considered as a single instance, or the action sequence is taken into account as a whole. For the second case, as time warping is not an option as explained above, each action sequence is truncated to the minimal number of time steps such that each sample has the same length. For longer time lines, the success value of some sequences would follow immediately from the height (z-coordinate) of the pancake.

In both cases, a whole sequence was used either completely as training or test data. This is to overcome the overfitting by *"learning"* single time lines. K-Nearest Neighbor(K-NN) would then be as good as any Machine Learning approach.

8 Results

In this section, we first give the learning results of the case that each action (time line) is taken as one instance. Then, we give the results of the second case in which each time step is a separate instance. For the first case, we use K-nearest neighbor algorithm with 3-fold cross validation for assessing the learning performance.

The prediction rate of the first case is given in Table 8. In this case, we use ReliefF feature selection prior to learning to analyze how different number of features will effect the performance. ReliefF assigns a relevancy weight for each feature. By assigning different relevancy thresholds, we train K-NN with different number of features. When we do not give any threshold, we have relatively low learning performance due to the curse of dimensionality. On the other hand, if we set the threshold too high, we do not have enough features to describe the whole situation. This also causes a decrease in

Table 2. Learning performance of the proposed scheme in the experiments for the first case in which we use whole sequences of a flipping action together with different ReliefF thresholds.

Relevancy Threshold	Number of Features	Precision	Recall
None	210	0.708	0.7
0	110	0.917	0.9
0.3	16	0.904	0.9
0.48	5	0.833	0.8

learning. For only ten instances other machine learning approaches are not considered useful.

Finally, in the second case we have used Bayes Nets, Decision Trees (J48) and K-Nearest Neigbor (KStar). For 80 percent training data the results are really good, even perfect for decision trees. This is misleading, because there is only one successful and one failed sequence for testing, even though there are more than 500 instances in the test set. Therefore we also used two sequences for training and eight for testing. The results are shown in Table 8. With more test data and less training data K-NN is outperformed by Bayes Nets and Decision Trees.

Table 3. Learning performance of the proposed scheme in the experiments when each time step of a flipping action is a seperate instance

Learning Algorithm	Split ratio (Train/ Test)	Precision	Recall
K-NN	80%/20%	0.99	0.99
Bayes Net	80%/20%	0.985	0.984
Decision Tree	80%/20%	1	1
K-NN	20%/80%	0.844	0.801
Bayes Net	20%/80%	0.945	0.946
Decision Tree	20%/80%	0.909	0.864

9 Conclusion and Future Work

We successfully used automatically segmented time steps to learn Bayes Nets and Decision Trees with high precision and recall. In the future we want to use methods from this proof of concept to learn from CRAM log data and incooperate the results into the plan execution. We want to learn two things for subtasks: The possible outcome, hopefully even before the task is nearing its end, and structural features of these subtasks. The latter to alert the system when crucial parts are missing, or are performed in a wrong ordering, according to our experience. With more logging and annotation by the knowledge base, the CRAM belief state and the automatic selection of time spans, we want to construct an automaton, such that each state of this automaton is a probabilistic reasoning mechanism and the time spans and the context determine, which state to use and how to change the current state.

10 Acknowledgements

This work has been supported by the DFG Project BayCogRob within the DFG Priority Programme 1527 for Autonomous Learning. We would especially like to thank Andrei Haidu for collecting experimental data of pancake flipping in a simulated environment.

References

1. M. Beetz. *Anticipating and Forestalling Execution Failures in Structured Reactive Plans*. Technical report, yale/dcs/rr1097, Yale University, 1996.
2. D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.
3. D. McDermott. A Reactive Plan Language. Research Report YALEU/DCS/RR-864, Yale University, 1991.
4. D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.
5. L. Mösenlechner and M. Beetz. Parameterizing Actions to have the Appropriate Effects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September 25–30 2011.
6. L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, Issue 2, pages 257–286, February 1989.
7. M. Tenorth and M. Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. Part 1: The KnowRob System. *International Journal of Robotics Research (IJRR)*, 32(5):566 – 590, April 2013.

Planning for Multiple Robots in a Smart Environment

Maurizio Di Rocco[1], Federico Pecora[1], Alessandro Manzi[2], Raffaele Limosani[2], Giancarlo Teti[3], Filippo Cavallo[2], Paolo Dario[2], and Alessandro Saffiotti[1]

[1] Örebro University, 70182 Örebro, Sweden {modo, fpa, asaffio}@aass.oru.se
[2] BioRobotics Institute, 56025 Pisa, Italy
{a.manzi, r.limosani, f.cavallo}@sssup.it
[3] RoboTech srl, 56037 Pisa, Italy g.teti@robotechsrl.com

Abstract. We consider configuration planning in a real assistive system that consists of multiple robots embedded in a smart environment. Our configuration plans are fine-grained action plans which specify the causal, temporal, resource and information dependencies between individual sensing, computation, and actuation components. Components reside on-board the robots, like cameras or manipulation components, or they may be distributed in the environment, like presence sensors or elevators. Our configuration planner exhibits closed-loop performance, thanks to the fact that plans are represented as constraint networks, which are shared between the planner and the executor and updated during execution. This mechanism allows our system to react to contingencies at execution time. We illustrate our system on both simulated and real experiments performed in the context of a large EU project on robotics for elderly assistance.

Keywords: configuration planning, multi-robot systems, smart environments, planning with time and resources, elderly assistance

1 Introduction

Like several current projects, the Robot-Era EU project [1] aims at using robots to provide physical and cognitive assistance to elderly people. Differently from other projects, however, Robot-Era aims at creating holistic robotic services that go beyond the confines of a single house. To do this, Robot-Era relies on a variety of robots that operate in different environments (in the home, in the residential area, in the streets of the town) and collaborate among them and with the sensors and devices distributed in a smart environment. Context-dependent services are implemented by *planning* the individual activities and the cooperation patterns of the robots and devices available in the Robot-Era system.

As a typical example, a plan for a "deliver meal to Sven" service may start by using an outdoor transportation robot to bring the meal from the producer to the door of the residential building; the outdoor robot would then transfer the meal to a local indoor robot that moves into the building and delivers the meal to the door of Sven's apartment; the indoor robot would interact with sensors and actuators in the environment to, e.g., operate the elevator, to avoid a corridor where some people are having a conversation, and to know when Sven is available to receive the meal. In this paper, we discuss the requirements and the development of a planner able to generate plans like the one above.

The field of AI planning has progressed enormously since its beginnings as an exploration of logical reasoning for robots [11]. Yet, if you look inside a typical autonomous robot today you will see little evidence of this progress, and you may suspect that planning in AI has focused on issues which are not the main concerns of robot builders. Indeed, if you ever tried to program a robot to accomplish tasks in unstructured, everyday environments, you would expect an AI planner to provide the ability to act as a *knowledge-based controller* [5] for the robot: given a goal, the planner continuously synthesizes actions which bring about the achievement of the goal as the state of the world changes during execution. This entails (*requirement 1*) that the planner should possess knowledge and reason about the physical aspects of the domain, like time, space, and resources. In the real world, plans are subject to execution-time perturbations, such as actions taking longer than expected, resources not being available, or assumed causal requirements not being met. Therefore, the planner should (*requirement 2*) generate plans that enable a sufficient degree of flexibility to accommodate these contingencies during execution whenever possible. Furthermore, robots today are not monolithic entities, rather a collection of sensors, drivers, actuators, and information processing components. This entails that the planner should (*requirement 3*) decide how to compose these enabling modules while taking into account their physical and logical dependencies.

Today's planners exhibit some of the above features. For instance, a considerable amount of work has been done to integrate metric time into planning [17, 8, 14, 9, 3, 10]. Including time has allowed some planning systems to perform *continuous planning*, i.e., continuously synthesize plans as new goals and contingencies become known. Execution monitoring techniques have been developed which leverage the explicit temporal representation of these plans [12, 24, 23], thus effectively providing a few examples of planning for real robots. Although these constitute important steps towards obtaining planners that are appropriate for robots, they address only partially (i.e., in the temporal dimension) requirements 1 and 2. Some work has addressed the issue of including further dimensions into the planning problem, e.g., resources. In addition to addressing more fully requirement 1, some of these approaches [18, 13, 15] would also be well suited for use in closed loop with actuation and perception, as they maintain a certain level of least-commitment with respect to the timing of plan execution. Nevertheless, they are not proposed nor evaluated as closed-loop planning systems. [20] propose an extension of the IxTeT planner [15] for closed loop execution monitoring with resources — however, the technique is exemplified on single robot navigation tasks.

Much of the work in AI planning has focused on action planning [16] — however, when planning for robots, actions are not the only aspect upon which a planner should reason. We are interested in *configuration plans*, namely fine-grained plans for collections of multiple robots and devices [27] which specify the causal, temporal, resource and information dependencies between individual sensing, computation, and actuation components. Configuration planners have been proposed before: the AsymTre architecture [25] considers a set of robots equipped with software modules, called schemas, able to sense and modify the environment. Schemas are interconnected through information channels which are decided by a planning process. Further refinements of this idea are presented by [28, 26]: the former introduces metrics to estimate the information quality gathered by the chosen configuration, while the latter provides more sophisticated techniques to gather the needed information. This approach focuses on information exchange ignoring resource and temporal requirements. Furthermore, relying on an auction procedure to incrementally build the plan, this architecture doesn't manage explicitly the presence of shared actions in the planning process. In fact, as the authors point out, the backtracking procedure involving these schemas could lead to problems that are currently not addressed. [29] address a limited form of resource reasoning, however the method does not provide closed-loop execution monitoring; interestingly, dependencies are used to enforce a basic form of temporal constraints, namely precedences. An explicit representation of the world is instead considered by [21]: differently from AsymTre, this system leverages a propositional logic description of the world based on standard planning techniques. Reasoning is performed by coupling an action planner together with a configuration planner: the former provides a sequence of actions that have to be further refined by the latter; in turn, the configuration planner chooses software modules to activate and decides the related communication linkage. This system allows a detailed representation of the evolution of the world — however, it is decoupled from execution, and therefore suffers from many of the aforementioned problems. An improved version [22] takes into account multiple goals through a merging sequence. Similarly to AsymTre, the approach lacks the ability to perform on-line plan monitoring and execution.

In this paper, we propose a configuration planner which fulfills the above requirements by combining three enabling factors: (1) representing a (configuration) plan as a constraint network; (2) defining the configuration planning process as a search in the space of such networks; and (3) sharing the constraint network between the planner and the executor.

2 Representation

Our approach is grounded on the notion of *state variable*, which models elements of the domain whose state in time is represented by a symbol. State variables, whose domains are discrete sets, represent parts of the real world that are relevant for the configuration planner's decision processes. These include the actuation and sensing capabilities of the robotic system, and the various aspects of the environment that are meaningful. E.g., a state variable can represent the capabilities of a robot, whose meaningful states might be "navigating" or "grasping", or the interesting states of the environment, e.g., a light which can be "on" or "off". Let $\mathcal{S}$ be the set of state variables in a given application scenario.

Some devices require resources when they are in given states. A *reusable resource* has a limited capacity which is fully available when not required by a device. An example is power: a maximum wattage is available, and devices can simultaneously require power so long as the sum of requirements is less than the maximum power. We denote with $\mathcal{R}$ the set of all resource identifiers, and with $\mathrm{Cap}(R) \in \mathbb{N}$ the capacity of $R \in \mathcal{R}$.

Devices in our domain may serve the purpose of providing or requiring *information contents*, e.g., a software component requiring range data from a laser range finder, and providing localization information. We denote IC the set of all information contents.

2.1 Representing Configuration Plans and Goals

We employ *activities* to represent predicates on the possible evolution of state variables:

Definition 1 *An* activity a *is a tuple* $(\mathrm{x}, \mathbf{v}, I, u, \mathrm{In}, \mathrm{Out})$*, where*

- $\mathrm{x} \in \mathcal{S} = \{\mathcal{S}_{contr} \cup \mathcal{S}_{obs} \cup \mathcal{S}_{int}\}$ *is a* state variable*, where*
  - $\mathcal{S}_{contr}$ *is the set of controllable variables*
  - $\mathcal{S}_{obs}$ *is the set of observable variables*
  - $\mathcal{S}_{int}$ *is the set of internal variables*
- $\mathbf{v}$ *is a* possible state *of the state variable* x*;*
- $I = [I_s, I_e]$ *is a* flexible temporal interval *within which the activity can occur, where* $I_s = [l_s, u_s], I_e = [l_e, u_e], l_{s/e}, u_{s/e} \in \mathbb{N}$ *represent, respectively, an interval of admissibility of its start and end times;*
- $u : \mathcal{R} \to \mathbb{N}$ *specifies the* resources used *by the activity;*
- $\mathrm{In} \subseteq \mathrm{IC}$ *is a set of* required information contents*;*
- $\mathrm{Out} \subseteq \mathrm{IC}$ *is a set of* provided information contents.

The notation $(\cdot)^{(a)}$ indicates an element of the five-tuple pertaining to activity a. The pair $(\mathrm{x}^{(a)}, \mathbf{v}^{(a)})$ asserts a particular state $\mathbf{v}$ of the state variable x; $I^{(a)}$ represents possible temporal intervals of occurrence of the state $\mathbf{v}^{(a)}$ of state variable $\mathrm{x}^{(a)}$. Note that a pair of activities (a, b) is *possibly concurrent* if $I^{(a)} \cap I^{(b)} \neq \emptyset$. A pair (a, b) of possibly concurrent activities thus indicates that $\mathrm{x}^{(a)}$ and $\mathrm{x}^{(b)}$ can be, respectively, in states $\mathbf{v}^{(a)}$ and $\mathbf{v}^{(b)}$ at the same time.

The notation $(\cdot)$ is used to indicate the unspecified parameters of an activity — e.g., $(\mathrm{x}, \cdot, I, u, \mathrm{In}, \mathrm{Out})$ denotes a predicate asserting that state variable x can be in any state during interval I.

Activities can be bound by *temporal constraints*, which restrict the occurrence in time of the predicates. Temporal constraints can be of two types. *Binary* temporal constraints in the form $a\,\mathrm{C}\,b$ prescribe the relative placement in time of activities a, b — these constraints are relations in Allen's Interval Algebra [2], and restrict the possible bounds for the activities' flexible temporal intervals $I^{(a)}$ and $I^{(b)}$. *Unary* temporal constraints in the form $\mathrm{C}\,a$ prescribe bounds on the start or end time of an activity a — these constraints are commonly referred to as *release time constraints* and *deadlines*. Allen's interval relations are the thirteen possible temporal relations between intervals, namely "precedes" (p), "meets" (m), "overlaps" (o), "during" (d), "starts" (s), "finishes" (f), their inverses (e.g., p^{-1}), and "equals" ($\equiv$).

When state variables are used to represent a system, the overall temporal evolution of such system is described by a *constraint network*:

Definition 2 *A* constraint network *is a pair* $(\mathcal{A}, \mathcal{C})$*, where* $\mathcal{A}$ *is a set of* activities *and* $\mathcal{C}$ *is a set of* constraints *among activities in* $\mathcal{A}$*.*

A constraint network can be used to represent a *configuration plan*. Configuration plans are said to be *feasible* if they are consistent with respect to the resource, state, and temporal requirements. Specifically,

Definition 3 *A configuration plan* $(\mathcal{A}, \mathcal{C})$ *is* feasible *iff:*

- *the constraint network is* temporally consistent*, i.e., there exists at least one allocation of fixed bounds to intervals such that all temporal constraints are satisfied;*
- *activities do not over-consume resources, i.e.,* $\sum_{a \in A} u^{(a)}(R) \leq \mathrm{Cap}(R), \forall R \in \mathcal{R}$, *where* $A \subseteq \mathcal{A}$ *is a set of possibly concurrent activities;*
- *activities do not prescribe that state variables assume different states in overlapping temporal intervals, i.e.,* $\mathbf{v}^{(a)} \neq \mathbf{v}^{(b)}, \forall (a,b) \in A \times A : \mathrm{x}^{(\mathrm{a})} = \mathrm{x}^{(\mathrm{b})}$, *where* $A \subseteq \mathcal{A}$ *is a set of possibly concurrent activities.*

A *goal* for a configuration planning problem is also represented as a constraint network, therefore expressing temporal, resource, state and information requirements. Typically, a goal (A_g, C_g) is an under-specified configuration plan. Initial conditions are feasible sub-networks of a goal.

2.2 Domain Representation

Given a goal (A_g, C_g) and a configuration plan $(\mathcal{A}, \mathcal{C})$ which contains the goal, the feasibility of the configuration plan is not a sufficient condition for achieving the goal. This is because feasibility does not enforce information and causal requirements. The way these requirements are to be enforced depends on a *domain*:

Definition 4 *A* configuration planning problem *is a pair* $((A_g, C_g), \mathcal{D})$*, where* (A_g, C_g) *is a goal constraint network, and* $\mathcal{D}$ *is a* domain*. The domain is a collection of* operators*, which describe the information and causal dependencies between activities.*

Definition 5 *An* operator *is a pair* $(a, (A, C))$ *where*

- $a = (\mathrm{x}, \mathbf{v}, \cdot, \cdot, \cdot, \mathrm{Out})$ *is the* head *of the operator;*
- $A = A_p \cup A_e \cup \{a\}$ *is a set of activities, where*
  - A_p *is a set of* preconditions*, i.e., requirements, in terms of state variable values, information input, and resource usage, needed to achieve the state* $\mathbf{v}^{(a)}$ *of state variable* $\mathrm{x}^{(a)}$ *and to produce* $\mathrm{Out}^{(a)}$*;*
  - A_e *is a set of* effects*, i.e., state variable values entailed by the achievement of state* $\mathbf{v}^{(a)}$ *of state variable* $\mathrm{x}^{(a)}$*;*
- C *is a set of temporal constraints among activities in* A*.*

Computing a configuration plan consists in selecting and instantiating operators form the domain into the goal constraint network. Unlike in classical planning, the relevance of an operator (denoted γ^{-1} in [16]) is not determined by unifying effects with sub-goals, rather by the unification of an operator's head with a sub-goal. The head of an operator is a non-ground activity which describes the value of a state variable and the information provided as a result of applying the operator. Preconditions and effects are used during execution, the former to determine the control action(s) given to the system (input regulation problem), the latter as a part of state estimation (see next Section).

An operator can specify the information requirements needed for achieving a particular functionality. For instance, the $\mathrm{MoveFromTo}$ operator, which does not provide any information content, requires the current position of the robot:

$$\begin{aligned} a &= (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \emptyset) \\ A_p &= \{a_1, a_2\}, A_e = \{a_3\}, \text{where} \\ a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{position}\}) \\ a_2 &= (\mathrm{RobotLocation}, \mathbf{kitchen}, \cdot, \cdot, \cdot, \cdot) \\ a_3 &= (\mathrm{RobotLocation}, \mathbf{livingroom}, \cdot, \cdot, \cdot, \cdot) \\ C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2, a \,\mathrm{m}\, a_3\} \end{aligned}$$

The head of the operator is a predicate on the functionality $\mathrm{MoveFromTo}$. The operator is considered relevant when the constraint network contains an activity $(\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \cdot)$, i.e., when a (sub-)goal stating that the robot must move from the kitchen to the living room is present in the network. The operator also prescribes the temporal relations that must exist between the activities, namely that the $\mathrm{MoveFromTo}$ functionality should occur during the availability of the position data ($a \,\mathrm{d}\, a_1$), that it should be met by the precondition of the robot being in the kitchen ($a \,\mathrm{m}^{-1}\, a_2$), and that it meets the effect of the robot being in the living room ($a \,\mathrm{m}\, a_3$).

An operator can also represent a means to achieve information requirements. For example, the operator

$$\begin{aligned} a &= (\mathrm{VisualSLAM}, \mathbf{running}, \cdot, u(\mathrm{CPU}) = 10, \cdot, \{\mathrm{position}\}) \\ A_p &= \{a_1, a_2\}, A_e = \emptyset, \text{where} \\ a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{range\_data}\}) \\ a_2 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{ref\_frame}\}) \\ C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2\} \end{aligned}$$

specifies one way to achieve the necessary information requirement (position) for the $\mathrm{MoveFromTo}$ operator, namely through visual SLAM. This localization functionality requires (1) a functionality which provides range data, (2) a reference frame for the computation of the position estimate, and (3) 10% of the CPU resource. Also, the operator states that range data should be available during the entire duration of the localization process, and that the reference frame is needed at the beginning of the process.

The above operator does not specify how to obtain needed information inputs. For instance, range data might be provided through the following operator:

$$\begin{aligned} a &= (\mathrm{StereoCamDriver}, \mathbf{on}, \cdot, u(\mathrm{Cam1}) = 1, \cdot, \{\mathrm{range\_data}\}) \\ A_p &= \{a_1\}, A_e = \emptyset, \text{where } a_1 = (\mathrm{Light}, \mathbf{on}, \cdot, \cdot, \cdot, \cdot) \\ C &= \{a \,\mathrm{d}\, a_1\} \end{aligned}$$

An operator may also specify that the reference frame is obtainable by invoking a functionality of the stereo camera's pan-tilt unit:

$$\begin{aligned} a &= (\mathrm{PanTilt}, \mathbf{return\_ref\_frame}, \cdot, \cdot, \cdot, \{\mathrm{ref\_frame}\}) \\ A_p &= \emptyset, A_e = \emptyset, C = \emptyset \end{aligned}$$

The above operators can be applied to obtain a configuration plan from the following goal constraint network:

$$\begin{aligned} A &= \{a_0 = (\text{MoveFromTo}, \textbf{kitchen\_livingroom}, I_0, \cdot, \cdot, \cdot)\}, \\ C &= \emptyset \end{aligned}$$

A particular application of the above operators may refine the given constraint network to the following:

$$\begin{aligned} A = \{a_0 &= (\text{MoveFromTo}, \textbf{kitchen\_livingroom}, I_0, \emptyset, \emptyset, \emptyset) \\ a_1 &= (\text{VisualSLAM}, \textbf{running}, I_1, u(\text{CPU}) = 10, \{\text{ref\_frame}, \text{range\_data}\}, \{\text{position}\}) \\ a_2 &= (\text{RobotLocation}, \textbf{kitchen}, I_2, \emptyset, \emptyset, \emptyset) \\ a_3 &= (\text{RobotLocation}, \textbf{livingroom}, I_3, \emptyset, \emptyset, \emptyset) \\ a_4 &= (\text{StereoCamDriver}, \textbf{on}, I_4, u(\text{Cam1}) = 1, \emptyset, \{\text{range\_data}\}) \\ a_5 &= (\text{PanTilt}, \textbf{return\_ref\_frame}, I_5, \emptyset, \emptyset, \{\text{ref\_frame}\}) \\ a_6 &= (\text{Light}, \textbf{on}, I_6, \emptyset, \emptyset, \emptyset)\}, \\ C = \{a_0 \, \mathrm{d} \, a_1&, a_0 \, \mathrm{m}^{-1} \, a_2, a_0 \, \mathrm{m} \, a_3, a_1 \, \mathrm{d} \, a_4, a_1 \, \mathrm{m} \, a_5, a_4 \, \mathrm{d} \, a_6\} \end{aligned}$$

This network represents a temporally consistent configuration plan in which resources are never used beyond their capacity, and state variables are never required to assume different values in overlapping temporal intervals. The plan is therefore *feasible*. Furthermore, the plan contains activities providing the required information contents as determined by the operators in the domain. However, not all causal dependencies are necessarily achieved by construction. If, e.g., the initial condition does not state that the light is on, the configuration planner would regard the activity a_6 as yet another sub-goal to satisfy, and might do so through the following operator:

$$\begin{aligned} a &= (\text{Light}, \textbf{on}, \cdot, \cdot, \cdot, \cdot) \\ A_p &= \emptyset, A_e = \{a_1\}, \text{where } a_1 = (\text{LightController}, \textbf{on}, \cdot, \emptyset, \cdot, \cdot) \\ C &= \{a \, \mathrm{p}^{-1} \, a_1\} \end{aligned}$$

This operator models an actuation process (Light represents an environment variable), and its application would refine the configuration plan by adding an activity $a_7 = (\text{LightController}, \textbf{on}, I_7, \emptyset, \emptyset, \emptyset)$ to the network, along with the constraint $a_6 \, \mathrm{p}^{-1} \, a_7$, prescribing that the LightController be in state **on** before the light is required to be on. Note that the light control functionality has no information requirements ($\text{In}^{(a_1)} = \emptyset$).

3 Planning in Closed Loop

Fig. 1 provides an overview of our approach. As a whole, our architecture can be seen as a controller for a dynamical system: the *controlled system* consists of the robot(s) and the environment in which they operate; the *controller* is composed of an observer and a regulator, and has the task to dispatch actuation commands to the controlled system so that its observed state S coincides with the desired state G.

At the core of the controller is a shared constraint network. The *observer* adds activities and temporal constraints to this network, which represent the current state of the environment as provided by sensors: if a sensor sensorX reports a new reading $\mathbf{v}$ at time t_{now}, the observer inserts a new activity $(\text{sensorX}, \mathbf{v}, I, \emptyset, \emptyset, \emptyset)$ and adds a temporal constraint restricting the beginning of I to be t_{now}; if the reading of sensorX changes, the previous activity is constrained to end at t_{now} and another activity is started.

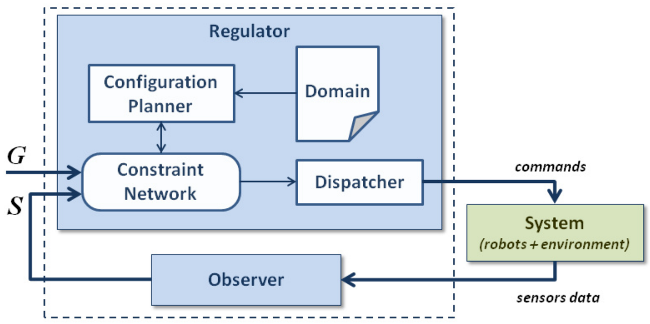


Fig. 1. Overview of the control architecture.

The *regulator* includes the configuration planner and a dispatcher that sends executable actions to the system. An action $(\mathrm{deviceY}, \mathbf{v}, [I_s, I_e], \cdot, \cdot, \cdot)$ is considered executable when two conditions are met: (1) $t_{\text{now}} \leq \min\{I_s\}$; and (2) its observable preconditions are verified in the current state. The latter condition is tested by attempting to unify each activity representing such a precondition with a sensed activity produced by the observer. If unification fails, the precondition is delayed by inserting a temporal constraint, and re-evaluated at the next iteration. If both conditions are met, the command $\mathbf{v}$ is transmitted to $\mathrm{deviceY}$.

The configuration planner, which implements the control logic of the regulator, is a search procedure which continuously modifies the constraint network so as to guarantee the presence of a feasible plan given the goal $G = (A_g, C_g)$. The resulting constraint network represents one or more temporal evolutions of the state variables that guarantee the achievement of G under nominal conditions. In this sense, our planner follows a least commitment principle. Feasible and goal-achieving configuration plans are obtained by means of five interacting solvers. (1) a *temporal solver* propagates temporal constraints to refine the bounds $[l_s, u_s]$, $[l_e, u_e]$ of activities, and returns failure if and only if temporally consistent bounds cannot be found; (2) a *resource scheduler*, which chooses and posts to the network temporal constraints so as to remove temporal overlap from sets of over-consuming, possibly concurrent activities [4]; (3) a *state variable scheduler*, which exploits the same constraint-posting mechanism of the resource scheduler to ensure that activities do not prescribe conflicting states in overlapping intervals; (4) an *information dependency reasoner*, which instantiates relevant operators (in the form of activities and temporal constraints) so as to enforce the information dependencies modeled in the domain; (5) and a *causal planner*, which instantiates relevant operators so as to enforce the causal dependencies modeled in the domain.

Temporal feasibility enforcement is not subject to multiple choices, as the temporal constraints form a Simple Temporal Problem [6], which is tractable. Tractability also holds for information dependencies, which in our approach constitute an acyclic propositional Horn theory. Conversely, all other reasoners must search in the space of alternative choices for conflict resolution (e.g., alternative sequencing decisions, alternative operator selections). All of these choices are seen as *decision variables* in a high-level Constraint Satisfaction Problem (CSP). Given a decision variable d, its possible values constitute a finite domain $\delta^d = \{(A_r^d, C_r^d)_1, \ldots, (A_r^d, C_r^d)_n\}$, whose values are alternative constraint networks, called *resolving constraint networks*. The individual solvers are used to determine resolving constraint networks $(A_r^d, C_r^d)_i$, which are iteratively added to the goal constraint network (A_g, C_g).

```
Function Backtrack($(A_g, C_g)$): success or failure
1  $d \leftarrow$ Choose($(A_g, C_g)$, $h_{var}$)
2  if $d \neq \emptyset$ then
3      $\delta^d = \{(A_r^d, C_r^d)_1, \ldots, (A_r^d, C_r^d)_n\}$
4      while $\delta^d \neq \emptyset$ do
5          $(A_r^d, C_r^d)_i \leftarrow$ Choose($d$, $h_{val}$)
6          if $(A_g \cup A_r^d, C_g \cup C_r^d)$ is temporally consistent then
7              return Backtrack($A_g \cup A_r^d, C_g \cup C_r^d$)
8          $\delta^d \leftarrow \delta^d \setminus \{(A_r^d, C_r^d)_i\}$
9      return failure
10 return success
```

In order to search for resolving constraint networks, we employ a systematic search (see Algorithm `Backtrack`), which occurs through standard CSP-style backtracking. The decision variables are chosen according to a variable ordering heuristic h_{var} (line 1); the alternative resolving constraint networks are chosen according to a value ordering heuristic h_{val} (line 5). The former decides which (sub-)goals to attempt to satisfy first, e.g., to support a functionality by applying another operator, or to resolve a scheduling conflict. The latter decides which value to attempt first, e.g., whether to prefer one operator over another (recall that there may be more than one decomposition for a given activity.) Note that adding resolving constraint networks may entail the presence of new decision variables to be considered.

The possible values for resource contention or unique state decision variables are temporal constraints. In particular, our framework relies on the earliest time timelines to assess both resource over-consumption and multiple overlapping states. Values for information decision variables are ground operators, as shown in the previous Section. Lastly, values for causal decision variables are either ground operators, or unifications with activities that already exist in the constraint network. Search uses unification to build on previously added activities — e.g., leveraging that the light has already been turned on to support a previously branched-upon causal dependency. Unification also allows to accommodate on-going sensing and execution monitoring processes during planning. For instance, $(\text{Light}, \textbf{on}, I^{(a)}, \emptyset, \emptyset, \emptyset)$ may be supported by unification with $(\text{Light}, \textbf{on}, [[0, 0][13, 13]], \emptyset, \emptyset, \emptyset)$ which models the temporal interval within which a light source was sensed.

Since the configuration planner is in a closed loop with the system (through the observer and dispatcher), modifications of the constraint network necessary to achieve the goal can occur whenever a disrupting renders the network an infeasible plan. Note that due to the presence of the above solvers in this loop, the modifications made to the network in the face of disturbances can take on the form of temporal propagation, resource or state variable scheduling, or operator application, depending on the situation. If temporal reasoning and scheduling fail, replanning is needed: the planner empties the constraint network of all activities whose lower bound is greater than the current time, and Algorithm `Backtrack` is re-invoked. Note that all activities representing operators currently under execution, as well as those already executed remain in the network, as do all activities representing the sensed state of the physical world. As shown below, this mechanism also enables to deal with dynamically posted goals.

4 Experimental Evaluation

We present two experiments describing distinctive features of our planner. The first experiment demonstrates the ability to pursue multiple goals in a multi-robot system; the second shows preliminary results achieved using real platforms.

4.1 Simulations

The first experiment has been carried out in the Gazebo simulator [19], and involves two Turtlebot platforms[4] in the following scenario:

> *Sven lives in an apartment with two robots. They can accomplish two tasks: "throw garbage" and "deliver evening pills". The "throw garbage" task consists of bringing the garbage outside apartment. This task has to be accomplished after the user has had dinner, but no later than 22:00 since this is the time when the garbage collector visits the building. The "deliver evening pills" consists of bringing to Sven his pills after he is in bed, and stay there waiting until he falls asleep since he occasionally needs some assistance after taking the pills, e.g., to get some more water or to be escorted to the bathroom. Both robots are equipped with a Kinect and a laser range-finder, and they may use both for navigation. The Kinect is usually preferred since it uses less energy; however, when the robots are colocated with Sven, the laser is preferred for privacy. Furthermore, since the kitchen is a cluttered environment, the two robots cannot access it at the same time.*

This simple scenario already introduces specific aspects that are very important in robotic domains: deadlines (throw garbage task), use of shared resources (the presence of Sven and the use of the Kinect are mutually exclusive, as well as access to the kitchen) and alternative ways to achieve tasks (either laser- or Kinect-based navigation).

Fig. 4.1-b shows the plan generated by the planner (note that only Robot1 is employed). The robot will first help Sven and then throw away the garbage. Unfortunately, this evening Sven is particularly interested in a TV program, and will go to bed later. Therefore, the plan will be subject to an initial delay which will ultimately make the plan fail due to the missed deadline. When the delay becomes apparent, the planner decides to use both robots, as shown in Fig. 4.1-a, so as to meet the deadline. Note that the two robots' tasks are partially concurrent (due to shared resources).

[4] www.ros.org/wiki/turtlebot_simulator

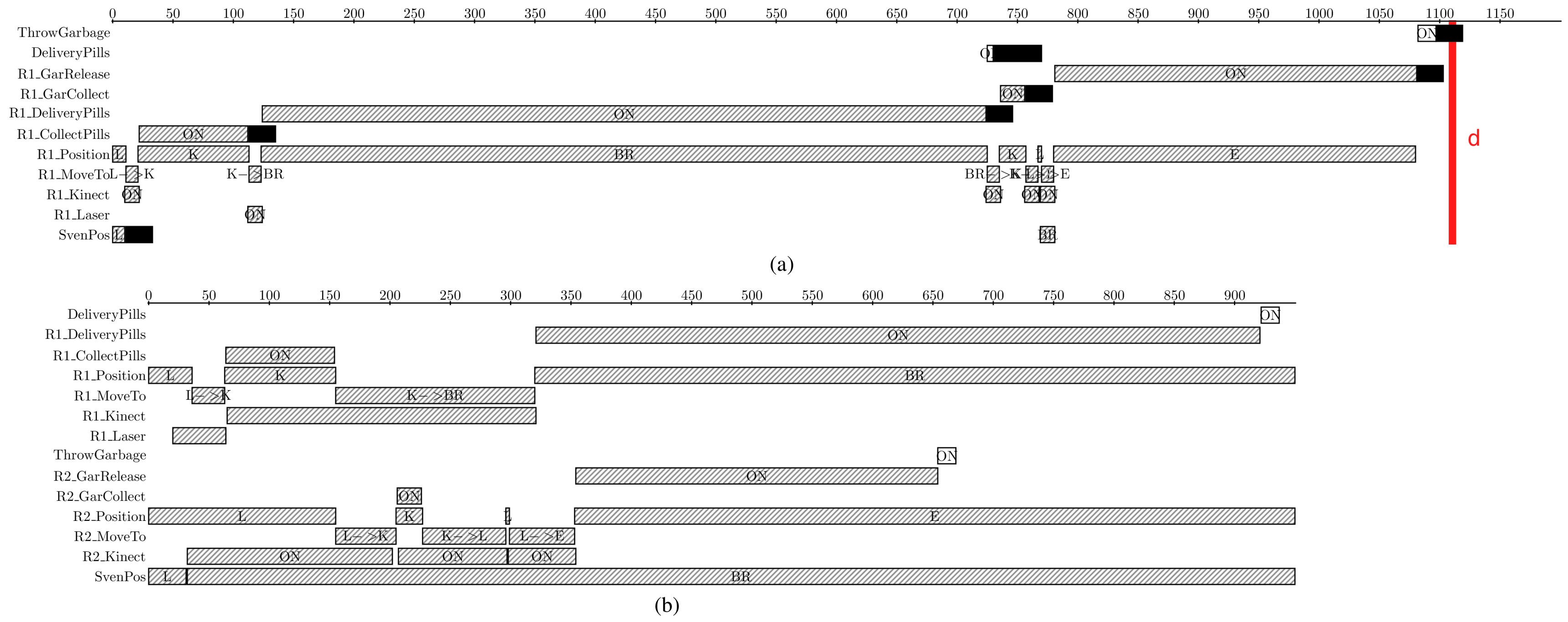


Fig. 2. a) The robot1 is in charge of both tasks: from the livingroom (L) it will go to the kitchen (K) to collect the pills (R1_CollectPills) and afterwards it will go to the bedroom (BR) to deliver the pills (R1_DeliveryPills). After executing this task it should go to the kitchen again in order to collect the garbage (R1_GarCollect) and then throwing it away at the entrance (R1_GarRelease) of the apartment. The motion from one location to another is handled by the MoveTo functionality that can either exploit the Kinect or Laser modules. Unfortunately Sven is late (SvenPos) so his permanence in the livingroom is extended (black bar). This delay is propagated in the plan so that the goal related to the garbage (ThrowGarbage) misses its deadline (d). Note how the navigation to BR is supported by the laser since the robot can not use the Kinect if Sven is in the same room. b) Robot1 is in charge of delivering the pills while Robot2 is in charge of delivering the garbage. Robot2 will go to the kitchen (after R1 has left since that room is modeled as a shared resource) to collect the garbage (R2_GarCollect) and then it will go to the entrance (passing through the livingroom) to throw it away (R2_GarRelease). The motion from one location to another is handled by the MoveTo functionality that can either exploit the Kinect or Laser modules. The parallelization of the tasks execution allows to respect the imposed deadline related to the garbage task.

4.2 Real scenario

The following experiment illustrates a scenario which constitutes one of the eleven case-studies used in a long-term field evaluation within the EU project Robot-ERA [1]. Robot-ERA aims at studying the application of robotic technologies for elderly care. In the following, we consider a *delivery groceries* case similar to the example discussed in the Introduction:

> *Sven has ordered some goods at the nearby supermarket; he lives at the first floor of a building where a condominium robot,* coro*, and an outdoor robot,* oro*, provide services for the inhabitants.* oro *is instructed to go the supermarket to collect goods prepared by the clerk. When coming back, it will pass the goods to* coro *at the entrance of the building.* coro *will then deliver the goods to Sven; it will interact with the elevator in order to move across floors inside the building.*

In this simple scenario we have not modeled unexpected contingencies, but the planner was in charge of coordinating real platforms and an actuator, i.e., the planner directly controls the elevator door and its position. The experiment was executed at the DomoCasa laboratory of the Biorobotics center of Scuola Superiore SantAnna in Peccioli, Italy, (Fig. 3-a,b), using the first (Fig. 3-c) and ground (Fig. 3-d) floors of the building.

The indoor robot is a Scitos G5 (Fig. 3-c) from MetraLabs, while the outdoor robot is a DustCart (Fig. 3-d) from Robotech Srl: both robots are equipped with ROS [5]. The planner communicates with the robots and elevator controller through the PEIS Middleware, that provides a transparent communication system among heterogeneous devices [27].

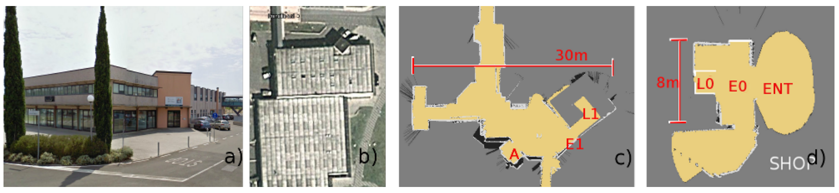


Fig. 3. a) The Biorobotics Institute where the experiments have been executed. b) Satellite image of the Biorobotics Institute. c) ROS map of the first floor. d) ROS map of the ground floor. In both maps the locations taken into account in the plan are reported.

The plan generated is depicted in Fig. 4 while video of the experiment is available at http://youtu.be/jY74RtufYIo.

5 Conclusions and future work

In this paper we have provided preliminary experiments using a real robotic system orchestrated by a configuration planner. The latter provides the capability of reason-

[5] http://www.ros.org/

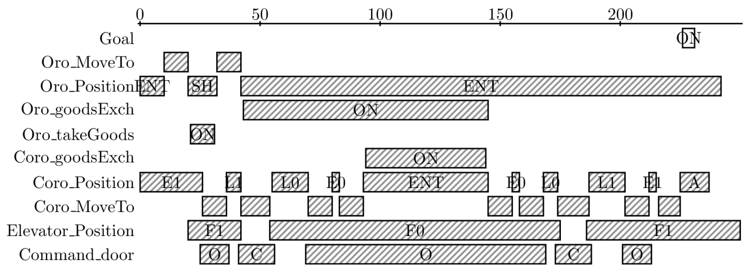


Fig. 4. *Oro* moves from the entrance (ENT) to the shop (SH) to fetch goods (Oro_takeGoods) and then comes back. In the meanwhile *Oro* moves from its location (E1) to the elevator at the first floor (L1); the elevator brings *Oro* to the ground floor (L0). Then, it exits the elevator (E0) and finally moves to ENT where the exchange occurs (Coro_goodsExch and Oro_goodsExch). After the completion of the exchange, *Coro* goes upstairs to Sven's apartment (A) to deliver the package (Goal). The motion from one location to another is handled by the MoveTo functionality that can either exploit the Kinect or Laser modules. Note how travelling to and from the elevator implies the opening of the elevator door, and that moving the elevator requires the door to be closed.

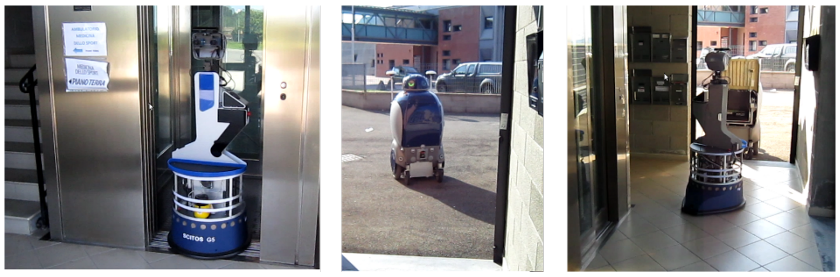


Fig. 5. Left: *coro* - a SCITOS-G5 from Metralabs; Center: *doro* - a DustBot from Robotech Srl; Right: The robots exhanging goods.

ing about several features that are particularly important in robotics, namely temporal aspects, shared resources, causal relations and reaction to unexpected contingencies. While the real experiment is focused on the integration between the planner and real devices, the distinctive features of the planner are demonstrated by simulations. Further experiments aimed at evaluating our system's performance in real scenarios in the presence of contingencies have been performed and are reported in [7]. Future work points towards several directions: theoretical studies will be conducted to study the performance of the planner with respect to the use of variable and value ordering heuristics aimed at guiding the search during the plan generation. In order to enhance the effectiveness of the system both in the planning and execution phases, context recognition modules devoted to reason about human activity will be integrated. Furthermore extensive experiments regarding the aforementioned case studies will be conducted exploiting a richer set of sensors and actuators.

Acknowledgements

This work was funded by the EC Seventh Framework Programme (FP7/2007-2013) grant agreement no. 288899 Robot-Era – "Implementation and integration of advanced **Robot**ic systems and intelligent **E**nvironments in **r**eal scenarios for the **a**geing population" (`www.robot-era.eu`).

References

1. The Robot-ERA project. Home page: `www.robot-era.eu`.
2. J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
3. J. Barreiro, M. Boyce, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, T. Smith, and M. Do. EUROPA: A platform for AI planning. In *Proc of ICAPS-ICKEPS*, 2012.
4. A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, January 2002.
5. T.L. Dean and M.P. Wellman. *Planning and control*. Morgan Kaufmann series in representation and reasoning. M. Kaufmann Publishers, 1991.
6. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.
7. M. Di Rocco, F. Pecora, and A. Saffiotti. When robots are late: Configuration planning for multiple robots with dynamic goals. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, nov. 2013.
8. Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, 20:155–194, 2003.
9. P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.
10. Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2009.
11. R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.
12. A. Finzi, F. Ingrand, and N. Muscettola. Model-based executive control through reactive planning for autonomous rovers. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004.
13. S. Fratini, F. Pecora, and A. Cesta. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.
14. Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Artif. Int. Res.*, 25(1):187–231, February 2006.
15. M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, pages 61–67, 1994.
16. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
17. S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: Space exploration through continuous planning. *Intelligent Systems*, 16(5):70–75, 2001.
18. U. Köckemann, F. Pecora, and L. Karlsson. Towards planning with very expressive languages via problem decomposition into multiple csps. In *Proc. of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*, 2012.

19. N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004.
20. Solange Lemai and Félix Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 617–622. AAAI Press, 2004.
21. R. Lundh, L. Karlsson, and A. Saffiotti. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems*, 56(10):819–830, 2008.
22. Robert Lundh. Robert lundh. robots that help each-other: Self-configuration of distributed robot systems. In *PhD Thesis. rebro University, rebro, Sweden, May 2009.*, 2009.
23. C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence*, Chicago, IL, 2008.
24. C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. A Deliberative Architecture for AUV Control. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*, Pasadena, May 2008.
25. L.E. Parker and F. Tang. Building multirobot coalitions through automated task solution synthesis. *Proc of the IEEE*, 94(7):1289–1305, 2006.
26. L.E. Parker and Y. Zhang. Task allocation with executable coalitions in multirobot tasks. *Proc of the Int. Conf. on Robotics and Automation (ICRA)*, 2012.
27. A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B.S. Seo, and Y.J. Cho. The PEIS-ecology project: vision and results. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2329–2335, 2008.
28. Yu Zhang and Lynne E. Parker. Solution space reasoning to improve iq-asymtre in tightly-coupled multirobot tasks. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 370 –377, may 2011.
29. Yu Zhang and Lynne E. Parker. Considering inter-task resource constraints in task allocation. *Autonomous Agents and Multi-Agent Systems*, 26(3):389–419, 2013.

Search Strategies for Partial-Order Causal-Link Planning with Preferences

Pascal Bercher and Fabian Ginter and Susanne Biundo

Institute of Artificial Intelligence, Ulm University, Germany

Abstract. This paper studies how to solve classical planning problems with preferences by means of a partial-order causal-link (POCL) planning algorithm. Preferences are given by soft goals – optional goals which increase a plan's benefit if satisfied at the end of a plan. Thus, we aim at finding a plan with the best *net-benefit*, which is the difference of the achieved preferences' benefit minus the cost of all actions in the plan that achieves them.
While many approaches compile soft goals away, we study how they can be addressed *natively* by a POCL planning system. We propose novel search and flaw selection strategies for that problem class and evaluate them empirically.

1 Introduction

Partial-order causal-link (POCL) planning in the tradition of SNLP [16] and UCPOP [17] provides plans as partially ordered sets of actions where causal dependencies between the actions are explicitly shown. This allows for flexibility with respect to the order in which actions can finally be executed. Moreover, it enables a human user to grasp the causal structure of the plan and to understand why certain actions are part of it [22]. Therefore, POCL planning is particularly suitable when plans have to be generated that are not to be carried out by machines or systems, but by humans. This appears, for example, in applications that realize support of the elderly [1,7,19] or provide assistance in some daily routines [5,6]. Plan quality is of particular importance in this kind of applications, since plans have to be "accepted" by the user which is more likely if they respect the individual's personal preferences.

In this paper, we address POCL planning with preferences by optimizing the net-benefit of solutions. In net-benefit planning [11], a classical planning problem is augmented by a set of soft goals: state features one would like to see satisfied in the final state produced by a solution plan, but if they are not, the plan is still regarded a valid solution. Each soft goal has a certain benefit which has a positive impact on the solution quality; every action contained in the solution decrements the respective quality, however. Thus, the goal is to find a solution to a planning problem that satisfies the preferences to the largest extent while taking into account the negative impact, like costs or time consumption, of the actions necessary to achieve them. This way, individual user preferences can be defined through soft goals with a corresponding (individual) benefit. We describe these preferences by arbitrary formulae over state features, so-called *simple preferences*.

In the literature, several kinds of preferences like simple preferences, state trajectory constraints, and action occurrences have been addressed. Many planning systems exist capable of handling such preferences and implementing various paradigms and search algorithms including hierarchical task network (HTN) planning [15,24], search in the space of states [2,3,8], and search using binary decision diagrams [10]. While several of these approaches keep the explicit representation of the preferences, other approaches suggest to compile them away [9,14]. This allows to use any planning system without making any changes due to the presence of preferences. If one does not compile the preferences away, one can either select a subset of the optional planning goals before the planning process and regard all of them as non-optional [23], or one keeps the preferences and decides during the planning process on which preference to work on, as it is done in this work. Not compiling away preferences but addressing them natively allows to incorporate a human user into the planning process, s.t. he can actively influence the planning process [21].

In our setting, a problem specification is given in terms of a POCL problem. The preferences are specified by so-called *at-end* preferences, as described in PDDL3 [11], the language for the fifth International Planning Competition (IPC-5). An at-end preference (or simple preference) is an arbitrary formula over state features which should hold at the end of a plan if possible. We developed a POCL planning algorithm that is capable of solving this kind of problems. Each preference is transformed into disjunctive normal form (DNF). Our algorithm employs two different strategies to handle disjunctive preferences. Furthermore, several flaw selection strategies for planning with preferences were implemented. We evaluated these strategies on a large number of problems and show how they influence planning performance.

2 Problem Setting

POCL planning performs search in the space of plans. A plan P in POCL planning is a tuple $(PS, V, C, \prec)$. PS is a set of labeled plan steps, i.e., each plan step $l{:}o$ consists of a (partially) instantiated operator o and a label $l \in L$ to differentiate between multiple occurrences of the same operator, L being an infinite set of label symbols. For the sake of simplicity, we refer to a plan step $l{:}o$ by l. Operators are defined as usual, consisting of a precondition and an effect, each of which is a conjunction of literals. A ground operator is called an action and every action a has an associated cost, denoted by $\texttt{cost(a)}$. V is a set of variable constraints $v \circ x$ with v being a variable and $\circ \in \{=, \neq\}$ denoting a co- or a non-co-designation with x, x being an object or another variable. Please note that the variable constraints V can be interpreted as a constraint satisfaction problem (CSP) in which domains of the variables are the available objects. C is a set of causal links. A causal link (l, ϕ, l') denotes that the precondition literal

ϕ of plan step l' is an effect of plan step l and that ϕ is *supported* or *protected* that way. Causal links are used to ensure that all preconditions of all plan steps in a plan are satisfied in the state in which they are to be executed. The set $\prec$ of ordering constraints $(l, l'), l, l' \in L$ is used to resolve conflicts occurring during search.

A POCL planning problem π is a tuple $(\mathcal{O}, P_{init})$, where $\mathcal{O}$ is the set of available operators and P_{init} is the initial plan. The initial plan contains two artificial actions l_0:`init` and l_∞:`goal` which encode the initial state and the goal description, respectively: l_0 has no precondition and the initial state as effect, whereas l_∞ has no effect and the goal description as precondition. Initially, $(l_0, l_\infty) \in \prec$ and during planning all inserted plan steps are inserted between these two actions.

A plan $P_{sol} = (PS, V, C, \prec)$ is called a solution of π if and only if the following criteria hold:

1. P_{sol} is a refinement of $P_{init} = (PS', V', C', \prec')$, i.e., $PS \supseteq PS', V \supseteq V', C \supseteq C'$, and $\prec \supseteq \prec'$,
2. every precondition is protected by a causal link, i.e., for every precondition literal ϕ of any plan step $l{:}o \in PS$ there is a plan step $l'{:}o' \in PS$, s.t. $(l, \phi, l') \in C$,
3. no causal links are threatened, i.e., for each causal link $(l, \phi, l') \in C$ the ordering constraints $\prec$ and variable constraints V ensure that no plan step $l''{:}o'' \in PS$ with an effect $\neg\psi$ can be ordered between l and l', s.t. $\neg\phi$ is unifiable with $\neg\psi$,
4. the ordering constraints and causal links are free of contradiction, i.e., $\prec$ does not contradict the ordering induced by C and $\prec$ does not induce cycles, and
5. P_{sol} is ground, i.e., each variable in V has a domain of size 1.

Please note that these syntactical solution criteria ensure that every linearization of the plan steps in P_{sol} that is compatible with its ordering constraints is an action sequence that is executable in the initial state and leads to a state satisfying the goal description.

In addition to a planning problem π, we are given a set of preferences $\mathcal{P}$, each of which is an arbitrary formula over state features. The function $b : \mathcal{P} \to \mathbb{R}$ maps each preference p to its benefit $b(p)$. The preferences follow the semantics of the at-end preferences described in PDDL3. Thus, a solution P_{sol} satisfies a preference p if and only if every linearization of the actions in P_{sol} that is compatible with its ordering constraints generates a state s such that s satisfies p.

Let $P_{sol} = (PS, V, C, \prec)$ be a solution to the planning problem π. We are interested in finding good solutions w.r.t. action costs and satisfied preferences. To that end, the solution quality is defined by its net-benefit $\texttt{netBen}(\texttt{P}_{sol}) := \sum_{p \in \mathcal{P}(P_{sol})} b(p) - \sum_{a \in PS} \texttt{cost}(\texttt{a})$, where $\mathcal{P}(P_{sol})$ denotes the set of preferences satisfied by P_{sol}.

3 A POCL Algorithm for Planning with Preferences

In this section, we introduce our POCL planning algorithm (cf. Alg. 1) that is capable of solving problems containing simple preferences.

Algorithm 1: Preference-based POCL algorithm

```
Input  : The fringe fringe = $\{(P_{init}, \texttt{flaws}_h(P_{init}), \texttt{flaws}_s(P_{init}))\}$.
Output : A plan or fail.
1  best-netBen := $-\infty$
2  best-P := fail
3  while fringe $\neq \emptyset$ and "no timeout" do
4      $N := (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$ := planSel(fringe)
5      if $\texttt{flaws}_h(P) = \emptyset$ and netBen(P) > best-netBen then
6          best-netBen := netBen(P)
7          best-P := $P$
8      $f$ := flawSel($\texttt{flaws}_h(P) \cup \texttt{flaws}_s(P)$)
9      fringe := (fringe $\setminus \{N\}$) $\cup$ resolveFlaw($N, f$)
10 return best-P
```

Alg. 1 is a standard *flaw-based* POCL algorithm, meaning that every violation of a solution criterion is represented by a so-called *flaw*. More precisely, *for every plan component* that is involved in the violation of solution criteria 2. to 5. (cf. previous section), like a precondition of a plan step that is not yet protected by a causal link, one flaw is introduced to the set of all flaws of the current plan. Hence, a plan is a solution if there are no flaws. The main procedure consequently follows the idea of (1) selecting a promising search node/plan from a search fringe, (2) select a flaw for the current plan, and (3) resolve the selected flaw using all possible plan modifications and insert the resulting successor plans into the fringe; this is also the step, where new flaws are calculated. This loop is continued until the search space is exhausted or some timeout criterion is satisfied (cf. line 3).

Our algorithm still follows that basic procedure; however, there are important additional considerations one has to make, since we want to search for a solution with the best net-benefit, which imposes certain problems to standard flaw-based POCL algorithms.

To be able to search for plans which satisfy preferences, we need to include them in the plan structure. To that end, we first normalize the preferences by transforming them into DNFs. Then, we alter the structure of the artificial goal action l_∞, s.t. it additionally contains all preferences. As our POCL planning algorithm is based on the concept of flaws, we include additional flaws for the unsatisfied preferences. However, since these flaws do not violate a solution criterion, they must be distinguishable from the "ordinary" ones. Hence, we differentiate between two classes of flaws: *hard flaws*, as they were already described in the beginning of this section, and *soft flaws*, which indicate the non-satisfaction of a preference. The test for a solution is then performed purely based on the set of hard flaws (cf. line 5).

Alg. 1 has two "decision" points: (1) which search node/plan to work on and (2) given a search node, which flaw should be resolved next and *how* will it be resolved. The implementation of the first decision point defines the main search strategy. A reasonable strategy is A* with some heuristic judging the quality or goal distance of a plan. Hence, this heuristic should incorporate both action costs and preferences. We developed such a heuristic in previous work [4] and thus focus on the second decision point in this paper: the flaw selection in the presence of preferences (cf. line 8).

The question which flaw to resolve next is of major importance for the performance of any POCL planner, as the order in which flaws are selected directly influences the produced search space. Hence, several papers address the problem of finding appropriate flaw selection functions in POCL planning [12,13,18,20], but none is tailored to the selection of soft goals. The specification and empirical evaluation of flaw selection strategies for POCL systems in the presence of soft flaws is thus one of our main contributions.

The question *how* to resolve soft goals rises from two sources: First, a soft flaw does not *need* to be resolved and must consequently be treated differently than a hard flaw. Second, since soft flaws are represented as a DNF of literals in this work, many standard flaw selection strategies are not applicable anymore, since they are based on flaws being single literals.

4 Soft Flaw Resolution Strategies

The function $\texttt{resolveFlaw}(N, f)$ in line 9 of Algorithm 1 calculates all successor plans that resulted from addressing and/or resolving the previously selected flaw f in the plan P of the current search node $N = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$.

An important observation when planning in the presence of soft flaws is that it is not sufficient to alter the POCL solution criterion from "there are no flaws" to "there are no hard flaws" (cf. line 5). Note that the flaw selection is not a backtrack point, as in standard POCL planning (without optional goals), *all* flaws need to be resolved and hence the actual order is irrelevant for completeness. Since working on a preference is optional, selecting and resolving a soft might invalidate completeness; hence, one *must* be able to ignore a soft flaw in order to maintain completeness. Thus, given a search node $N = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$ and a soft flaw f, the function $\texttt{resolveFlaw}(N, f)$ must also include $(P, \texttt{flaws}_h(P), \texttt{flaws}_s(P) \setminus \{f\})$. Otherwise, the choice to work on the soft flaw f could have the effect that certain solutions might not be found. Please note that this observation generalizes to POCL planning with soft flaws and is thus not specific to the case in which a preference is an at-end preference in DNF.

Given an unsatisfied preference p in DNF, there are several possibilities how to work on p, i.e., how to define the function $\texttt{resolveFlaw}(N, f)$ for a search node N containing the soft flaw f, which represents p.

We evaluated two approaches called the *split strategy* and the *no-split strategy*. The general idea of the split strategy is to prevent the system from working on different disjuncts within the same preference. The no-split strategy, on the other hand, does allow to work on different disjuncts within the same plan.

4.1 The Split Strategy

The idea behind this strategy is to prevent the search process to work on different disjuncts in order to safe unnecessary search effort, as a preference in DNF is satisfied is a single disjuncts satisfied – protecting literals in other disjuncts is thus unnecessary search effort.

Let $p = \varphi_1 \vee \cdots \vee \varphi_n$ with $\varphi_i = \psi_{i_1} \wedge \cdots \wedge \psi_{i_m}$ be a preference that has never been addressed before, i.e., the corresponding soft flaw f has never been selected by the flaw selection function $\texttt{flawSel}$. Let $N = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$ and $f \in \texttt{flaws}_s(P)$.

When f is selected the first time, $\texttt{resolveFlaw}(N, f) = \{N', N_1, \ldots N_n\}$, where $N' = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P) \setminus \{f\})$ for the reason of completeness, and $N_i = (P_i, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$, and P_i being P with p being set to φ_i.

When f is selected, but not for the first time, $\texttt{resolveFlaw}(N, f)$ produces all possibilities to protect an unprotected literal of the conjunction $\varphi_i = \psi_{i_1} \wedge \cdots \wedge \psi_{i_m}$. The literal to be protected is chosen by the flaw selection function defined in the next section. The flaw f is removed from the current plan under consideration as soon as every literal in φ_i is protected by a causal link.

4.2 The No-Split Strategy

As opposed to the last strategy, this one does allow to work different disjuncts within the same preference. While this strategy does allow for plans containing redundant plan steps and causal links, it shows advantages in terms of flexibility.

Let p be in DNF and f be the corresponding soft flaw. When we do not perform splitting into different conjunctions, $N' = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P) \setminus \{f\}) \in \texttt{resolveFlaw}(N, f)$ for $N = (P, \texttt{flaws}_h(P), \texttt{flaws}_s(P))$ and f being selected for the first time is not a sufficient criterion for completeness. Although it is still a necessary criterion, we also have to take into account p's disjunctive structure as follows: Let f and, in particular, the literal ψ_{i_j} be selected by the flaw selection function. Now, we distinguish two cases: Either some literal in the disjunct φ_i in which ψ_{i_j} occurs[1] was already selected before or ψ_{i_j} is the first literal selected in φ_i. In the former case, $\texttt{resolveFlaw}(N, f)$ contains all plans resulting from protecting ψ_{i_j}. In the latter case, this set additionally contains P but with p modified by φ_i set to false.

[1] For the sake of simplicity we assume that there is exactly one disjunct in p containing ψ_{i_j}. The described procedure is easily adapted to the general case.

Example Let $p = \psi_{1_1} \vee (\psi_{2_1} \wedge \psi_{2_2})$ be a preference in the solution plan P with none of its literals being protected by a causal link. Further, let f be the soft flaw representation of p. Let us assume that P can not be developed into a solution if ψ_{2_1} is protected by a causal link, but there are solutions if protecting ψ_{1_1}. Let us further assume the flaw selection function `flawSel` estimates ψ_{2_1} to be the most promising literal hence selecting f and, in particular, ψ_{2_1}. By assumption, none of the successor plans of P can be developed to a solution. However, we lose completeness, since there are solutions if protecting only ψ_{1_1}, but not protecting ψ_{2_1}. To solve that problem it is sufficient to allow ignoring disjuncts in the same way as we allow ignoring a complete preference: When addressing the disjunct $(\psi_{2_1} \wedge \psi_{2_2})$ the first time, we additionally add P' to the search fringe in which p is replaced by $p' = \psi_{1_1}$.

5 Flaw Selection Functions

In line 8 of Algorithm 1, the function `flawSel` selects a flaw from the current set of hard and soft flaws. In our experiments we observed the hard flaw selection strategy *Least-Cost Flaw Repair (LCFR)* [13] being one of the best performing strategies. It always selects a flaw f with a minimal number of refinement options $|\texttt{resolveFlaw}(N, f)|$. When selecting a soft flaw, we followed that idea and implemented a strategy selecting a soft flaw with a minimal estimated branching factor taking into account that each preference is a disjunction (possibly with exactly one disjunct if the split strategy is used) of conjuncts. We call the resulting strategies $\mathit{LCFR}_{\mathit{DNF}}\text{-}\sum$, $\mathit{LCFR}_{\mathit{DNF}}\text{-}\prod$, and $\mathit{LCFR}_{\mathit{DNF}}\text{-}\min$, which estimate the branching factor based on the "cheapest" disjunct of a preference based on summarizing, multiplying or taking the minimal number of refinement options for each literal in that disjunct.

Let $p = \varphi_1 \vee \cdots \vee \varphi_n$ with $\varphi_i = \psi_{i_1} \wedge \cdots \wedge \psi_{i_m}$ and f the soft flaw representing p in plan P. For $\circ \in \{\sum, \prod, \min\}$, we define the preference-based Least-Cost Flaw Repair strategy as follows:

$$\mathit{LCFR}_{\mathit{DNF}}\text{-}\circ\,(N, f) := \min_{\varphi_i} \underset{\psi_{i_j}}{\circ} |\texttt{resolveFlaw}(N, f(\psi_{i_j}))|$$

where $f(\psi_{i_j})$ is the flaw representation of the literal ψ_{i_j} assuming only unprotected literals are taken into account.

Example $\mathit{LCFR}_{\mathit{DNF}}\text{-}\min$ always selects a soft flaw for which there is a literal with a minimal number of supporters. However, it completely ignores the size of the conjuncts. For example, let $p = \psi_{1_1} \vee (\psi_{2_1} \wedge \psi_{2_2})$ and $|\texttt{resolveFlaw}(N, f(\psi))|$ be $2, 1$, and 3 for $\psi := \psi_{1_1}, \psi_{2_1}$, and ψ_{2_2}, respectively. Then, $\mathit{LCFR}_{\mathit{DNF}}\text{-}\min = 1$ selecting its *argmin* ψ_{2_1} ignoring that ψ_{2_2} also has to be supported afterwards in order to fulfill p. The other two strategies try to address that overly optimistic estimate of $\mathit{LCFR}_{\mathit{DNF}}\text{-}\min$ by taking into account *all* conjuncts. In the previous example, $\mathit{LCFR}_{\mathit{DNF}}\text{-}\circ = 2$ for $\circ \in \{\sum, \prod\}$ and the *argmin* ψ_{1_1}.

6 Evaluation

In this section, we evaluate the POCL algorithm using the presented strategies. Our evaluation focuses on three different questions:

– Does one of the two proposed soft-flaw-resolution techniques perform better than the other?
– In which order should hard and soft flaws be addressed, i.e., is it beneficial to work on preferences before a plan is a solution?
– Is there a soft flaw selection strategy that outperforms the others?

6.1 System Configuration

Since we want to maximize the net-benefit, we define our plan selection strategy `planSel` to prefer a plan P with maximal value of $\texttt{netBen(P)} - h(P)$, where $h(P)$ is a heuristic function estimating the action costs necessary to achieve all hard goals of the current partial plan P. To be more precise, h is a variant of the *additive heuristic for POCL planning*, as described by Younes and Simmons [25]. Our heuristic guidance is thus limited to action costs ignoring the benefit of satisfied preferences. Although we developed a heuristic for POCL which takes into account optional goals [4], we did not yet implement that heuristic.

We observed that our plan selection strategy `planSel` selects a *unique* plan in only 45% to 70% of all cases. To avoid a random plan selection among the plans with an identical value of $\texttt{netBen(P)} - h(P)$, we can define an arbitrarily long sequence of tie-breakers. In all experiments, the following sequence was used: *Maximize Benefit (MB)* $\rightarrow$ *Minimize Costs (MC)* $\rightarrow$ *Minimize # Ignored Preferences (MIP).* MB maximizes the sum of the fulfilled preferences' benefit thus ignoring action costs, whereas MC minimizes these costs. *MIP* minimizes the number of ignored preferences to favor plans which still have the potential to fulfill more preferences. If a unique plan was still not found, a plan is picked at random.

The function `flawSel` consists of two parts: a sequence responsible for selecting a hard flaw and a sequence for selecting soft flaws, respectively. The hard flaw selection sequence $\texttt{flawSel}_h$ is always given by *Prefer Hard Flaw (PHF)* $\rightarrow$ *LCFR* $\rightarrow$ *Earliest Flaws (EF)* $\rightarrow$ *Select Hard Flaw (SHF). PHF* always favors a hard flaw before a soft flaw. As a consequence, a soft flaw can only be selected by a subsequent flaw selection if the current plan has no more hard flaws. *LCFR* minimizes the branching factor as explained in the previous section. *EF* favors flaws which were introduced earlier to a plan and *SHF* finally ensures that some (random) hard flaw is selected if the choice is still invariant. For the selection of soft flaws, we use the flaw selection sequence $\texttt{flawSel}_s$ given by *Prefer Soft Flaw (PSF)* $\rightarrow$ $LCFR_{DNF}$-$\circ$ $\rightarrow$ *Select Soft Flaw (SHF). PSF* and *SSF* behave exactly as *PHF* and *SHF*, but for soft instead of hard flaws. $LCFR_{DNF}$-$\circ$ is one of the three strategies as described in the last section.

In the empirical evaluation, we tested the system configuration with the `planSel` function as described above and all of the following combinations:

– For f being a soft flaw, $\texttt{resolveFlaw}(P, f)$ implements the *Split Strategy* or the *No-Split Strategy.*
– The flaw selection strategy `flawSel` is one of the sequences $\texttt{flawSel}_{hs} := \texttt{flawSel}_h \rightarrow \texttt{flawSel}_s$ and $\texttt{flawSel}_{sh} := \texttt{flawSel}_s \rightarrow \texttt{flawSel}_h$

Note that there are actually six flaw selection sequences, as each soft flaw selection sequence contains the $\mathit{LCFR}_{\mathit{DNF}}$-$\circ$ strategy which is parametrized by $\circ \in \{\sum, \prod, \min\}$. In total, we thus evaluated twelve different configurations for every problem instance.

6.2 Benchmarks and Empirical Results

To compare our proposed strategies we evaluated two very different domains.

The first domain is a randomly generated domain to obtain a very large set of problem instances containing many preferences. The problems specify a goal description (which is not mandatory in preference-based planning) and 100 preferences consisting of 2 to 5 disjuncts, each of which being a conjunction of size 2 to 6.

The second domain is from an ongoing research project in which we want to assist a human user in every-day life situations like making appointments and going shopping. The domain is still rather small and only a few problem instances are modeled, yet. Each of them specifies certain mandatory goals as well as between 5 and 12 preferences in DNF.

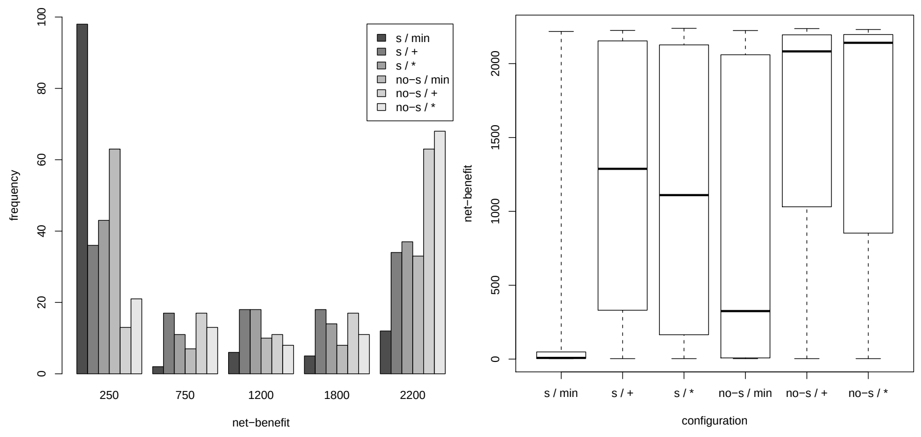
Random Domain In the random domain, we evaluated 120 problem instances and report the net-benefit of the best plan found by each configuration within a search space limit of 10,000 plans.

We visualize our results by means of a histogram (Fig. 1a) and a boxplot (Fig. 1b). Please note that we only include 6 of the 12 tested configurations in the diagrams because we observed one configuration parameter to cause the search to fail consistently: In only two problem instances our system was able to find a solution if the flaw selection sequence $\texttt{flawSel}_{sh}$ is chosen, in which soft flaws are resolved first. This perfectly meets our expectations, as it is more important to have any valid solution than to have a plan that satisfies many preferences but does not respect the solution criteria.

Comparing the remaining configurations, we clearly notice that using the split strategy in combination with $\mathit{LCFR}_{\mathit{DNF}}$ - min is dominated by all other configurations (Fig. 1b). In almost 100 of all 120 instances it produced only plans with a net-benefit of up to 250, the mean being close to zero and 2200 being the maximal net-benefit achieved by any configuration/problem instance combination (Fig. 1a). This result is not surprising, since further investigation shows that both the split strategy as well as the $\mathit{LCFR}_{\mathit{DNF}}$ - min strategy perform worse than their respective counterparts.

When we compare the different versions of the soft flaw selection strategy, the results clearly show that $\mathit{LCFR}_{\mathit{DNF}}$ - min is dominated by $\mathit{LCFR}_{\mathit{DNF}}$ - $\sum$ and $\mathit{LCFR}_{\mathit{DNF}}$ - $\prod$ (Fig. 1b). While the latter two strategies mostly find solutions of similar quality (Fig. 1a), the respective difference between these two strategies and $LCFR_{DNF}$-min is quite large. This observation is quite plausible as the $LCFR_{DNF}$-min strategy completely ignores the size of the conjunctions upon which it bases its decision.

The last remaining comparison is the one between the two flaw resolution strategies split and no-split. We observe that the no-split strategy clearly dominates the split strategy. We do not know the cause of that behavior and regard this result as the most interesting one. A possible cause of the advantage of the split strategy might be its flexibility to switch to some disjunct after the planner already worked on another. Consider the following example for clarification: Let P be a solution and $p = \psi_{1_1} \vee (\psi_{2_1} \wedge \psi_{2_2})$ one of its preferences. Further, let ψ_{2_2} be the literal selected by the flaw selection function. Let us assume protecting ψ_{2_2} introduced new hard flaws which need be to be resolved first before the planning system can continue working on p. Let P' be the resulting solution in which ψ_{1_1} and ψ_{2_1} are still unprotected. Assuming that no solution exists if ψ_{2_1} is protected by a causal link, the only possibility for the split strategy to protect ψ_{1_1} is to refine P which might take longer than refining P' or a plan along the path from P to P'.



(a) Histogram for net-benefit.

(b) Boxplot for mean value and distribution of net-benefit.

Fig. 1: Fig. 1a shows the impact of the chosen configurations on plan quality while Fig. 1b shows the distribution and mean value of the net-benefit of the best solution found by the different configurations. s stands for the split strategy, *no-s* for the no-split strategy and *min*, $+$, and $*$ stand for corresponding $LCFR_{DNF}$-$\circ$ soft flaw selection strategies, $\circ$ being min, $\sum$, and $\prod$, respectively.

User Assistance Domain Because our domain models and corresponding problem instances are quite small, optimal solutions can be found very fast. Thus, we set a timeout of only 1.5 minutes and report the best net-benefit of any solution and the current search space size when that solution was found. Tab. 1 shows the results obtained by two runs taking the mean values.

Table 1: This table shows the impact of the different configurations (rows) on different problem instances (columns) in the user assistance domain. `netBen` denotes the net benefit of the best found solution and SS the size of the search space when it was found. Configurations are named as in Fig. 1. hs stands for the flaw selection function $\texttt{flawSel}_{hs}$ and sh for $\texttt{flawSel}_{sh}$.

		#1		#2		#3		#4		#5	
		`netBen`	SS	`netBen`	SS	`netBen`	SS	`netBen`	SS	`netBen`	SS
s/min	hs	25	64194	10	786	7	184	2	5079	31	121
	sh	-42	1	0	1	7	294	0	1	0	1
s/+	hs	26	15908	20	658	12	272687	2	2223	31	125
	sh	-42	1	0	1	7	153	0	1	0	1
s/*	hs	13	177	13	4705	12	418678	2	2223	31	123
	sh	-42	1	0	1	7	133	0	1	0	1
no-s/min	hs	28	1466	10	144	9	82561	2	1768	31	619
	sh	-42	1	0	1	7	76	0	1	15	285942
no-s/+	hs	17	134	20	442	7	94	2	1215	31	701
	sh	-42	1	0	1	7	100	0	1	0	1
no-s/*	hs	32	1776	15	243	7	82901	2	1215	31	116
	sh	-42	1	0	1	7	60	0	1	0	1

In the evaluation of the previous domain we were able to identify five very clear observations. The results of the user assistance domain do not reproduce these observations that clearly. In this domain, almost all configurations can find optimal solutions; there are only a few deviations which can not justify any conclusion.

However, our results do not *contradict* the previous observations as well, since all configuration comparisons result more or less in a tie. Furthermore, there is one observation that we can confirm very clearly. Preferring soft flaws before hard flaws (strategy $\texttt{flawSel}_{sh}$) is strongly dominated by the reverse ordering. In almost all cases the best quality is significantly worse compared to the opposite ordering. There is only one instance (cf. instance 3, last configuration), in which $\texttt{flawSel}_{sh}$ explores less nodes than $\texttt{flawSel}_{hs}$.

7 Conclusion & Future Work

In this paper, we introduced a POCL algorithm capable of solving problems with (simple) preferences while optimizing their net-benefit. For selecting and satisfying a preference during planning, we developed three novel flaw selection functions, which are based on a successful strategy known from (standard) POCL planning without preferences. Furthermore, we addressed the question of *how* to address preferences when they are represented in terms of a disjunctive normal form. We evaluated these strategies empirically on two different domains.

Our empirical evaluation is still preliminary: So far, the plan selection strategy does not take into account the preferences; future work will thus include an empirical evaluation with a plan selection function guided by the preferences. Furthermore, we did not evaluate problems from the IPC which also remains future work.

Acknowledgements

This work is done within the Transregional Collaborative Research Centre SFB/ TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

1. Bahadori, S., Cesta, A., Iocchi, L., Leone, G., Nardi, D., Pecora, F., Rasconi, R., Scozzafava, L.: Towards ambient intelligence for the domestic care of the elderly. In: Remagnino, P., Foresti, G., Ellis, T. (eds.) Ambient Intelligence, pp. 15–38. Springer New York (2005)
2. Baier, J.A., Bacchus, F., McIlraith, S.A.: A heuristic search approach to planning with temporally extended preferences. Artificial Intelligence 173, 593–618 (2009)
3. Benton, J., Do, M., Kambhampati, S.: Anytime heuristic search for partial satisfaction planning. Artificial Intelligence 173, 562–592 (2009)
4. Bercher, P., Biundo, S.: A heuristic for hybrid planning with preferences. In: Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2012). pp. 120–123. AAAI Press (2012)
5. Bidot, J., Biundo, S.: Artificial intelligence planning for ambient environments. In: Heinroth, T., Minker, W. (eds.) Next Generation Intelligence Environments - Ambient Adaptive Systems, chap. 6, pp. 195–225. Springer, 1 edn. (2011)
6. Biundo, S., Bercher, P., Geier, T., Müller, F., Schattenberg, B.: Advanced user assistance based on AI planning. Cognitive Systems Research 12(3-4), 219–236 (2011), special Issue on Complex Cognition
7. Cesta, A., Cortellessa, G., Pecora, F., Rasconi, R.: Supporting interaction in the robocare intelligent assistive environment. In: Proceedings of AAAI Spring Symposium on Interaction Challenges for Intelligent Assistants. pp. 18–25. AAAI (2007)
8. Coles, A., Coles, A.: LPRPG-P: Relaxed plan heuristics for planning with preferences. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011). pp. 26–33 (2011)
9. Edelkamp, S.: On the compilation of plan constraints and preferences. In: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006). pp. 374–377. AAAI Press (2006)

10. Edelkamp, S., Kissmann, P.: Optimal symbolic planning with action costs and preferences. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1690–1695. AAAI Press (July 2009)
11. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. Artificial Intelligence 173(5-6), 619–668 (2009)
12. Gerevini, A., Schubert, L.K.: Accelerating partial-order planners: Some techniques for effective search control and pruning. Journal of Artificial Intelligence Research (JAIR) 5, 95–137 (1996)
13. Joslin, D., Pollack, M.E.: Least-cost flaw repair: A plan refinement strategy for partial-order planning. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994). pp. 1004–1009 (1994)
14. Keyder, E., Geffner, H.: Soft goals can be compiled away. Journal of Artificial Intelligence Research 36, 547–556 (2009)
15. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: ESWC'08: Proceedings of the 5th European Semantic Web Conference. pp. 629–643. Springer, Berlin, Heidelberg (2008)
16. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991). pp. 634–639 (1991)
17. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In: Proceedings of the third International Conference on Knowledge Representation and Reasoning. pp. 103–114 (1992)
18. Pollack, M.E., Joslin, D., Paolucci, M.: Flaw selection strategies for partial-order planning. Journal of Artificial Intelligence Research (JAIR) 6, 223–262 (1997)
19. Pollack, M.: Planning technology for intelligent cognitive orthotics. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002). pp. 322–331 (2002)
20. Schattenberg, B., Bidot, J., Biundo, S.: On the construction and evaluation of flexible plan-refinement strategies. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) Advances in Artificial Intelligence, Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007). Lecture Notes in Artificial Intelligence, vol. 4667, pp. 367–381. Springer, Osnabrck, Germany (September 2007)
21. Schattenberg, B., Bidot, J., Geßler, S., Biundo, S.: A framework for interactive hybrid planning. In: Mertsching, B., Hund, M., Aziz, Z. (eds.) Advances in Artificial Intelligence, Proceedings of the 32nd German Conference on Artificial Intelligence (KI 2009). Lecture Notes in Artificial Intelligence, vol. 5803, pp. 17–24. Springer (September 2009)
22. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012). pp. 225–233. AAAI Press (6 2012)
23. Smith, D.E.: Choosing objectives in over-subscription planning. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004). pp. 393–401. AAAI Press (2004)
24. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1790–1797. AAAI Press (July 2009)
25. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research (JAIR) 20, 405–430 (2003)

Combining Symbolic with Bitvector Memory-Limited Search for Solving Games

Stefan Edelkamp Martha Rohte
Institute for Artificial Intelligence (IAI)
Universität Bremen

Peter Kissmann
Foundations of Artificial Intelligence (FAI) Group
Universität des Saarlandes

This work combines recent advances in combinatorial search under memory limitation, namely bitvector and symbolic search. Bitvector search assumes a bijective mapping between state and memory addresses, while symbolic search compactly represents state sets. The memory requirements vary with the structure of the problem to be solved.

The integration of the two algorithms into one hybrid algorithm for strongly solving general games initiates a BDD-based solving algorithm, which consists of a forward computation of the reachable state set, possibly followed by a layered backward retrograde analysis. If the main memory becomes exhausted, it switches to explicit-state two-bit retrograde search.

We use the classical game of *Connect Four* as a case study, and solve some instances of the problem space-efficiently with the proposed hybrid search algorithm.

1 Introduction

This works combines recent advances in AI search under memory limitation, namely *bitvector search* that has been successfully applied in solving, e.g., *Pancake Flipping* [15], and *Rubik's Cube*, *Chinese Checkers* [21], *Nine-Men-Morris* [13], *Awari* [19], or *Checkers* endgames [20]; as well as *symbolic search*, which has been successfully applied in solving *cost-optimal AI planning* problems [9] and strongly solving *general games* [14].

Bitvector search [6] assumes a perfect hash function, in form of a one-to-one mapping between state and memory address indices, so that each state only takes a constant number of bits (the state itself is implicitly encoded in the memory address), while symbolic search [18] compactly represents and progresses state sets usually represented in form of BDDs [5].

When addressing practical needs in the applications, the memory requirements for explicit-state space search based on a bitvector encoding of the state space, and for a BDD-based symbolic state space traversal based on a binary encoding of a state, vary with the structure of the problem to be solved. The sweet spot of efficient space usage is mostly between the two extremes.

We will address the integration of the two algorithms into a single hybrid combined explicit-state and symbolic search algorithm for strongly solving general games described in the game description language GDL [17]. First, it initiates a BDD-based solving algorithm, which consists of a forward computation of the reachable state set, followed by a layered backward retrograde analysis. Based on the players' turn, for general two-player games the state space can become twice as large as the number of possible game positions. As the state vector for the first player and the second player to move are interleaved, the player's turn can be found by looking at the mod 2 value of a state's rank.

Submitted to:
GRAPHITE 2013

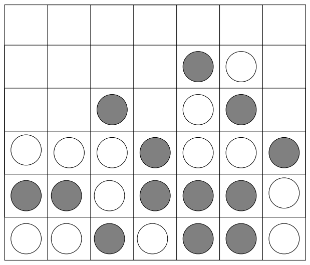


Figure 1: The Game *Connect Four*: The Player with the Pieces Shaded in Gray has Won.

For the BDD exploration and solving we will apply the layered approach that is described in [14]. After storing BDDs for each layer of the set of reachable states, the solving algorithm chains backward layer by layer, with decreasing distance to the initial state instead of increasing distance to the goal states. This way, the BDDs for all but the currently addressed layers can be flushed to and read from disk.

For the explicit-state space analysis we will use a bitvector-based retrograde analysis algorithm. We achieve the connection from the explicit to the symbolic representation obtained in the reachability analysis by applying efficient ranking and unranking with BDDs [7].

If the main memory requirements become exhausted, the hybrid algorithm switches to explicit-state two-bit backward state space search. The BDD representation of the forward-layer that is currently worked on in the retrograde analysis serves as a perfect hash function to address the index in the bitvector with the state and to retrieve and reconstruct it from the index.

We use *Connect Four* as a case study, and solve some instances of the problem space-efficiently with the proposed hybrid search algorithm. Moreover, we predict the search efforts needed for strongly solving larger *Connect Four* problems with the approach.

The paper is structured as follows. First, we motivate the problem of limited main memory capacity that we encountered, while trying to strongly solve the game of *Connect Four* with symbolic search. Next, we introduce bitvector-based search as used in combinatorial games as well as ranking and unranking with BDDs. We then show how to combine the algorithm into one hybrid search strategy. Finally, we provide initial experiments in smaller instances of the *Connect Four* problem, predict the search efforts for solving larger instances, discuss the outcome and conclude.

2 Case Study: Connect Four

Although most of the algorithms are applicable to most two-player games, our focus is on one particular case, namely the game *Connect Four* (see Figure 1). The game [2, 1] is played on a grid of c columns and r rows. In the classical setting we have $c = 7$ and $r = 6$. While the game is simple to follow and play, it can be rather challenging to win. This game is similar to *Tic-Tac-Toe*, with two main differences: The players must connect four of their pieces (horizontally, vertically, or diagonally) in order to win and gravity pulls the pieces always as far to the bottom of the chosen column as possible. The number of states for different settings of $c \times r$ is shown in Table 1.

BDD search can efficiently execute a breadth-first enumeration of the state space in (7×6) *Connect Four* [8]. Table 2 displays the exploration results of the search (derived on an Intel Xeon X5690 CPU with 3.47 GHz and 192 GB RAM – forward search takes less than 16 GB). It has been formally shown that – while the reachable set leads to polynomially-sized BDDs – the symbolic representation of the

Table 1: Number of Reachable States for Various *Connect Four* Instances.

Layer	7×6	6×6	6×5	5×6	5×5
0	1	1	1	1	1
1	7	6	6	5	5
2	49	36	36	25	25
3	238	156	156	95	95
4	1,120	651	651	345	345
5	4,263	2,256	2,256	1,075	1,075
6	16,422	7,876	7,870	3,355	3,350
7	54,859	24,330	24,120	9,495	9,355
8	184,275	74,922	72,312	26,480	25,060
9	558,186	211,042	194,122	68,602	60,842
10	1,662,623	576,266	502,058	169,107	139,632
11	4,568,683	1,468,114	1,202,338	394,032	299,764
12	12,236,101	3,596,076	2,734,506	866,916	596,136
13	30,929,111	8,394,784	5,868,640	1,836,560	1,128,408
14	75,437,595	18,629,174	11,812,224	3,620,237	1,948,956
15	176,541,259	39,979,044	22,771,514	6,955,925	3,231,341
16	394,591,391	80,684,814	40,496,484	12,286,909	4,769,837
17	858,218,743	159,433,890	69,753,028	21,344,079	6,789,890
18	1,763,883,894	292,803,624	108,862,608	33,562,334	8,396,345
19	3,568,259,802	531,045,746	165,943,600	51,966,652	9,955,530
20	6,746,155,945	884,124,974	224,098,249	71,726,433	9,812,925
21	12,673,345,045	1,463,364,020	296,344,032	97,556,959	9,020,543
22	22,010,823,988	2,196,180,492	338,749,998	116,176,690	6,632,480
23	38,263,228,189	3,286,589,804	378,092,536	134,736,003	4,345,913
24	60,830,813,459	4,398,259,442	352,607,428	132,834,750	2,011,598
25	97,266,114,959	5,862,955,926	314,710,752	124,251,351	584,249
26	140,728,569,039	6,891,603,916	224,395,452	97,021,801	
27	205,289,508,055	8,034,014,154	149,076,078	70,647,088	
28	268,057,611,944	8,106,160,185	74,046,977	40,708,770	
29	352,626,845,666	7,994,700,764	30,162,078	19,932,896	
30	410,378,505,447	6,636,410,522	6,440,532	5,629,467	
31	479,206,477,733	5,261,162,538			
32	488,906,447,183	3,435,759,942			
33	496,636,890,702	2,095,299,732			
34	433,471,730,336	998,252,492			
35	370,947,887,723	401,230,354			
36	266,313,901,222	90,026,720			
37	183,615,682,381				
38	104,004,465,349				
39	55,156,010,773				
40	22,695,896,495				
41	7,811,825,938				
42	1,459,332,899				
Σ	4,531,985,219,092	69,212,342,175	2,818,972,642	1,044,334,437	69,763,700

termination criterion appears to be exponential [10]. The set of all 4,531,985,219,092 reachable states can be found within a few hours of computation, while explicit-state search took about 10,000 hours.

As illustrated in Table 3, of the 4,531,985,219,092 reachable states *only* 1,211,380,164,911 (about

Table 2: Number of Nodes and States in (7×6) *Connect Four* (l layer, n BDD nodes, s states).

l	n	s
0	85	1
1	163	7
2	316	49
3	513	238
4	890	1,120
5	1,502	4,263
6	2,390	16,422
7	4,022	54,859
8	7,231	184,275
9	12,300	558,186
10	21,304	1,662,623
11	36,285	4,568,683
12	56,360	12,236,101
13	98,509	30,929,111
14	155,224	75,437,595
15	299,618	176,541,259
16	477,658	394,591,391
17	909,552	858,218,743
18	1,411,969	1,763,883,894
19	2,579,276	3,568,259,802
20	3,819,845	6,746,155,945
21	6,484,038	12,673,345,045
22	9,021,770	22,010,823,988
23	14,147,195	38,263,228,189
24	18,419,345	60,830,813,459
25	26,752,487	97,266,114,959
26	32,470,229	140,728,569,039
27	43,735,234	205,289,508,055
28	49,881,463	268,057,611,944
29	62,630,776	352,626,845,666
30	67,227,899	410,378,505,447
31	78,552,207	479,206,477,733
32	78,855,269	488,906,447,183
33	86,113,718	496,636,890,702
34	81,020,323	433,471,730,336
35	81,731,891	370,947,887,723
36	70,932,427	266,313,901,222
37	64,284,620	183,615,682,381
38	49,500,513	104,004,465,349
39	38,777,133	55,156,010,773
40	24,442,147	22,695,896,495
41	13,880,474	7,811,825,938
42	4,839,221	1,459,332,899
Total		4,531,985,219,092

26.72%) have been left unsolved in the layered BDD retrograde analysis. (More precisely, there are 1,265,297,048,241 states left unsolved by the algorithm, but the remaining set of 53,916,883,330 states in layer 30 is implied by the solvability status of the other states in the layer.)

Even while providing space in form of 192 GB of RAM, however, it was not possible to proceed the symbolic solving algorithm to layers smaller than 30. The reason is while the peak of the solution for the state sets has already been passed, the BDDs for representing the state sets are still growing.

This motivates looking at other options for memory-limited search and a hybrid approach that takes the symbolic information into account to eventually perform the complete solution of the problem.

3 Binary Decision Diagrams for Strongly Solving Games

Binary decision diagrams (BDDs) are a memory-efficient data structure used to represent Boolean functions [5] as well as to perform set-based search [18]. In short, a BDD is a directed acyclic graph with one root and two terminal nodes, the 0- and the 1-sink. Each internal node corresponds to a binary variable and has two successors, one (along the *Then*-edge) representing that the current variable is true (1) and the other (along the *Else*-edge) representing that it is false (0). For any assignment of the variables derived from a path from the root to the 1-sink the represented function will be evaluated to 1.

Bryant [5] proposed using a fixed variable ordering, for which he also provided two reduction rules (eliminating nodes with the same Then and Else successors and merging two nodes representing the same variable that share the same Then successor as well as the same Else successor). These BDDs are called reduced ordered binary decision diagrams (ROBDDs). Whenever we mention BDDs in this paper, we actually refer to ROBDDs. We also assume that the variable ordering is the same for all the BDDs and

Table 3: Result of Symbolic Retrograde Analysis (excl. terminal goals, *l* layer, *n* BDD nodes, *s* states).

l	*n* (won)	*s* (won)	*n* (draw)	*s* (draw)	*n* (lost)	*s* (lost)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
29	o.o.m.	o.o.m.	o.o.m.	o.o.m.	o.o.m.	o.o.m.
30	589,818,676	199,698,237,436	442,186,667	6,071,049,190	o.o.m.	o.o.m.
31	458,334,850	64,575,211,590	391,835,510	7,481,813,611	600,184,350	201,906,000,786
32	434,712,475	221,858,140,210	329,128,230	9,048,082,187	431,635,078	57,701,213,064
33	296,171,698	59,055,227,990	265,790,497	10,381,952,902	407,772,871	194,705,107,378
34	269,914,837	180,530,409,295	204,879,421	11,668,229,290	255,030,652	45,845,152,952
35	158,392,456	37,941,816,854	151,396,255	12,225,240,861	231,007,885	132,714,989,361
36	140,866,642	98,839,977,654	106,870,288	12,431,825,174	121,562,152	24,027,994,344
37	68,384,931	14,174,513,115	72,503,659	11,509,102,126	105,342,224	57,747,247,782
38	58,428,179	32,161,409,500	44,463,367	10,220,085,105	42,722,598	6,906,069,443
39	19,660,468	2,395,524,395	27,201,091	7,792,641,079	35,022,531	13,697,133,737
40	17,499,402	4,831,822,472	13,858,002	5,153,271,363	8,233,719	738,628,818
41	0	0	5,994,843	2,496,557,393	7,059,429	1,033,139,763
42	0	0	0	0	0	0

has been optimized prior to the search.

BDDs have been shown to be very effective in the verification of hard- and software systems, where BDD traversal is referred to as symbolic model checking [18]. Adopting terminology to state space search, we are interested in the *image* of a state set S with respect to a transition relation *Trans*. The result is a characteristic function of all states reachable from the states in S in one step.

For the application of the image operator we need two sets of variables, one, x, representing the current state variables, another, x', representing the successor state variables. The image *Succ* of the state set S is then computed as $Succ(x') = \exists x\ (Trans(x,x')\ \wedge\ S(x))$. The *preimage Pre* of the state set S is computed as $Pre(x) = \exists x'\ (Trans(x,x')\ \wedge\ S(x'))$ and results in the set of predecessor states.

Using the image operator, implementing a layered symbolic breadth-first search (BFS) is straightforward. All we need to do is to apply the image operator to the initial state resulting in the first layer, then apply the image operator to the first layer resulting in the second and so on. The search ends when no successor states can be found. General games (and in this case, *Connect Four*) are guaranteed to terminate after a finite number of steps, so that the forward search will eventually terminate as well.

For strongly solving two-player games [14], we find all the reachable states by performing layered symbolic BFS, storing all layers separately. The solving starts in the last reached layer and performs regression search towards the initial state, which resides in layer 0. The last reached layer contains only terminal states (otherwise the forward search would have progressed farther), which can be solved immediately by calculating the conjunction with the BDDs representing the rewards for the two players. Once this is done, the search continues in the preceding layer. If another layer contains terminal states as well, these are solved in the same manner before continuing with the remaining states of that layer. The rewards are handled in a certain order: In case of a zero-sum game the order is always according to the highest reward of the active player, modeling the *MiniMax* procedure [23]. All the solved states of the successor layer are loaded in this order and the preimage is calculated, which results in those states of the current layer that will achieve the same rewards, so that they can be stored on the disk as well. Once the initial state is solved and stored the strong solution resides completely on the hard disk.

4 Ranking and Unranking with BDDs

A *rank* is a unique number of a state and the inverse process of *unranking* reconstructs the state given its rank. Perfect hash functions to efficiently rank and unrank states have been shown to be very successful in traversing single-player problems like *Rubik's Cube* or the *Pancake Problem* [15] or two-player games like *Awari* [19]. They are also used for creating pattern databases [4]. The problem of the construction of perfect hash functions for algorithms like *two-bit breadth-first search* is that they are problem-dependent.

The approach exploited in this paper builds on top of findings by [7], who illustrated that ranking and unranking of states in a state set represented as a BDD is available in time linear to the length of the state vector (in binary). In other words, BDD ranking aims at the symbolic equivalent of constructing a perfect hash function in explicit-state space search [3]. For the construction of the perfect hash function, the underlying state set to be hashed is generated in advance in form of a BDD. This is plausible when computing strong solutions to problems, where we are interested in the game-theoretical value of all reachable states. Applications are, e.g., endgame databases or planning tasks where the problem to be solved is harder than computing the reachability set.

The $\mathit{index}(n)$ of a BDD node n is its unique position in the shared representation and $\mathit{level}(n)$ its position in the variable ordering. Moreover, we assume the 1-sink to have index 1 and the 0-sink to have index 0. Let $C_f = |\{a \in \{0,1\}^n \mid f(a) = 1\}|$ denote the number of satisfying assignments (*satcount*, here also *sc* for short) of f. With *bin* (and *invbin*) we denote the *conversion* of the binary value of a bitvector (and its inverse). The *rank* of a satisfying assignment $a \in \{0,1\}^n$ is the position in the lexicographical ordering of all satisfying assignments, while the *unranking* of a number r in $\{0,\ldots,C_f - 1\}$ is its inverse.

Figure 2 shows the ranking and unranking functions in pseudo-code. The procedures determine the rank given a satisfying assignment and vice versa. They access the satcount values on the Else-successor of each node (adding for the ranking and subtracting in the unranking). Missing nodes (due to BDD reduction) have to be accounted for by their binary representation, i.e., gaps of l missing nodes are accounted for 2^l. While the ranking procedure is recursive the unranking procedure is not.

The satcount values of all BDD nodes are precomputed and stored along with the nodes. As BDDs are reduced, not all variables on a path are present but need to be accounted for in the satcount procedure. The time (and space) complexity of it is $O(|G_f|)$, where $|G_f|$ is the number of nodes of the BDD G_f representing f. With the precomputed values, rank and unrank both require linear time $O(n)$, where n is the number of variables in the function represented in the BDD. [7] provide invariances showing that the procedures work correctly.

4.1 Ranking and Unranking Examples

To illustrate the ranking and unranking procedures, take the example BDD given in Figure 3. Assume we want to calculate the rank of state $s = 110011$. The rank of s is then

$$\begin{aligned}
\mathit{rank}(s) = 0 + rA(v_{13},s) - 1 &= (2^{1-0-1} \cdot sc(v_{11}) + 0 + rA(v_{16},s)) - 1 \\
&= sc(v_{11}) + (2^{3-1-1} \cdot sc(v_8) + \mathit{bin}(0) \cdot sc(v_9) + rA(v_9,s)) - 1 \\
&= sc(v_{11}) + 2sc(v_8) + (0 + rA(v_5,s)) - 1 \\
&= sc(v_{11}) + 2sc(v_8) + (2^{6-4-1} \cdot sc(v_0) + \mathit{bin}(1) \cdot sc(v_1) + rA(v_1,s)) - 1 \\
&= sc(v_{11}) + 2sc(v_8) + 2sc(v_0) + sc(v_1) + 1 - 1 \\
&= 14 + 2 \cdot 5 + 2 \cdot 0 + 1 + 1 - 1 = 25
\end{aligned}$$

```
rank(s)
  i = level(root);
  d = bin(s[0..i-1]);
  return d*sc(root) + rankAux(root,s) - 1;

rankAux(n, s)
  if (n <= 1) return n;
  i = level(n);
  j = level(Else(n));
  k = level(Then(n));
  if (s[i] == 0)
    return bin(s[i+1..j-1]) * sc(Else(n))
      + rankAux(Else(n),s);
  else
    return 2^(j-i-1) * sc(Else(n))
      + bin(s[i+1..k-1]) * sc(Then(n))
      + rankAux(Then(n),s);
```

```
unrank(r)
  i = level(root);
  d = r / sc(root);
  s[0..i-1] = invbin(d);
  n = root;
  while (n > 1)
    r = r mod satCount(n);
    j = level(Else(n));
    k = level(Then(n));
    if (r < (2^(j-i-1) * sc(Else(n))))
      s[i] = 0;
      d = r / sc(Else(n));
      s[i+1..j-1] = invbin(d);
      n = Else(n);
      i = j;
    else
      s[i] = 1;
      r = r - (2^(j-i-1) * sc(Else(n)));
      d = r / sc(Then(n));
      s[i+1..k-1] = invbin(d);
      n = Then(n);
      i = k;
  return s;
```

Figure 2: Ranking and Unranking.

with $rA(s, v_i)$ being the recursive call of the rankAux function for state s in node v_i and $sc(v_i)$ the satcount stored in node v_i.

For unranking the state with index 19 ($r = 19$) from the BDD depicted in Figure 3 we get:

- $i = 0, n = v_{13}$: $r = 19 \mod sc(v_{13}) = 19 \mod 30 = 19 \not< 2^{1-0-1} sc(v_{11}) = 14$, thus $s[0] = 1$; $r = r - 2^{1-0-1} sc(v_{11}) = 19 - 14 = 5$
- $i = 1, n = v_{12}$: $r = 5 \mod sc(v_{12}) = 5 \mod 16 = 5 < 2^{3-1-1} sc(v_8) = 2 \cdot 5 = 10$, thus $s[1] = 0$; $s[2] = invbin(r/sc(v_8)) = invbin(5/5) = 1$
- $i = 3, n = v_8$: $r = 5 \mod sc(v_8) = 5 \mod 5 = 0 < 2^{4-3-1} sc(v_4) = 1$, thus $s[3] = 0$
- $i = 4, n = v_4$: $r = 0 \mod sc(v_4) = 0 \mod 1 = 0 \not< 2^{6-4-1} sc(v_0) = 0$, thus $s[4] = 1$; $r = r - 2^{6-4-1} sc(v_0) = 0 - 0 = 0$
- $i = 5, n = v_2$: $r = 0 \mod sc(v_2) = 0 \mod 1 = 0 \not< 2^{7-6-1} sc(v_{12}) = 0$, thus $s[5] = 1$; $r = r - 2^{7-6-1} sc(v_{12}) = 0 - 0$
- $i = 6; n = v_1$: return s (= 101011)

5 Retrograde Analysis on a Bitvector

Two-bit breadth-first search has first been used to enumerate so-called *Cayley Graphs* [6]. As a subsequent result the authors proved an upper bound to solve every possible configuration of *Rubik's Cube* [16]. By performing a breadth-first search over subsets of configurations in 63 hours together with the help of 128 processor cores and 7 TB of disk space it was shown that 26 moves always suffice

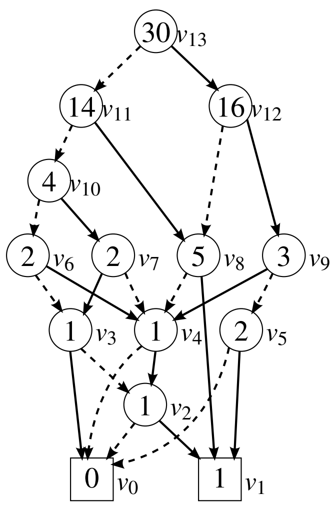


Figure 3: BDD for the ranking and unranking examples. Dashed arrows denote Else-edges; solid ones Then-edges. The numbers in the nodes correspond to the satcount. Each v_i denotes the index (i) of the corresponding node.

to rescramble it. [15] has applied the two-bit breadth-first search algorithm to generate the state spaces for hard instances of the *Pancake* problem I/O-efficiently.

In the two-bit breadth-first search algorithm every state is expanded at most once. The two bits encode values in $\{0,\ldots,3\}$ with value 3 representing an unvisited state, and values 0, 1, or 2 denoting the current search depth *mod* 3. This allows to distinguish generated and visited states from ones expanded in the current breadth-first layer.

5.1 Adaptation to Our Setting

In our implementation (see Algorithm 4) we also use two bits, but with a different meaning. We apply the algorithm to solve two-player zero-sum games where the outcomes are only won/lost/drawn from the starting player's point of view. This is reflected in the interpretation of the two bits: Value 0 means that the state has not yet been evaluated; value 1 means it is won by the starting player (the player with index 0); value 2 means it is won by the player with index 1; value 3 means it is drawn. Retrograde analysis solves the entire set of positions in backward direction, starting from won and lost terminal ones. Bit-state retrograde analysis applies backward BFS starting from the states that are already decided.

For the sake of simplicity, the rank and unrank functions are both context-sensitive wrt. to the layer of the search in which the operations take place. In the implementation we use BDDs for the different layers.

The algorithm assumes a maximal number of moves, that terminal drawn states appear only in the last layer (as is the case in *Connect Four*; extension to different settings is possible), that the game is turn-taking, and that the player can be found in the encoding of the game. It takes as input a decision procedure for determining whether a situation is *won* by one of the players as well as the index of the last reached layer (*maxlayer*). Starting at the final layer, it iterates toward the initial state residing in layer 0.

For each layer, it first of all determines the number of states. Next it sets all values of the vector B for the states in the current state to 0 – not yet solved. Then it iterates over all states in that layer.

It takes one state (by unranking it from the layer), checks whether it is won by one of the players.

```
retrograde(won, maxlayers)
for layer in maxlayers,...,0
  m = sc(bdd(layer))
  for i in 0,...,m - 1
    B[layer][i] = 0
  for i in 0,...,m - 1
    state = unrank(i)
    if (won(state))
      if (layer mod 2 == 1)
        B[layer][i] = 1
      else
        B[layer][i] = 2
    else if (layer == maxlayer)
      B[layer][i] = 3
    else
      succs = expand(state)
      process(succs)
```

```
process (succs)
  if (layer mod 2 == 1)
    for all s in succs
      if B[layer+1][rank(s)] == 2
        B[layer][rank(i)] = 2
        break
      else if (B[layer+1][rank(s)] == 3)
        B[layer][rank(i)] = 3
    if (B[layer][rank(i)] == 0)
      B[layer][rank(i)] = 1
  else
    for all s in succs
      if B[layer+1][rank(s)] == 1
        B[layer][rank(i)] = 1
        break
      else if (B[layer+1][rank(s)] == 3)
        B[layer][rank(i)] = 3
    if (B[layer][rank(i)] == 0)
      B[layer][rank(i)] = 2
```

Figure 4: Retrograde Analysis with Bits for Two-Player Zero-Sum Game (rank and unrank are sensitive to the layer they are called in).

If so, it can be solved correspondingly (setting its value to either 1 or 2). Otherwise, if it resides in the final layer, it must be a drawn state (value 3). In case neither holds, we calculate the state's successors. For each successor we check whether it is won by the currently active player, which is determined by checking the current layer's index. In this case the state is assigned the same value and we continue with the next state. Otherwise if the successor is drawn, the value of this state is set to draw as well. In the end, if the state is still unsolved that means that all successors are won by the opponent, so that the corresponding value is assigned to this state as well.

6 Hybrid Algorithm

The hybrid algorithm combines the two precursing approaches. It generates the state space with symbolic forward search on disk and subsequently applies explicit-state retrograde analysis based on the results in form of the BDD encoded layers read from disk.

Figure 5 illustrates the strong solution process. On the right hand side we see the bitvector used in retrograde analysis and on the left hand side we see the BDD generated in forward search and used in backward search.

The process of solving one layer is depicted in Figure 5 (right). While the bitvector in the layer n (shown at the bottom of the figure) is scanned and states within the layer are unranked and expanded, existing information on the solvability status of ranked successor states in the subsequent layer $n+1$ is retrieved.

Ranking and unranking wrt. the BDD is executed to look up the status (won/lost/drawn) of a node in the set of successors. We observed that there is a trade-off for evaluating immediate termination. There are two options, one is procedural by evaluating the goal condition directly on the explicit state, the other is a dictionary lookup by traversing the corresponding reward BDD. In our case of *Connect Four* the latter was not only more general but also faster. A third option would be to determine if there are any

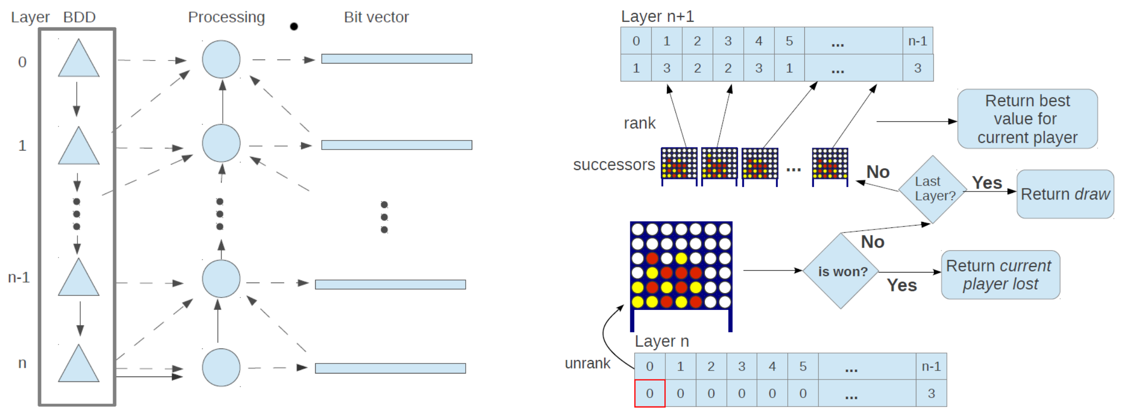


Figure 5: Hybrid Algorithm: Visualization of Data Flow in the Strong Solution Process (left). Processing a Layer in the Retrograde Analysis (right).

successors and set the rewards according to the current layer (as it is done in the pseudo-code).

To increase the exploration performance of the system we distributed the explicit-state solving algorithms on multiple CPU cores. We divide the bitvector for the layer to be solved into equally-sized chunks. The bitvector for the next layer is shared among all the threads.

For the ease of implementation, we duplicate the query BDDs for each individual core. This is unfortunate, as we only use concurrent read in the BDD for evaluating the perfect hash function but the computation of the rank involves setting and reading local variables and requires significant changes in the BDD package to be organized lock-free. There are distributed usages of BDD libraries, e.g., reported by Christian Stangier documented in the CUDD files, but – up to our knowledge – there is no currently available multi-core version. Recent research work shows some steps into that direction [22], but the status is far from being at library use.

Compared to the size of the bitvector the BDDs for the reachability layers are considerably smaller, so that we can afford re-reading the BDDs for each running thread.

Our prototype currently works with reachability sets from forward symbolic search and a complete explicit-state retrograde analysis. In a subsequent refinement of the implementation the hybrid search algorithm can be extended to start solving using symbolic search and then switch over to explicit search using the stored solutions if the BDD solving cannot finish within certain amounts of RAM.

7 Experiments

The experiments for the hybrid algorithm have been conducted on a Desktop PC with an Intel i7-920 CPU (with 2.66 GHz) with four hyperthreaded cores. It is equipped with 24 GB RAM, which mainly limits the maximum number of elements we could handle (operating system: Ubuntu Linux; Compiler: GNU's `g++` optimized with flag `-O3`). For multi-core parallelization we used pthreads. As indicated above, the BDDs for the (7×6) *Connect Four* have been generated on a much larger cluster computer, running a single Java program via a native interface to CUDD that was compiled and optimized on 64 bits with GNU's `g++`. The program ran on a single core with main memory bounded by 192 GB RAM (details of this machine were given above).

Table 4: Results for (5×5) *Connect Four* with 69,763,699 States, Different Termination Criteria, Single- and Multi-Core.

Algorithm	Time	States/Sec
1 Core Direct Goal Evaluation	283s	246,514
8 Core Direct Goal Evaluation	123s	567,184
1 Core BDD Goal	291s	239,737
8 Core BDD Goal	131s	532,547
1 Core No Succ. BDD	288s	242,235
8 Core No Succ. BDD	127s	549,320

Table 5: Results for (6×5) *Connect Four* with 2,938,430,579 States, Different Termination Criteria, Single- and Multi-Core.

Algorithm	Time	States/Sec
1 Core Direct Goal Evaluation	14,197s	206,975
2 Core Direct Goal Evaluation	7,540s	389,712
8 Core Direct Goal Evaluation	3,510s	837,159
1 Core No Succ. BDD	14,944s	196,629
2 Core No Succ. BDD	7,665s	383,356
8 Core No Succ. BDD	3,600s	816,230

The results of the solving of (5×5) *Connect Four* are shown in Table 4. We see that more cores are helpful to reduce the running time significantly. Moreover, the performance for the three strategies to elevate the goal conditions varies only a little. By a small margin, the direct evaluation that has the lowest memory requirements is best.

For the (5×5) case using a bitvector the total space requirements are 17 MB, while the BDDs take 28 MB. However, when loading the data in RAM we also need the reachability sets in form of a BDD taking 5.4 MB, and the goal BDDs taking 4.7 MB. All in all, we obtain memory needs in the order of 22.4 MB, which still documents a possible saving in memory. The last 4.7 MB could be saved by not using the goal BDDs but rather by evaluation of the condition explicitly

For solving (6×5) *Connect Four* the results are depicted in Table 5. Again, we see that more cores clearly reduce runtime (with two cores by a factor of rougly two; with eight cores by a factor of roughly four – note that our CPU uses Hyperthreading, i.e., it has only four physical cores, so the speedup seems to be linear in the number of used cores). Concerning the goal evaluations, we can see that again the direct evaluation is a bit faster. The third criterion was not tested because it was expected to take more space without any significant speed-up.

Table 6 gives some insight into the actual sizes required by the BDDs and the bitvectors in the solving of (6×5) *Connect Four*. For each forward layer we provide the number of states in that layer, the number of BDD nodes needed to represent them, the size of the BDD representation of all states in that layer (assuming that each node requires 20 bytes), the size of the corresponding bitvector as well as a theoretical estimation of the memory required to solve that layer with the hybrid approach in case of a single-core based setting. The required memory is simply the sum of the BDD sizes of the current and the successor layer (both must be loaded for the ranking and unranking in the algorithm) as well as the sizes of the bitvectors (both must be loaded for the actual solving). Performing pure symbolic search we arrive at a peak node count of 136,001,819. Assuming the same node size of 20 bytes this corresponds

Table 6: Space consumption computing (6×5) *Connect Four*. l is the layer, s the number of states in that layer, n the number of BDD nodes needed to represent it, $size_{bdd}$ the size of the corresponding BDD (assuming 20 Bytes per node), $size_{bv}$ the size of the corresponding bitvector, and mem_{req} the memory needed for solving a layer (i.e., loading the current and successor layer's bitvectors and BDDs) in a single-core setting.

l	s	n	$size_{bdd}$	$size_{bv}$	mem_{req}
0	1	61	1.2KB	1B	3.5KB
1	6	116	2.3KB	2B	6.6KB
2	36	223	4.4KB	9B	12KB
3	156	366	7.2KB	39B	20KB
4	651	637	13KB	163B	34KB
5	2,256	1,080	21KB	564B	57KB
6	7,870	1,702	33KB	1.9KB	96KB
7	24,120	2,793	55KB	5.9KB	171KB
8	72,312	4,772	93KB	18KB	305KB
9	194,122	7,498	146KB	47KB	526KB
10	502,058	10,722	209KB	123KB	964KB
11	1,202,338	17,316	338KB	294KB	1.8MB
12	2,734,506	25,987	508KB	668KB	3.4MB
13	5,868,640	43,898	857KB	1.4MB	6.4MB
14	11,812,224	68,223	1.3MB	2.8MB	12MB
15	22,771,514	122,322	2.3MB	5.4MB	21MB
16	40,496,484	187,493	3.6MB	9.7MB	36MB
17	69,753,028	327,553	6.2MB	17MB	58MB
18	108,862,608	475,887	9.1MB	26MB	89MB
19	165,943,600	769,944	15MB	40MB	127MB
20	224,098,249	1,004,398	19MB	53MB	171MB
21	296,344,032	1,437,885	27MB	71MB	210MB
22	338,749,998	1,656,510	32MB	81MB	242MB
23	378,092,536	2,080,932	40MB	90MB	254MB
24	352,607,428	2,123,251	41MB	84MB	243MB
25	314,710,752	2,294,960	44MB	75MB	210MB
26	224,395,452	2,004,090	38MB	54MB	162MB
27	149,076,078	1,814,442	35MB	36MB	111MB
28	74,046,977	1,257,586	24MB	18MB	64MB
29	30,162,078	789,650	15MB	7.2MB	29MB
30	6,440,532	282,339	5.4MB	1.5MB	6.9MB

to roughly 2.5 GB. Seeing that the highest required size in the hybrid approach is 254 MB the savings become apparent.

Note that the column mem_{req} of Table 6 refers to theortical values under ideal circumstances of a static BDD package allocating only space for the BDD nodes that appear on disk. The BDD package in use, however, has its own memory pool implementation, and, therefore, a significant overhead. Hence, we also computed the practical values (by analyzing VIRT and RES provided in the Unix command top after finalizing a layer). E.g., in layer 23 of the retrograde classification algorithm, we observed the maximal real peak memory requirements of 390 MB (VIRT) and 287 MB (RES)[1].

[1]By inspecting values from top in between two consecutive layers, we detected slightly higher intermediate RAM requirements of 351 MB (RES).

Table 7: Classification of all states in won, draw and lost in (6×6) *Connect Four*.

l	won(black)	draw	won(white)
0	1	0	0
1	6	0	0
2	6	24	6
3	98	52	6
4	131	220	300
5	1,324	534	398
6	1,752	1,580	4,544
7	13,868	3,982	6,480
8	18,640	10,280	46,002
9	118,724	25,104	67,214
10	156,360	56,710	363,196
11	815,366	129,592	523,156
12	1,050,857	267,636	2,277,583
13	4,597,758	565,760	3,231,266
14	5,831,790	1,098,276	11,699,108
15	21,523,754	2,144,618	16,310,672
16	27,021,039	3,911,893	49,751,882
17	83,960,708	7,060,426	68,412,756
18	104,937,956	12,096,840	175,768,828
19	272,162,860	20,210,438	238,672,448
20	339,135,354	32,320,349	512,669,271
21	725,182,660	50,189,136	687,992,224
22	901,278,168	75,033,304	1,219,869,020
23	1,561,655,780	108,518,894	1,616,415,130
24	1,929,105,096	150,351,002	2,318,803,344
25	2,645,054,112	202,034,082	3,015,867,732
26	3,223,332,998	259,072,600	3,409,198,318
27	3,392,753,538	322,390,736	4,318,869,880
28	4,030,760,404	384,569,265	3,690,830,516
29	3,089,884,946	435,398,174	4,469,417,644
30	3,476,328,802	471,148,650	2,688,933,070
31	1,785,392,828	468,490,136	3,007,279,574
32	1,841,291,613	450,464,011	1,144,004,318
33	554,598,782	373,041,874	1,167,659,076
34	502,035,320	276,221,222	219,995,950
35	54,244,612	149,951,066	197,034,676
36	40,044,990	49,981,730	0

The CPU time used to process all 2,938,430,579 states of (6×5) *Connect Four* is about 3,500 seconds. That's about 800,000 states per second on average.

On one core, the (6×6) exploration finished in 958,283 seconds, or approx. 11 days. It generated about 72,184.34 states/second. The peak of the real memory requirements was encountered in layers 29 and 28 with 6.5 GB. At the end of the exploration, a 14.5 GB-sized strong solution bitvector database was computed and flushed to disk. The outcome is that the second player wins, validating published results (e.g., by Tromp and Kissmann). Table 7 shows the final classification result. Unfortunately, on our machine we could not finalize the BDD-based solving due to the limited amount of RAM, even though 24 GB were reported to be sufficient for its completion in about 10 hours with the layered approach.

If the solving speed goes down to 200,000 states per second for (6×7) *Connect Four*, solving the entire layer 29 would take about $352{,}626{,}845{,}666/200{,}000/60^2 \approx 488$ hours, or little more than 20 days. For the remaining total of 1,265,297,048,241 states to be solved we estimate a CPU search time of (at least) 1,752 hours or 73 days. In layer 30, the BDD for the reachable states and the two BDDs computed for the solution consume disk space in the order of a few GB only but still take several hours to be loaded. Investing two bits for each state in layer 29 requires $2 \cdot 352{,}626{,}845{,}666$ Bits ≈ 82.1 GB. Therefore, it may be possible for the hybrid search algorithm in this paper to finalize the entire solution process within 192 GB, while the symbolic algorithm could not finish it. However, for an entire retrograde analysis our computer infrastructure is not sufficient to provide sufficient amount of random access memory to finalize the solving experiment for the (7×6) *Connect Four* instance. In layers 32/33 the bitvectors alone would occupy $2 \cdot (488{,}906{,}447{,}183 + 496{,}636{,}890{,}702)$ Bits ≈ 229.5 GB.

Therefore, we looked at the memory profile of a smaller instance, namely the (5×5), (6×5) and the (6×6) *Connect Four* variants. In all cases, we could show moderate savings in RAM in trade-off with a – still acceptable – slow-down in the run-time behavior: the entire BDD exploration for (6×5) (running on a single core of a modern PC) took about 40m, while the hybrid exploration (running on a multi-core PC) lasted for about one hour.

8 Conclusion

Memory-limitation is often a more severe problem to combinatorial search algorithms than computation time. In this paper we have combined two very promising approaches to cope with the problem within main memory, namely the compact representation of state sets in form of a BDD and the implicit representation of states by main memory addresses. Depending on the problem at hand, the one or the other can be advantageous. For cases like *Connect Four*, it is also the case that the memory profile for the one or the other is better in different parts of the search, so that we addressed the problem of how to change from one representation to the other.

The bridge between the explicit and symbolic search is linear-time perfect hashing based on the BDD representation of the state sets encountered. We introduce a hybrid solving algorithm for two-player general games and showed in the case study of *Connect Four* that memory savings are indeed possible. We predicted the space and time needs to finalize the solution for (6×7) *Connect Four* on a contemporary multi-core computer, to give a feasibility assessment on the strong solvability of the game on current technology. Based on the expected significant resources in running time and RAM usagem for the remaining solution, however, we presented experimental results only for smaller *Connect Four* instances.

Wrt. improved BDD exploration one research avenue is to look at problem representations with preconditions and effects, so that improved image operations based on the concept of transition trees apply [12]. Another option is to split the BDD in computing the image based on lex(icographic)-partitioning into equally sized state sets [11].

References

[1] James D. Allen. *The Complete Book of Connect 4: History, Strategy, Puzzles*. Puzzlewright, 2011.

[2] Louis Victor Allis. A knowledge-based approach of connect-four. Master's thesis, Vrije Universiteit Amsterdam, October 1988.

[3] F. C. Botelho, R. Pagh, and N. Ziviani. Simple and space-efficient minimal perfect hash functions. In *WADS*, pages 139–150, 2007.

[4] Teresa M. Breyer and Richard E. Korf. 1.6-bit pattern databases. In *AAAI*, pages 39–44, 2010.

[5] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[6] G. Cooperman and L. Finkelstein. New methods for using Cayley graphs in interconnection networks. *Discrete Applied Mathematics*, 37/38:95–118, 1992.

[7] Martin Dietzfelbinger and Stefan Edelkamp. Perfect hashing for state spaces in BDD representation. In *KI*, pages 33–40, 2009.

[8] Stefan Edelkamp and Peter Kissmann. Symbolic classification of general two-player games. In *KI*, pages 185–192, 2008.

[9] Stefan Edelkamp and Peter Kissmann. Optimal symbolic planning with action costs and preferences. In *IJCAI*, pages 1690–1695, 2009.

[10] Stefan Edelkamp and Peter Kissmann. On the complexity of BDDs for state space search: A case study in Connect Four. In *AAAI*, 2011.

[11] Stefan Edelkamp, Peter Kissmann, and Álvaro Torralba. Lex-partitioning: A new option for BDD search. In *GRAPHite*, pages 66–82, 2012.

[12] Stefan Edelkamp, Peter Kissmann, and Álvaro Torralba. Transition trees for cost-optimal symbolic planning. In *ICAPS*, 2013. To appear.

[13] Stefan Edelkamp, Damian Sulewski, and Cengizhan Yücel. GPU exploration of two-player games with perfect hash functions. In *ICAPS-Workshop on Planning in Games*, 2010.

[14] Peter Kissmann and Stefan Edelkamp. Layer-abstraction for symbolically solving general two-player games. In *SoCS*, pages 63–70, 2010.

[15] Richard E. Korf. Minimizing disk I/O in two-bit breadth-first search. In *AAAI*, pages 317–324, 2008.

[16] D. Kunkle and G. Cooperman. Twenty-six moves suffice for Rubik's cube. In *International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 235 – 242, 2007.

[17] Nathaniel C. Love, Timothy L. Hinrichs, and Michael R. Genesereth. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group, April 2006.

[18] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[19] John W. Romein and Henri E. Bal. Awari is solved. *International Computer Games Association (ICGA) Journal*, 25(3):162–165, 2002.

[20] J. Schaeffer, Y. Björnsson, N. Burch, A. Kishimoto, and M. Müller. Solving checkers. In *IJCAI*, pages 292–297, 2005.

[21] N.R. Sturtevant and M.J. Rutherford. Minimizing writes in parallel external memory search. *IJCAI*, 2013. To appear.

[22] Tom van Dijk, Alfons Laarman, and Jaco van de Pol. Multi-core and/or symbolic model checking. *ECEASST*, 53, 2012.

[23] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

Power plant scheduling in long term scenarios with GAMS

Maik Günther

Stadtwerke München GmbH,
D-80287 Munich, Germany
<guenther.maik@swm.de>

Abstract. Power plant scheduling can be very time-consuming in long term scenarios. At Stadtwerke München GmbH a schedule of 25 years in one hour intervals is completed in several days with the commercial solution BoFiT. In order to reduce this huge amount of time, a new software is developed. This paper describes the new system KEO and focuses on measures to reduce the CPU-time. These measures are reduction of the planning horizon, parallelization of the calculation and also, calculation with typical days. KEO significantly decrease required time to generate a schedule from 200 hours to roughly 21 minutes.

Keywords: power plant scheduling, long term scenarios, Mixed Integer Programming, CPU-time, parallelization, planning horizon, k-means

1 Introduction

Stadtwerke München GmbH (SWM) is one of the biggest utility companies in Germany. SWM uses the software BoFiT from ProCom GmbH [1] for short term scheduling of their power plants with district heating [2] in 15 minute intervals. In addition, a long term BoFiT-model exists at SWM. Actually, long term scenarios in one hour intervals from 2016 till 2025 are calculated. A CPU-time of approximately 200 hours is required to calculate long term scenarios with BoFiT 2.19. Among others, this was the reason to develop the system KEO (KEO means in German Kraftwerkseinsatzoptimierung) to calculate the optimal operation of power plants in long term scenarios.

With a focus on measures to reduce the CPU-time, KEO will be described in this paper. At first the planning horizon is divided into single days to decrease the complexity of the planning problem. Subsequently, the calculation is parallelized and typical days are generated with the k-means algorithm. With all these measures a long term schedule can be calculated in 21 minutes.[1] According to internal regulations of SWM, KEO should have developed based on software that is still in use in SWM in order to guarantee that other employees will be

[1] Both, KEO and BoFiT use CPLEX. But they run at different servers. Thus, a fair comparison of the CPU-time is not possible. A reduction of the CPU-time by more than 30% is not expected, if BoFiT would run at the KEO-server.

able to shortly get familiar with this software. Thus, GAMS [3], MS Excel and Visual Basic were used.

This paper is structured as follows: In section 2 the mathematical model of the scheduling problem is explained. Section 3 explains all measures to reduce the CPU-time. A conclusion is presented in section 4.

2 Model for power plant scheduling

The following power plant fleet of SWM in Munich is implemented in KEO [4]:

- 1 waste incineration plant with cogeneration of heat and power,
- 1 coal-fired power plant with cogeneration of heat and power,
- 3 gas-fired power plants with cogeneration of heat and power,
- 6 gas-fired heating plants,
- geothermal energy (the planned development in Munich [5] is implemented in KEO).

Very small power plants (e.g. emergency backup generators) and hydro power plants of SWM in Munich are not implemented in KEO. These plants are too small and not important for a power plant fleet with district heating. Furthermore, avoided grid use charges (§18 Stromnetzentgeltverordnung (StromNEV)), control reserve and other ancillary services are not part of the optimization in KEO.

As an example, Fig. 1 shows a diagram of the waste incineration plant. On the left hand side, the contract for waste C_W can be seen. It contains information about the minimal and maximal waste which has to be burned per hour. A minimal amount of waste has to be burned in every hour because of limited storage capacity for waste. Moreover, the contract gives information about the cost of waste.

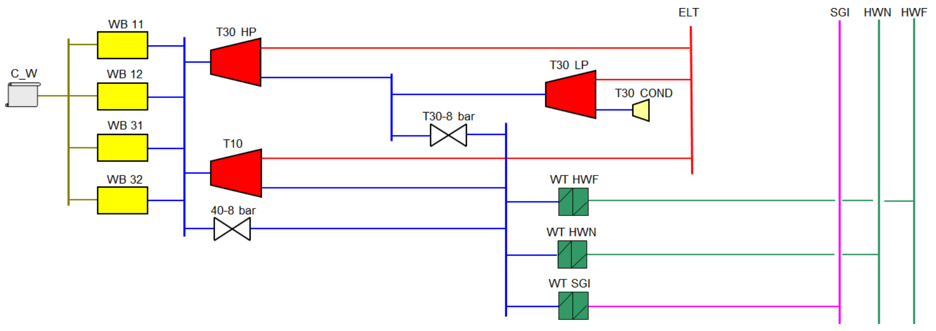


Fig. 1. Diagram of the waste incineration plant.

Connected to the contract are four waste-fueled boilers WB 11, WB 12, WB 31 and WB 32. By burning waste, steam (blue line) will be generated. The generated steam flows through the steam turbine $T30\ HP$ (HP stands for high pressure) and/or through the steam turbine $T10$ and/or through the pressure reducing station 40-8 *bar* (from 40 bar to 8 bar).

The steam from the boilers expands in the steam turbine. Thus, electricity will be produced (red line) and the steam is on a lower energy level. The waste incineration plant is connected to the electricity grid ELT. Steam can flow from the steam turbine $T30\ HP$ to the heat exchangers and/or to the steam turbine $T30\ LP$ (LP stands for low pressure). $T30\ COND$ is a condenser behind $T30\ LP$. The three heat exchangers $WT\ SGI$, $WT\ HWN$ and $WT\ HWF$ transfer the steam to the three district heating grids. Purple lines stand for a steam grid whereas green lines stand for hot water grids.

The power plant fleet of SWM is modeled with Mixed Integer Programming (MIP). A MIP-model is designed as follows:

$$P \begin{cases} & MinZ = f(x,y) \\ s.t. & g_j(x,y) \leq 0 \quad j \in J \\ & x \in X \cap \mathbb{R}^n, y \in Y \cap \mathbb{Z}^m \end{cases} \tag{1}$$

The functions f and g are convex, differentiable and linear. The space X is compact and convex. The space Y is bounded by hyperplanes of integer values: $Y = \{y \mid y \cap \mathbb{Z}^m, A_y \leq a\}$.

Assuming a set of assemblies $\mathcal{I} = 1, \ldots, I$, a discrete timeframe $\mathcal{T}$ in one hour intervalls with the index $t = 1, \ldots, T$ and a set of fuel types $\mathcal{J} = 1, \ldots, J$. The objective of power plant scheduling is maximization of profit by taking constraints into account. Revenue can be realized by selling electricity $ELT_{t,i}$ and heat $Heat_{t,i}$. Costs result from fuel $Fuel_{t,j}$, CO_2 allowances $CO2_{t,i}$ and taxes $Taxes_{t,i}$.

$$max\ Profit = \sum_{t=1}^{T}\sum_{i=1}^{I}\sum_{j=1}^{J} ELT_{t,i} + Heat_{t,h} - Fuel_{t,j} - Taxes_{t,i} - CO2_{t,i} \tag{2}$$

As a hard constraint the thermal energy demand $Demand_{t,h}$ at all district heating grids h has to be covered at all times.

$$Heat_{t,h} = Demand_{t,h} \quad \forall t \in T; h \in H \tag{3}$$

Exemplified, the mathematical model of the four waste-fueled boilers WB 11, WB 12, WB 31 and WB 32 is presented now. Due to limited number of pages the formulas of the compulsory utilization, of the priority control and of the gradient must be omitted. A transfer function makes a connection between fuel $FuelIn_{t,i}$, efficiency $Efficiency_i$ and produced thermal energy $HeatOut_{t,i}$.

$$FuelIn_{t,i} * Efficiency_i = HeatOut_{t,i} \quad \forall t \in T; i \in I \tag{4}$$

The produced thermal energy $HeatOut_{t,i}$ has a minimal limit $MinLim_i$ and a maximal limit $MaxLim_i$ at each waste-fueled boiler. The binary variable $OnOff_{t,i}$ determines if boiler i is running in timeslot t or not.

$$OnOff_{t,i} * MinLim_i \leq HeatOut_{t,i} \quad \forall t \in T; i \in I \tag{5}$$

$$OnOff_{t,i} * MaxLim_i \geq HeatOut_{t,i} \quad \forall t \in T; i \in I \tag{6}$$

The waste-fueled boiler i starts ($Start_{t,i} = 1$), if it runs in t and if it does not run in $t-1$. Otherwise, $Start_{t,i} = 0$.

$$OnOff_{t-1,i} - OnOff_{t,i} + Start_{t,i} \geq 0 \quad \forall t \in T; i \in I \tag{7}$$

To control the operation of the waste-fueled boilers a minimal working time $MinOn_i$ and a minimal idle time $MinOff_i$ are necessary. For this, the binary variables $OnOff_{t,i}$ and $Start_{t,i}$ are applied. The operation modes of all power plants in KEO are influenced only by time depended constraints. Another way is the implementation of starting costs. But it is difficult to determine correct starting costs. Thus, time dependent constraints are used with observed data from reality.

$$Start_{t,i} * MinOn_i - \sum_{t^*=t}^{t+MinEin_i-1} OnOff_{t^*,i} \leq 0 \quad \forall t \in T; i \in I \tag{8}$$

$$Start_{t,i} * MinOff_i - \sum_{t^*=t-MinOff_i}^{t-1} OnOff_{t^*,i} \leq MinOff_i \quad \forall t \in T; i \in I \tag{9}$$

3 Measures to reduce the CPU-time

The application of MIP is established for many optimization problems at the energy sector [6]. E.g. Bagemihl [7] and Groß[8] use MIP for power plant scheduling.

The tool to build the MIP model of KEO is GAMS, because GAMS was still in use at SWM. All parameters (demand at the district heating grids, electricity prices, temperature, parameters of power plants, etc.) are stored in MS EXCEL files. These files are imported into the GAMS model of KEO via GDX interface of GAMS. The solver is CPLEX 12.2.0.0. It runs until the result of the MIP problem is greater than three percent of the result of the Linear Programming (LP) problem. All test runs were conducted on a XEON 5860 with Intel 24 x 3.33 GHz processors and 196.6 GB of RAM under Windows 2008 R2.

In a first step KEO was developed for short term problems with a planning horizon of only seven days. The short term version of KEO optimized the seven days en bloc. In the second step it was planned to simplify the short term version of KEO to calculate long term scenarios of 25 years. This two-stage process was chosen to discuss all parameters with the departments of SWM on the basis of single weeks and not on the basis of a simplified model.

One possible way to simplify KEO is to implement e.g. the waste incineration plant in Fig. 1 only as one element and not as different elements (waste-fueled boiler, steam turbine, condenser, pressure reducing station, heat exchanger). This approach helps to reduce the CPU-time. But, at expense of loosing accuracy. By building the long term version of KEO it was obvious that other measures are able to reduce the CPU-time to 21 minutes. Thus, a simplification of KEO was not done. The implemented measures are explained in the following subsections.

3.1 Fragmentation of the planning horizon

A way to create a long term model out of a short term model is to increase the planning horizon. Thus, in the long term model 9,125 days (=25 years) have to be calculated en bloc. Such a huge problem leads to a computer with an unrealizable RAM (at the moment) and to an unacceptable CPU-time. An increase of the planning horizon is the wrong approach.

Thus, BoFiT calculates five days en bloc sequentially until all 9,125 days are calculated. A stronger fragmentation of the planning horizon can be achieved by calculating only single days instead of five days en bloc. This approach is realized in KEO. The MIP model of a single day with 24 hours leads to a matrix in GAMS with 41,989 rows, 27,523 columns and 116,358 elements not equal to zero. In contrast, the MIP model of seven days leads to a matrix with 293,677 rows, 192,211 columns and 826,662 elements not equal to zero.

At the transition of two separate calculated days it may happen at some power plants that there are differences between the calculated schedule and the operation in reality. However, that is not a problem, because KEO should not calculate a guideline for power plants in reality. KEO was only designed to calculate long term scenarios for the EW-model. The EW-model is a separate model. It contains the value chain of SWM e.g. for electricity and heat. It operates on an annual basis. Thus, KEO must only provide the sums of produced electricity, produced heat and used fuel for each year. It does not matter if a power plant runs some hours too long or too short at the end or at the beginning of a few days. The percentage mean value between the calculation with one or seven days en bloc is 1.0% and the standard deviation is 0.9%.

Table 1 shows the results of the calculation with whole weeks and single days. KEO performs more than four times faster by calculating only single days instead of single weeks. Thus, the following calculations are done with a reduced planning horizon of single days. The results of KEO with single days are very close to the results of the commercial solution BoFiT. The percentage mean value between the calculation with KEO and the calculation with BoFiT is 2.9% and the standard deviation is 3.4%.

Table 1. Fragmentation of the planning horizon.

software	BoFiT	KEO	KEO
calculation type	sequential	sequential	sequential
calculated days en bloc	5	7	1
CPU-time to calculate 25 years	720,000 sec.	261,956 sec.	60,405 sec.

3.2 Parallelization of the calculation

The short term version of KEO calculates all days sequentially. Fig. 2 shows this approach. KEO starts with Monday, then with Tuesday, etc.

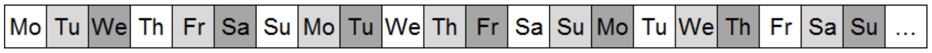

Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su ...

Fig. 2. Sequential calculation.

It could be observed in experiments that the CPU usage was never higher than 7% during the sequential calculation of the schedule, and only at the moments when CPLEX was running. Thus, the calculation was parallelized. The example in Fig. 3 shows: At first all Mondays of the three weeks are calculated at the same time, then all Tuesdays, etc. The usage of the CPU is now up to 100%.

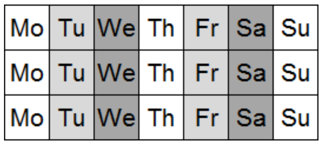


Fig. 3. Parallel calculation.

The architecture of KEO was modified to implement the parallelization of the calculation (see Fig. 4). The file EW-model.xls represents the EW-model and the file KEO.xlsm contains the Visual Basic source code to control the parallelization. Furthermore, the interface between KEO and the EW-model is controlled by KEO.xlsm. Parameters can be transferred from the EW-model to the parameter files of each year (e.g. 2010.xls for the year 2010). In addition, the results of the optimization can be transferred to the EW-model.

The parallelization works in this way: At first KEO.xlsm copies all relevant data of the first calculated day (1$^{\text{st}}$ January) of each year into the parameter files of each year. Then KEO.xlsm starts the optimization of each year at the same time (only the 1$^{\text{st}}$ January will be optimized). For this, KEO.xlsm is using application programming interfaces (API) to generate an own process for each year. KEO.xlsm is waiting until a GAMS-process is still running. Thereafter the second iteration starts. Parameters of the 2$^{\text{nd}}$ January of each year are copied into the parameter files, then the optimization starts parallel, etc. The calculation of the power plant schedule stops, if 365 days are calculated. The result is a schedule in one hour intervals from 2016 till 2025. Results are stored by GAMS in CSV files. KEO.xlsm is using these CSV files to provide for each year the sums of produced electricity, produced heat and used fuel for the EW-model.

Table 2 shows the results of the sequential and parallel calculation. It can be seen that the parallel calculation reduces the CPU-time by nearly 68%. Thus, the parallel calculation is used in the next subsection.

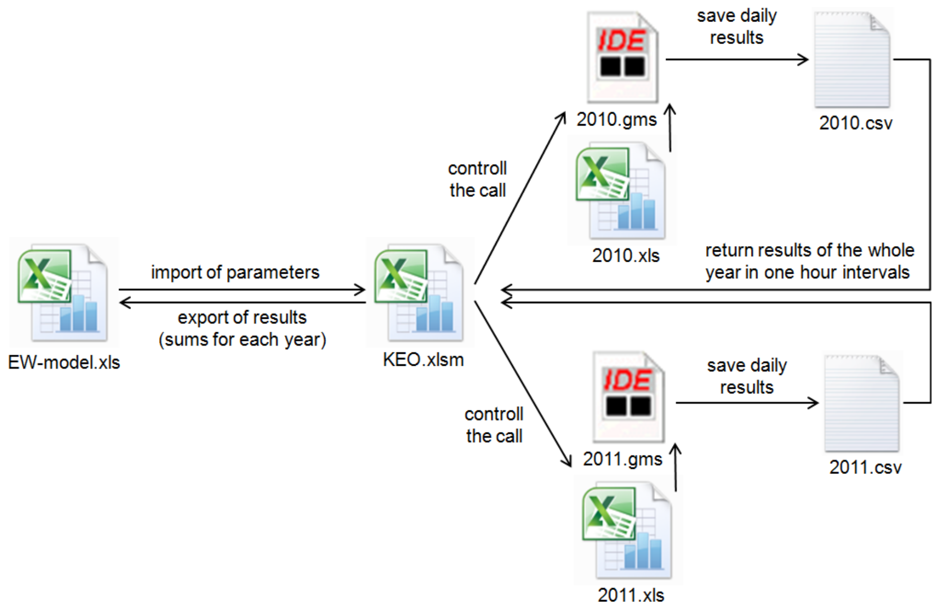


Fig. 4. Architecture of KEO (two years are calculated parallel in this example).

Table 2. Parallelization of the calculation.

software	BoFiT	KEO	KEO
calculation type	sequential	sequential	parallel
calculated days en bloc	5	1	1
CPU-time to calculate 25 years	720,000 sec.	60,405 sec.	12,760 sec.

3.3 Typical days

The input parameters electricity prices per hour and demand of the district heating grids per hour are sometimes very similar at different days. To reduce the CPU-time it is helpful not to calculate 365 days per year. It would be better to merge similar days to typical days with the aid of cluster analysis and only to calculate these typical days.

K-means clustering, expectation-maximization (EM) algorithm and hierarchical clustering were analyzed in a separate study (not published). In experiments k-means provides better results than the EM algorithm. Hierarchical clustering was not applicable, because the demand of the district heating grids and the electricity prices are not normally distributed. But this is a requirement for the application of the EM algorithm. Thus, k-means was chosen for clustering. It is worthy of mention that the three day types weekday, Saturday/holiday, Sunday and the three seasons summer, winter, transition (spring, autumn) are distinguished.

Instead of calculating 365 days for every year it is now possible to calculate only 36 typical days. The CPU-time is reduced to 1,261 seconds for a scenario of 25 years (see Table 3). The percentage mean value between the calculation with 36 typical days and 365 days is 0.9% and the standard deviation is 1.2%.

Table 3. Typical days.

software	BoFiT	KEO	KEO
calculation type	sequential	parallel	parallel
calculated days en bloc	5	1	1
calculated days/typical days per year	365	365	36
CPU-time to calculate 25 years	720,000 sec.	12,760 sec.	1,261 sec.

4 Conclusion

It was explained in this paper how KEO works. One focus was the reduction of the CPU-time. Different measures were introduced to calculate long term scenarios which lead to a CPU-time of nearly 21 minutes. These measures are the fragmentation of the planning horizon into single days, the parallelization of the calculation and the calculation with 36 typical days instead of 365 days per year. In contrast to 21 minutes, approximately 200 hours are needed by the commercial solution BoFiT 2.19 to generate a power plant schedule.

In our current activities it is planned to implement a SQL database. This helps to reduce the CPU-time, because at the moment MS EXCEL is used as a database.

References

1. ProCom, `www.procom.de`
2. Stock, G.; Ressenig, A.: Intraday-Einsatzoptimierung mit BoFiT am Beispiel der Stadtwerke München. In: Optimierung in der Energiewirtschaft, VDI-Berichte 2018, pp. 43–58 (2007)
3. GAMS Home Page, `www.gams.com`
4. Energy Generation, `www.swm.de/english/company/energy-generation.html`
5. Stadtwerke München: SWM Vision. Fernwärmeversorgung bis 2040 zu 100% aus erneuerbaren Energien, press release (2012)
6. von Brunn, H.: Energiebezugs- und Einsatzoptimierung in Querverbundsystemen Markterhebung der AGFW, In: Optimierung in der Energieversorgung. Planungsaufgaben in liberalisierten Energiemärkten, VDI-Berichte 1508, pp. 65–74 (1999)
7. Bagemihl, J.: Optimierung eines Portfolios mit hydrothermischem Kraftwerkspark im börslichen Strom- und Gasterminmarkt, Forschungsbericht 94 (2003)
8. Groß, S.: Untersuchung der Speicherfähigkeit von Fernwärmenetzen und deren Auswirkungen auf die Einsatzplanung von Wärmeerzeugern, TU Dresden (2012)

Decentralized, cooperative agv control – first results

Christoph Schwarz[1], Jürgen Sauer[2]

[1] OFFIS – Institute for Information Technology, Christoph.Schwarz@offis.de

[2] Carl von Ossietzky University Oldenburg, Juergen.Sauer@uni-oldenburg.de

Introduction

We developed a multi agent based decentralized control system for an automated guided vehicle system. Controlling of an automated guided vehicle system, especially a decentralized controlling has to deal with two points: route-planning and order allocation. We solved these points in a strictly decentralized way meaning that the single vehicles are autonomous but behave in a cooperative manner.

The route-planning was solved using a decentralized version of the free-time window approach introduced by ter Mors ([1]) with some modifications to make the agents cooperative. The order allocation is done with the help of auctions and intelligent shop scheduling. The main ideas are described in [2]. We used a MASON / JADE based simulation which can simulate the agv system fast in realistic. The simulation and the coupling between the MASON based and the JADE based simulation is described in detail in [3].

We tested our approach with a real life scenario in contrast to the implemented, central control system which is used in this scenario. First results show an overall system performance which is slightly better than the used central control.

Route Planning

In autonomous controlled AGV systems the individual AGVs have to plan their route through the area (for example the warehouse) by their own. But since there are other AGVs in the area the risk of conflicts arise. For example see figure 1. In this situation a conflict between AGV 1 and AGV 2 would arise at the section of the layout which is marked with a red circle if both AGVs would plan without considering the plan of the other vehicle.

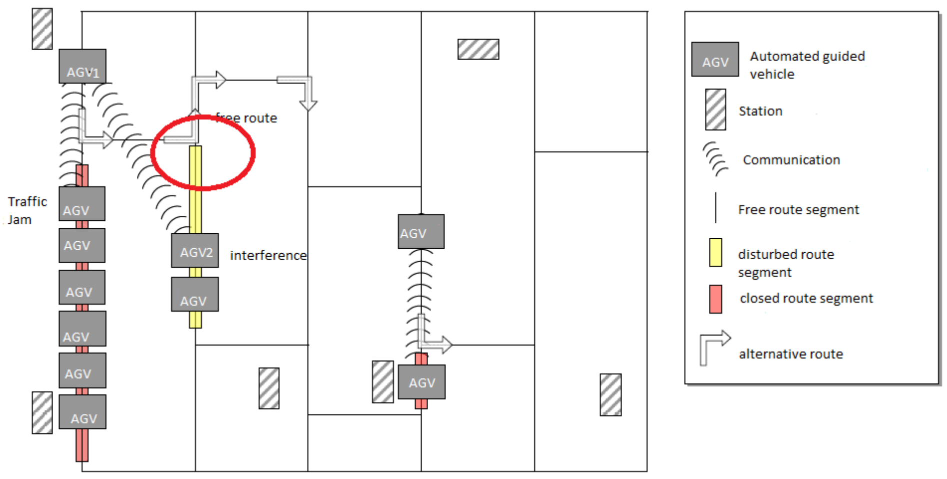


Figure 1

To avoid such conflicts and the resulting resolutions a form of conflict free routing is necessary. The concept of conflict free shortest path routing for AGVs was introduced in 1985 by Broadbent et. al.[4].

In 2007 ter Mors et. al. presented a free path routing approach named context-aware routing which solves the conflict free routing problem with an asymptotic running time of O(nv log(nv) + nv2) where v corresponds to the number of vehicles and n to the number of nodes in the given way graph (see [1]). In their case the agents reserve the way segments they plan to use and thus alter the conditions for agents which are planning after them (because more reservations are in the system). When agents plan a new route they are not allowed to violate reservations (i.e. to use way segments on time intervals for which they are reserved by other vehicles). These approach, built for a central planning station was used for our decentralized approach.

Our agents all have graph in which they store the locks they know about. Whenever they plan a route (using the algorithm described above and their inner state knowledge) they send a lock message to all other agents they can reach. In these lock messages the all resource / time slot pairs are stored. The agents which receive these messages include these locks in their internal graphs. When an agent is idle he has to move to an parking slot in order to not block important way segments for other agents.

Our first added cooperative behavior gets rid of the necessity of driving to a parking slot when idle. If an agent is idle he reserves its current position for the rest of the time horizon and marks this reservation as a parking reservation. If an agent plans his route his route he calculates two routes: One ignoring all parking reservations and one in the normal way. If the route which ignores the parking reservations is better (e.g. faster or shorter) he sends a message to the agents which have made these reservations and ask him to give the spot free. These agents than search for a spot with now reservations and plan a way to these spots . They then withdraw the old parking reservations and made new ones for the found spot. In this way the asking agents can use the better route.

In the future we want to integrate more cooperative strategies as shown in [2].

Order Allocation

The decentralized allocation of orders is an important task in nearly any decentralized system (see for example [5]). The purpose of task allocation is to assign the tasks that occur in the system to the different entities that could handle them. In the case of an AGV system for intralogistics task are usually transportation orders i.e. the order to transport a good from a source to a target destination. In decentralized order allocation there is no global entity which assigns all of the orders thus the autonomous units have to solve this problem by their own. A decentralized order allocation becomes dynamic if the assignments can change over time (meaning a once made assignment can be revoked).

We used a dynamic auction to realize order allocation. If an order has to be allocated the station on which the order starts sends an order notification message to each agent (or each agent in a given range).These agents then send a proposal for this order. The score of the proposal can depend on factors like starting time, finishing time, distance driven to fulfill the order or similar factors. After a given time the station closes the auction and sends a proposal accepted message to the winner and proposal rejected messages to the other agents.

If the score of the transportation order increases the score in the proposal by a certain threshold, the agent can give the order back to the station which then restarts the order. This can happen at the moment for example when errors occur. In the future, when more cooperative strategies have been implemented this can maybe occur more often.

First Results

We tested our approach described in the previous sections with a simulation of a beverage bottling plant (shown in figure 2). The area is rectangular with a dimension of 79m x 125m. The plant has 33 stations at which loading and unloading of transport goods takes place and 4 parking positions. In average 46 transport orders must be performed per hour. In total 8 AGVs are in use in order to fulfill these transport orders. We had access to the central control system which is used for the agv system of these bottling plant.

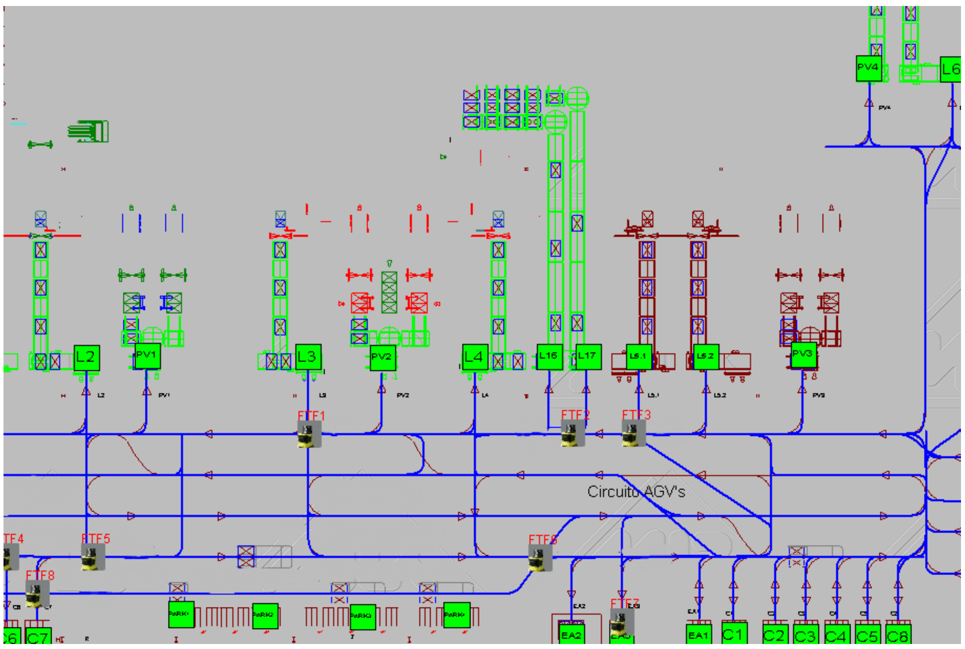


Figure 2

We compared the following configurations:

1. Given central control (8 vehicles)
2. 8 vehicles, decentralized control
3. 8 vehicles, decentralized control with idle lock negoations

The results can be seen in table 1. The decentralized approach leads to significant shorter average duration of an transport order and 7,9% less driven distance. The addition of the idle lock negations reduce the total driven distance and the time driven without cargo even more. The reason is that the vehicles do not make the drives to the parking slots. The parking slots in this scenario are in the far south part of the layout. It is possible that these improvements are less significant in scenarios where the parking slots are in more central part of the layout. We will make experiments with different layouts in the future to check this.

Table 1

	Average Duration	**Total Driven Distance**	**Driven without cargo**	**Average waiting time**	**Max. waiting time**
1	371,1 s	4500 m	56,77 %	67,5 s	278,8 s
3	275,5 s	4144 m	55,35 %	59,3 s	244,1 s
5	263,5 s	3897 m	52,12 %	54,12 s	241,6 s

Conclusion

Our decentralized approach has led to slightly better system performance then the central control in our reference scenario. The addition of the idle lock cooperation has improved the performance even more. In the future we want to add more cooperation strategies and verify the results with a different scenario.

References

[1] A.W. ter Mors and J. Zutt. Context-aware logistic routing and scheduling. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, pages 328–335, 2007.

[2] C. Schwarz and J.Sauer. Towards decentralized agv control with negotiations. Proceedings of the Sixth Starting AI Researchers' Symposium - STAIRS. 2012.

[3] C. Schwarz, J. Schachmanow, J. Sauer, L. Overmeyer and G. Ullmann. Coupling of a discrete event multi agent simulation with a real time multi agent simulation. Proceedings of the ISC 2013.

[4] A.J. Broadbent, C.B. Besant, S.K. Premi, and S.P. Walker. Free ranging agv systems: promises, problems and pathways. Proc. 2nd Int. Conf. on Automated Materials Handling, 1985.

[5] Jacques Ferber. Multi-agent systems : an introduction to distributed artificial intelligence. Addison-Wesley, Harlow, 1998.

Scheduling4Green

Jürgen Sauer

Department of Computer Science
University of Oldenburg
D-26111 Oldenburg
juergen.sauer@uni-oldenburg.de

Abstract: Reducing energy consumption is one of the actual challenges in all parts of industry. This can lead to a low carbon footprint for activities under consideration as well as to cost effective reductions of overall energy consumptions. This paper shows two examples how intelligent scheduling strategies can be used to limit energy consumption in production to a given range.

1 Introduction

Nearly all areas of production and logistics have presented their own „we are going green" proposals. Most often green means the reduction of power consumption but also reducing waste, reducing the usage of dangerous ingredients etc. can be found here. To meet the reduction goals several actions may be taken:

- using new technologies e.g. in the production process may reduce ingredients, waste as well as energy consumption
- using new machines may reduce energy consumption and waste
- changing ingredients may reduce energy consumption
- rescheduling the production process may reduce energy consumption.

The last topic shall be tackled in this paper. Especially the "right" scheduling of energy intensive production processes can lead to a reduction of overall or peak energy needs and this also has some advantages regarding costs.

This paper introduces the actual project "Scheduling4Green" and will present at first some of the problems and production processes in which the scheduling according to energy consumption will lead to main advantages. This is followed by some ideas and approaches on how to cope with the scheduling regarding energy consumption. Two examples from glas production and injection molding shall illustrate the problems and approaches.

2 Energy consumption and costs in production

Most of the production processes are using special equipment within the single steps that are performed. This equipment, typically machines or apparatus, needs energy on a constant basis or on a fluctuating basis. The fluctuation is generated by the energy consumption within the production process. Such fluctuation is found e.g. in most of the processes in which material has to be heated.

Example 1: Production of security glass. The production process consists mainly of three steps. In a first step, the existing sheet of glass is heated up to 640 °C and then cooled down quickly by air. The third step is a quality control. Figure 1 shows the power consumption profile of this process.

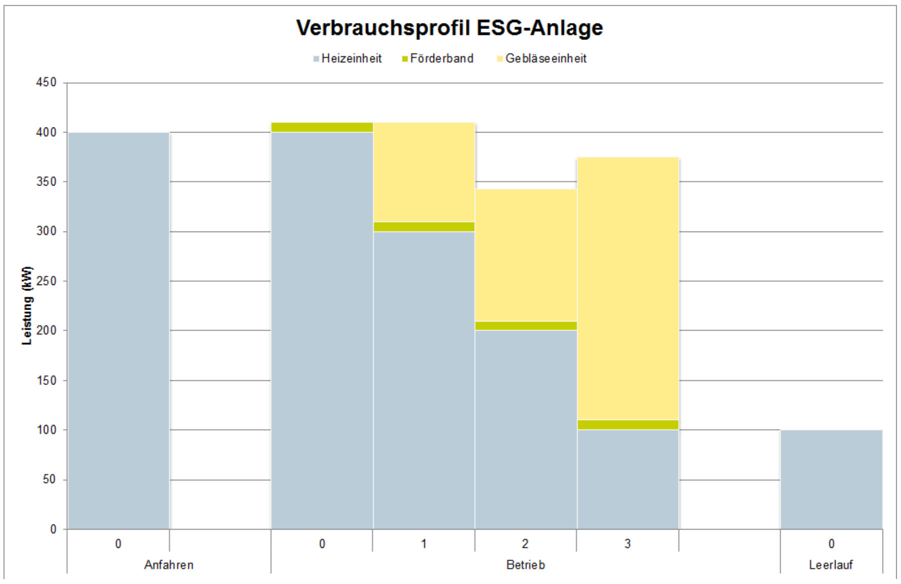


Figure 1: Power consumption profile in security glas production [Ben12]

Example 2: Injection molding. This technology is used for the production of a lot of products and parts mostly made up of plastics. The process has three major steps. In a first step, the needed material is fed into a barrel. In the second step it is mixed and heated up to its melting point and then pressed into a mold cavity. There it is cooled down and hardens to the configuration of the cavity. Figure 2 shows the power consumption profile of this process. The difference in the power consumption of the process steps is considerable higher than in the example before.

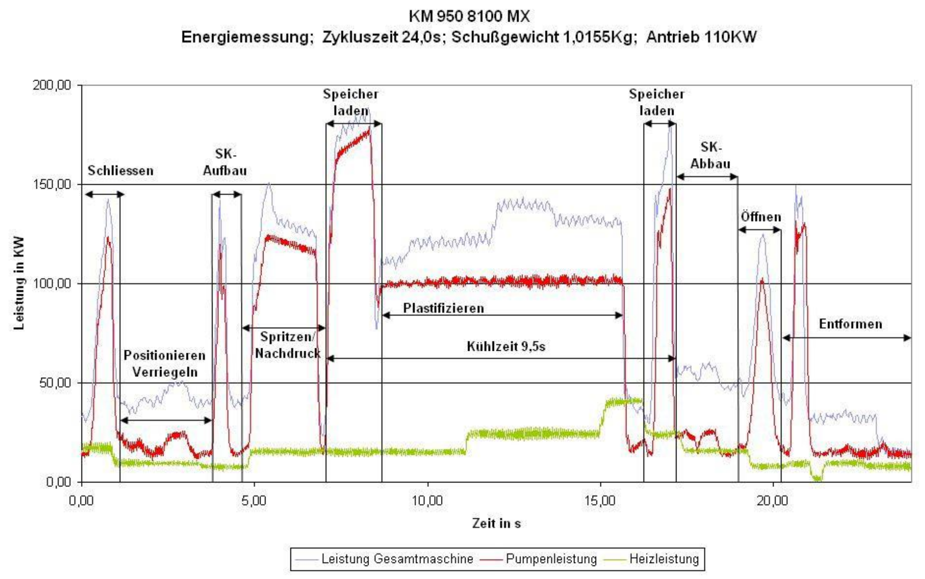


Figure 2: Power consumption profile in injection molding [Wil08]

The energy consumption of production is typically result of a forecast process and is used to calculate energy costs on the basis of specific contracts. These contracts usually contain peak energy needs (upper bounds) that are basis of the rate. With this forecasts the energy producer plans the energy production. If the consumer exceeds the upper bounds then a penalty or the next higher rate has to be paid, normally for the whole time period. This can easily occur if machines are used in parallel and several of the energy peaks of the production processes will superimpose. Therefore most of the companies have some kind of emergency plan, if the energy consumption approaches the upper bounds, e.g. the unplanned shut down of machines. But this is disturbing the whole manufacturing process and eventually damaging the machines. This is the reason for looking for a scheduling solution that can lead to a production schedule that avoids the power consumption peaks.

3 Scheduling regarding energy consumption

As stated in section 2 the main goal of a scheduling algorithm that regards the energy consumption should be to keep the energy needs in a given range and avoid peaks.

Figure 3 and 4 [Giz12] show what this means. Figure 3 shows the accumulated energy consumption of five injection molding machines if it is allowed to start them randomly, which especially means that they can start at the same time.

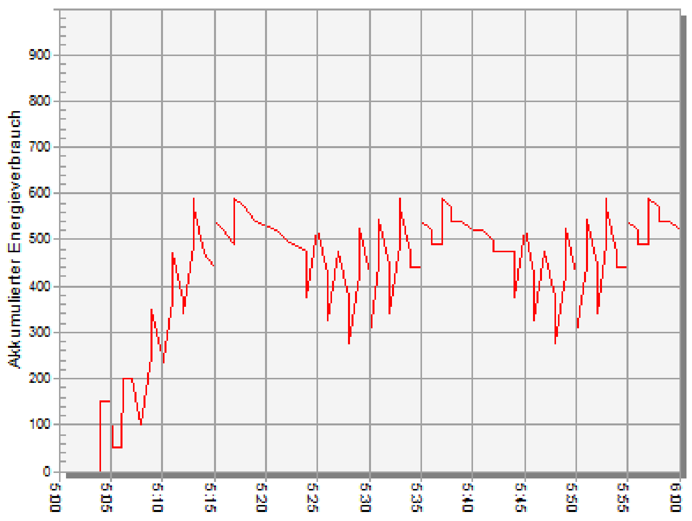


Figure 3: Example of accumulated power consumption

Figure 4 shows the diagram with the reduced accumulated power consumption when a fixed delay is used between the start of the machines. This works well because we have a simple problem with a few identical machines and processes, which is not the case in other examples.

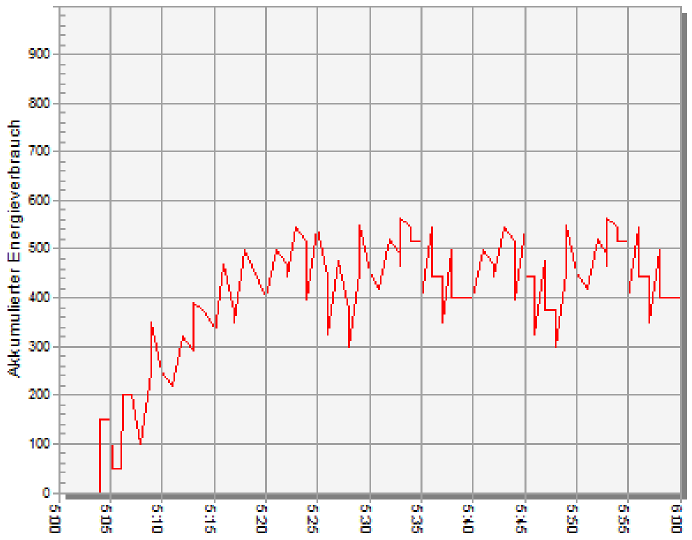


Figure 4: Example of improved accumulated power consumption

In our project we created in a first step a simulation model to show the effects of superimposed energy consumption profiles. Figure 5 gives a sketch of the model build with PlantSimulation. During the simulation the energy consumption is collected and leads to the profiles in figure 3 and 4.

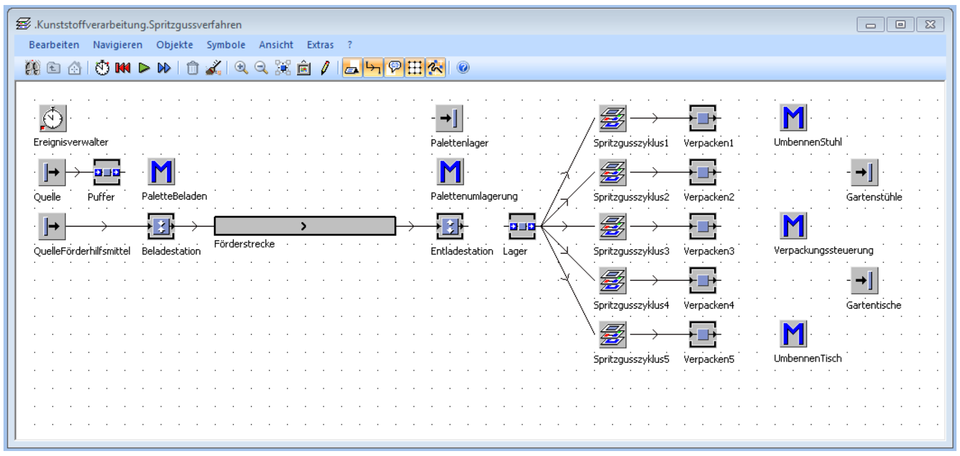


Figure 5: Simulation model in PlantSimulation [Giz12]

Actually we are developing a scheduling algorithm that takes into account the energy consumption of the process steps and tries to keep the energy consumption within the given range. Main conceptual parts in the development of the algorithm are

- to create a more abstract view of the energy consumption profile that makes it easier to schedule it, e.g. use a rectangular function for energy consumption.

- to find some heuristics that can be used to create a schedule in an easy way, thereby using a general order-based heuristic [Sau04] for scheduling.

References

[Ben12] Bender, T.: Dynamische Lastganganalyse für eine Produktions- und Energieoptimierung, Masters Thesis, Universität Oldenburg, 2012.

[Giz12] Giza, N.: Simulation von Energieverbrauch in der Produktion am Beispiel der Spritzgussindustrie, Bachelor Thesis, Universität Oldenburg, 2012.

[Sau04] Sauer, J.: Intelligente Ablaufplanung in lokalen und verteilten Anwendungsszenarien, Teubner, 2004.

[SRB13] Sauer, J., Runge, S.A., Bender, T.: Lastgangbezogene Prioritätsregeln für eine Produktionsplanung bei der Veredelung von Glasprodukten, in: J. Marx Gómez et al. (Eds.), IT-gestütztes Ressourcen- und Energiemanagement, Springer-Verlag, Berlin Heidelberg, 2013, pp. 3-9.

[Wil08] Wilken, M.: Entwicklung eines Systems zur Optimierung der Energieeffizienz in der Kunststoffspritzgussindustrie, Diploma Thesis, Universität Oldenburg, 2008.