Proceedings

16. Workshop

„Planen, Scheduling und Konfigurieren, Entwerfen“

PuK 2002

Edited by
Jürgen Sauer

Freiburg, 10./ 11.10.2002

Inhalt

Workshop Organisation

Dr. Jürgen Sauer
Universität Oldenburg
FB Informatik
Escherweg 2
D-26121 Oldenburg, Germany
Tel.: ++49-441-9722-220
Fax: ++49-441-9722-202
e-mail: sauer@informatik.uni-oldenburg.de
www: http://www-is.informatik.uni-oldenburg.de/~sauer/

Malte Helmert
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee, Geb. 52
D-79110 Freiburg, Germany
Tel.: ++49-761-203-8225
Fax: ++49-761-203-8222
e-mail: helmert@informatik.uni-freiburg.de
www: http://www.informatik.uni-freiburg.de/~helmert

Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots

Maren Bennewitz[†] Wolfram Burgard[†] Sebastian Thrun[‡]

[†]Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany
[‡]School of Computer Science, Carnegie Mellon University, Pittsburgh PA, USA

Abstract

Coordinating the motion of multiple mobile robots is one of the fundamental problems in robotics. The predominant algorithms for coordinating teams of robots are decoupled and prioritized, thereby avoiding combinatorially hard planning problems typically faced by centralized approaches. While these methods are very efficient, they have two major drawbacks. First, they are incomplete, i.e. they sometimes fail to find a solution even if one exists, and second, the resulting solutions are often not optimal. In this paper we present a method for finding and optimizing priority schemes for such prioritized and decoupled planning techniques. Existing approaches apply a single priority scheme which makes them overly prone to failure in cases where valid solutions exist. By searching in the space of priorization schemes, our approach overcomes this limitation. It performs a randomized search with hill-climbing to find solutions and to minimize the overall path length. To focus the search, our algorithm is guided by constraints generated from the task specification. To illustrate the appropriateness of this approach, this paper discusses experimental results obtained with real robots and through systematic robot simulation. The experimental results illustrate the superior performance of our approach, both in terms of efficiency of robot motion and in the ability to find valid plans.

1 Introduction

Path planning is one of the fundamental problems in mobile robotics. As mentioned by Latombe [10], the capability of effectively planning its motions is "eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world."

In this paper we consider the problem of motion planning for multiple mobile robots. The goal is to compute trajectories for the individual robots such that collisions between the robots are avoided. Especially in the context of multi-robot systems different undesirable situations can occur like congestions or even deadlocks. Since the size of the joint state space of the robots grows exponentially in the number of robots, planning paths for teams of mobile robots is significantly harder than the path planning problem for single robot systems. Therefore, the existing approaches for single robot systems cannot directly be transferred to multi-robot systems.

The existing methods for solving the problem of motion planning for multiple robots can be divided into two categories [10]. In the *centralized* approach [3, 15, 19] the configuration spaces of the individual robots are combined into one composite configuration space which is then searched for a path for the whole composite system. In contrast, the *decoupled* approach [6, 8, 13, 18] first computes separate paths for the individual robots and then resolves possible conflicts of the generated paths.

While centralized approaches (at least theoretically) are able to find the optimal solution to any planning problem for which a solution exists, their time complexity is ex-

ponential in the dimension of the composite configuration space. In practice one is therefore forced to use heuristics for the exploration of the huge joint state space.

Many methods use potential field techniques [2, 3, 20] to guide the search. These techniques apply different approaches to deal with the problem of local minima in the potential function. Other methods restrict the motions of the robots to reduce the size of the search space. For example, the approaches presented in [9, 11, 19] restrict the trajectories of the robots to lie on independent road maps. The coordination is achieved by searching the Cartesian product of the separate road maps.

Decoupled planners determine the paths of the individual robots independently and then employ different strategies to resolve possible conflicts. According to that, decoupled techniques are incomplete, i.e. they may fail to find a solution even if there is one. A popular decoupled approach is planning in the configuration time-space [6] which can be constructed for each robot given the positions and orientations of all other robots at every point in time. Techniques of this type assign priorities to the individual robots and compute the paths of the robots based on the order implied by these priorities. The method presented in [21] uses a fixed order and applies potential field techniques in the configuration time-space to avoid collisions. The approach described in [7] also uses a fixed priority scheme and chooses random detours for the robots with lower priority.

Another approach to decoupled planning is the path coordination method which was first introduced in [18]. The key idea of this technique is to keep the robots on their individual paths and let the robots stop, move forward, or even move backward on their trajectories in order to avoid collisions (see also [4]). To reduce the complexity in the case of huge teams of robots [13] recently presented a technique to separate the overall coordination problem into sub-problems. This approach, however, assumes that the overall problem can be divided into very small subproblems, a serious assumption which, as various examples described below demonstrate, is often not justified. In general, therefore, a prioritized variant has to be applied.

The methods described above leave open how to assign the priorities to the individual robots. In the past, different techniques for selecting priorities have been used. [5] applied a heuristic which assigns higher priority to robots which can move on a straight line from the starting point to their target location. In [1] all possible assignments are considered. Due to its complexity this approach has only been applied to groups of up to three robots.

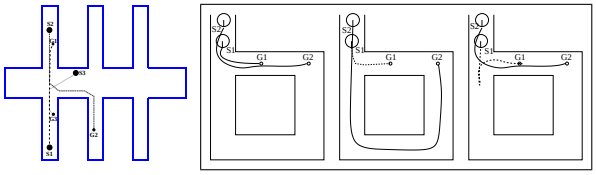

Figure 1: Situation in which no solution can be found if robot 3 has higher priority than robot 1 (leftmost image) and independently planned optimal paths for two robots (second image), sub-optimal solution if robot 1 has higher priority (third image), and solution resulting if the path for robot 2 is planned first (right).

For decoupled and prioritized methods the order in which the paths are planned has a serious influence on whether at all a solution can be found and how long the resulting paths are. To illustrate this, let us consider two examples. Figure 1 (leftmost image) shows a situation in which no solution can be found *if* robot 3 has a higher priority than robot 1. Since the path of robot 3 is planned without considering robot 1, it enters the corridor containing its target location (marked G_3) before robot 1 has left this corridor. Since the corridors are too narrow to allow two robots to pass by, robot 3 blocks the way of robot 1 so that it cannot reach its target point G_1. However, if we change the priorities and plan the trajectory of robot 1 before that of robot 3, then robot 3 considers the trajectory of robot 1 during path planning and thus will wait in the hallway until robot 1 has left the corridor. Another example is shown in Figure 1 (three images on the right). If we start with robot 1 then we have to choose a large detour for robot 2 (see Figure 1, third image). This is because robot 1 blocks the corridor. However, if the path of robot 2 is planned first, then we can obtain a much more efficient solution (see Figure 1, right). These two examples illustrate that the priority scheme has a serious influence on whether a solution can be found and on how long the resulting paths are. Moreover, it suggests that no single prioritization scheme will be sufficient for all possible multi-robot motion problems.

In this paper, we present a technique that searches in

the space of all priority schemes while solving hard multi-robot planning problems. Our approach performs a randomized hill-climbing search in the space of possible priority schemes. Since each change of a scheme requires the computation of the paths for many of the robots, it is important to focus the search. Our method achieves this by exploiting constraints between the different robots which are derived from the task specification. This has two serious advantages. First, it reduces the time required to find a solution, and second, it increases the number of problems for which a solution can be found in a given amount of time. Additionally, our algorithm is able to reduce the overall path length once a solution has been found. It has anytime characteristics [22], which means that the quality of the solution depends on the available computation time.

Our approach has been successfully applied to physical mobile robots. These results are complemented by extensive simulations, to characterize the relation between the planning performance and various problem parameters. In all experiments, we found that our approach produces highly efficient motion plans even for very large teams of robots, for different environments, and using two different decoupled path planning techniques.

The paper is organized as follows. In the following section, we introduce two decoupled path planning techniques that will be used throughout this paper. Section 3 describes our algorithm for searching for priority schemes during planning. Finally, in Section 4, we present systematic experimental results illustrating the capabilities of our approach.

2 Prioritized A^*-based Path Planning and Path Coordination

2.1 A^*-based Path Planning

Our system applies the A^* procedure [17] to compute the cost-optimal paths for the individual robots. A^* addresses the problem of finding a shortest path from an initial state to a goal state in a graph. To search efficiently, the A^* procedure takes into account the accumulated cost of reaching a certain location $\langle x, y\rangle$ from the starting position, and an estimate of the cost of reaching the target location $\langle x^*, y^*\rangle$ from $\langle x, y\rangle$. By doing so, A^* tends to focus its search in parts of the state space most relevant to the problem of finding a shortest path. This property, which makes A^* an efficient search algorithm, has given A^* an enormous popularity in the robotics community. However, A^* also requires a discrete search graph, whereas robot configuration spaces are continuous. In our case we assume that the environment is readily represented by a discrete occupancy grid map—which is common in the mobile robotics literature.

Occupancy grids [16] separate the environment into a grid of equally spaced cells and store in each cell $\langle x, y\rangle$ the probability $P(occ_{x,y})$ that it is occupied by a static object. The cost for traversing a cell $\langle x, y\rangle$ is proportional to its occupancy probability $P(occ_{x,y})$. Furthermore, the estimated cost for reaching the target location is approximated by $c \cdot ||\langle x, y\rangle - \langle x^*, y^*\rangle||$ where $c > 0$ is chosen as the minimum occupancy probability $P(occ_{x,y})$ in the map and $||\langle x, y\rangle - \langle x^*, y^*\rangle||$ is the straight-line distance between $\langle x, y\rangle$ and $\langle x^*, y^*\rangle$. Since this heuristic is admissible (see [17]), A^* determines the cost-optimal path from the starting position to the target location.

2.2 Decoupled Path Planning for Teams of Robots

A^* can easily be extended to the problem of decoupled and prioritized path planning. Recall that in the multi-robot path planning problem, many robots simultaneously seek to traverse an environment. If the robots could move freely regardless of other robot's positions, the problem could easily be decoupled into many local path planning problems, in which each robot applied A^* to determine its optimal path. However, the impossibility for robots to occupy the same location at the same point in time introduces non-trivial restrictions that have to be incorporated into the individual robot paths.

A common approach is the following. In a first path planning step, each robot computes its optimal path using A^*, without any consideration of the paths of the other robots. In the remainder we denote the paths generated in this step as the independently planned optimal paths for the individual robots. Clearly, these paths might not be admissible since they lead to collisions, if executed. Thus, in a second planning step, each robot checks for possible conflicts with all other robots. Conflicts between

robots are then resolved by introducing a priority scheme. A priority scheme determines the order in which the paths for the robots are re-planned. The path of a robot is then planned in its configuration time-space computed based on the map of the environment and the paths of the robots with higher priority.

As already mentioned, A^* search can also be used to plan the motions of the robots in the configuration time-space. To incorporate the restrictions imposed by the other robots we do not allow a robot to enter a cell that is occupied by a robot with higher priority at that time. In addition to the general A^*-based planning in the configuration time-space we consider a second and restricted version of this approach denoted as the path coordination technique [13]. It differs from the general approach in that it only explores a subset of the configuration time-space given by those states which lie on the independently planned optimal paths for the individual robots. The path coordination technique thus forces the robots to stay on their initial trajectories. The overall complexity of both approaches is $O(n \cdot m \cdot \log(m))$ where n is the number of robots and m is the maximum number of states expanded by A^* during planning in the configuration time-space (i.e. the maximum length of the OPEN-list). Due to the restriction during the search, the path coordination method is more efficient than the general A^* search. Its major disadvantage, however, lies in the fact that it fails more often.

As already discussed above, the introduction of a priority scheme for the decoupled path planning leads to a serious reduction of the overall complexity. Whereas there are schemes leading to a viable solution with collision-free paths, it is easy to see that there are schemes for which no solution can be found. In addition to the fact, that the order in which the robots may plan their paths has a profound impact on the ability of finding a solution, even the quality of the solution depends heavily on the priority scheme. Examples of such situations were already discussed in the introduction to this paper. Unfortunately, the problem of finding the optimal priority scheme, is a non-trivial matter. More specifically, the NP-hard Job-Shop Scheduling problem with the goal to minimize maximum completion time [14, 12] can be regarded as an instance of the path coordination method. Therefore, we have to be content with possibly sub-optimal planning orders.

3 Finding and Optimizing Solvable Priority Schemes

This section describes our approach to searching in the space of priority schemes during decoupled path planning. The examples given above illustrate that the order in which the paths are planned has a significant influence on whether a solution can be found and on how long the resulting paths are. This raises the question of how to find a priority scheme for which the decoupled approach does not fail and for which the length of the resulting paths is minimized.

3.1 The Randomized Search Technique

Our algorithm for finding eligible priority schemes for decoupled and prioritized path planning techniques interleaves the search for collision-free paths with the search for a solvable priority scheme. It performs a randomized search combined with hill-climbing. It starts with an arbitrary initial priority scheme and randomly exchanges the priorities of two robots in this scheme. If the new order results in a solution with shorter paths than the best one found so far, it continues with this new order. Since hill-climbing approaches like this frequently get stuck in local minima, it performs random restarts with different initial orders of the robots. The number of restarts and priority exchanges are controlled by the two parameters `maxTries` and `maxFlips`.

3.2 Exploiting Constraints to Focus the Search

Whereas the plain randomized search technique produces good results, it has the major disadvantage that often a lot of iterations are necessary to come up with a solution. For example, we found that for a situation with ten robots in the environment shown in Figure 2 (leftmost image) more than 20 iterations on average were necessary to find a solvable priority scheme. In this section we therefore present a technique to focus the search that tends to reduce the search time significantly.

Our approach can be motivated through the situation depicted in the leftmost image of Figure 1. In this situation, it is impossible to find a path for robot 1 if the path

of robot 3 is planned first, because the goal location of robot 3 lies on the optimal path for robot 1. The key idea of our approach is to introduce a constraint $p_i > p_j$ between the priorities of two robots i and j, whenever the goal position of robot j lies on the optimal path of robot i. In our example we thus obtain the constraint $p_1 > p_3$ between the robots 1 and 3. Additionally, we get the constraint $p_2 > p_1$, since the goal location of robot 1 lies too close to the trajectory of robot 2.

Although the satisfaction of the constraints by a certain priority scheme does not guarantee that valid paths can be found, orders satisfying the constraints more often have a solution than priority schemes violating constraints. Unfortunately, depending on the environment and the number of the robots, it is possible that there is no order satisfying all constraints. In such a case the constraints produce a cyclic dependency. The key idea of our approach is to initially reorder only those robots that are involved in such a cycle in the constraint graph. Thus, we separate all robots into two sets. The first group R_1 contains all robots that, according to the constraints, do not lie on a cycle and have a higher priority than the robot with highest priority which lies on a cycle. This set of robots is ordered according to the constraints and this order is not changed during the search. The second set, denoted as R_2 contains all other robots. Initially, our algorithm only changes the order of the robots in the second group. After a certain number of iterations, we include all robots in the search for a priority scheme. In our experiments we figured out that this leads to better results with respect to the overall path length, especially for large numbers of iterations. The number of iterations carried out using the robots in R_2 only is controlled by a parameter denoted k in the remainder of this paper.

To illustrate our approach, again consider the situation with ten robots shown in the leftmost image of Figure 2. Whereas the starting locations are marked by $S_0, \ldots, S_9$ the corresponding goal positions are marked by $G_0, \ldots, G_9$. The independently planned optimal trajectories are indicated by solid lines. Given these paths we obtain the constraints depicted in the center image of Figure 2. According to the constraints, in the beginning the order of the six robots 3, 6, 7, 2, 4, and 9 remains unchanged during the search process. Given the restricted search space, our system quickly finds a solution. In this example, we obtained the order 0, 1, 5, and 8, for the robots lying on a cycle in the constraint graph. The resulting corresponding collision-free paths for all robots are shown in the right image of Figure 2. This demonstrates, that the constraints can drastically reduce the search space and still allow the system to quickly find solvable priority schemes.

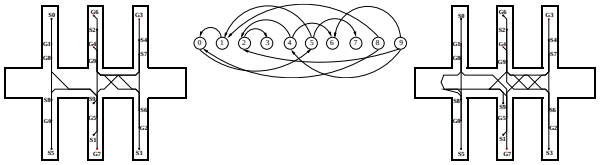

Figure 2: Independently planned paths for ten robots (left), resulting constraints (center), and the paths resulting after priority optimization (right).

4 Experimental Results

Our approach has been tested thoroughly on real robots and in extensive simulation runs. The key questions addressed in our experiments were: (1) Practicability: is our approach relevant and applicable to real robot systems? (2) Solvability: Does our approach succeed more frequently in finding valid multi-robot paths than approaches with fixed prioritization? (3) Optimality: If our approach succeeds, does it generate more efficient plans? All experiments were carried out using different environments. To evaluate the general applicability, we applied our method to the two decoupled and prioritized path planning techniques described above. The current implementation is highly efficient. It requires less than 0.1 seconds on a 1000 MHz Pentium III to plan a collision-free path for one robot in all environments described below. The whole optimization for 10 robots with 10 restarts and 10 iterations per restart requires approximately one minute.

4.1 An Example with Real Robots

The goal of the first experiment is to demonstrate the applicability of our approach to real robot systems. This experiment has been carried out using the pioneer I robots of the CS-Freiburg RoboCup team. The task of the robots is to get into their initial formation which has to be done at

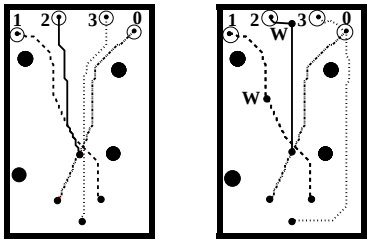

Figure 3: An application example with the Robots of the CS-Freiburg RoboCup team. The left image shows the independently planned optimal paths for the four robots and the resulting collision-free paths computed by our algorithm are depicted on the right.

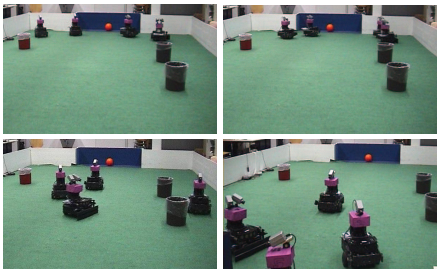

Figure 4: These four pictures show the robots carrying out the navigation plans. The top left images depicts the initial situation. In the top right image robot 2 makes space for robot 1 while robot 3 takes a detour. The lower left images shows robot 1 waiting to let robot 0 pass by. The lower right image shows the robots at their final locations.

the beginning of each match. Thereby they have to avoid colliding with other robots that are on the field already. In the particular example described here, the robots are deployed on one side of the field and have to move to their home positions on the other side. The left image of Figure 3 shows this initial configuration along with the independently planned optimal paths. As can be seen from this figure, these paths cross each other close to the center of the field leading to several several conflicts. Here we applied our approach to A^* search in the configuration time-space, and the paths of the robots were planned in the order 0, 1, 2, and 3. The resulting paths are depicted in the right image of Figure 3. As can be seen, the paths of the robots are changed in order to avoid collisions. Whereas the robots 1 and 2 shortly wait to let robot 0 pass by (the corresponding positions are marked with a "W" in the right image of Figure 3), robot 3 has to take a detour.

4.2 Simulation Experiments

To elucidate the scaling properties of our approach to larger number of robots, we performed extensive simulation experiments. In particular, we were interested in characterizing the dependence between the performance of our system on various components of our approach. In our experiments, we analyzed the number of planning problems that can be solved using our strategy, the speed-up obtained by exploiting the constraints, and the reduction of the overall path length. In all experiments, we found that our approach produces highly efficient motion plans even for very large teams of robots, for different environments, and regardless of the specific baseline path planning technique (e.g., general A^* or the path coordination method).

4.2.1 Solved Planning Problems

This first set of experiments was designed to characterize the effect of our search scheme on the overall number of failures. For each number of robots considered, we performed 100 experiments. In each experiment we randomly chose the starting and target locations of the robots. We applied four different strategies to find solvable priority schemes:

1. A single randomly chosen order for the robots.
2. A single order which satisfies the constraints for the robots in R_1 and consists of a randomly chosen order for the robots in R_2.
3. Unconstrained randomized search starting with a random order and without considering the constraints.
4. Constrained randomized search starting with an order computed in the same way as strategy (2).

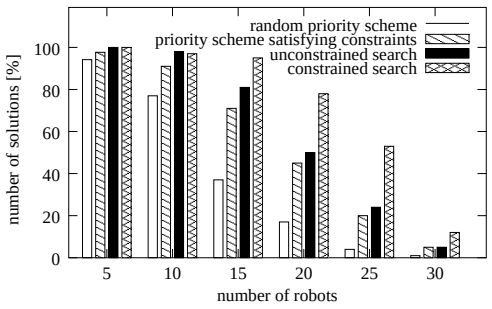

Figure 5: Solved planning problems for different strategies using A^*-based planning in the configuration time-space in the cyclic corridor environment depicted in Figure 6.

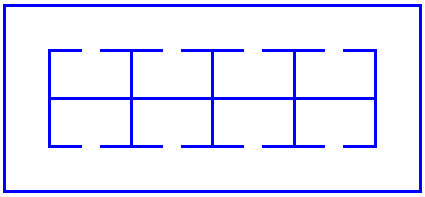

Figure 6: Cyclic corridor environment used for simulation experiments.

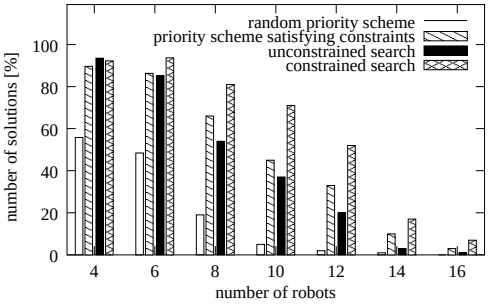

Figure 7: Solved planning problems for all four strategies using A^*-based planning in the configuration time-space in the noncyclic environment depicted in Fig. 2

All four strategies can be cast as special cases of our algorithm. In the first two strategies the corresponding values for `maxTries` and `maxFlips` are 1. For the first strategy the value of the threshold `k` is 0. The strategies 3 and 4 only differ in the value of the threshold `k`. Whereas the unconstrained search is obtained by setting $\texttt{k} = 0$, the constrained search corresponds to a value of $\texttt{k} = \infty$.

Please note that in this experiment we chose a small number of iterations for the last two strategies in order to assess the advantages of the constrained search under serious time constraints. Particularly, we chose a value of 3 for the parameters `maxFlips` and `maxTries`. Obviously, the larger the number of iterations, the higher is the probability that a solution can be found by an arbitrary randomized search. However, larger numbers of iterations drastically increase the computation time. For each technique, we performed A^*-based planning in the configuration time-space and counted the number of solved planning problems.

Figure 5 summarizes the results we obtained for the cyclic corridor environment depicted in Figure 6. The horizontal axis represents the number of robots, and the vertical axis depicts the percentage of solved path planning problems. As this result illustrates, our constrained search technique succeeds more often than any of the alternative strategies. It is interesting to note that the second strategy, which exploits the constraints but considers only one scheme in each experiment, shows a similar performance than the unconstrained randomized search. To complement these results, we performed a similar series of experiments for the noncyclic corridor environment depicted in Figure 2. The results are shown in Figure 7. Again, our constrained-based search outperforms all other strategies. All these and the following results are significant on the 95% confidence level.

To investigate the performance using a different baseline path planning algorithm, we applied all four strategies using the path coordination method instead of plain A^*. We used a variant of the environment depicted in Figure 2 with five corridors on both sides. Since the path coordination method restricts the robots to stay on their independently planned optimal trajectories, the number of unsolvable problems is much higher compared to the general A^*-based planning in the configuration time-space. As can be seen from Figure 8, our constrained search leads to a much higher success rate.

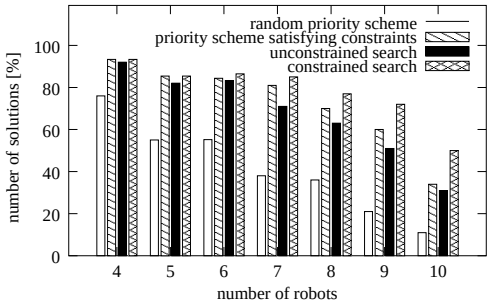

Figure 8: Solved planning problems for the four strategies using the path coordination method in the noncyclic environment.

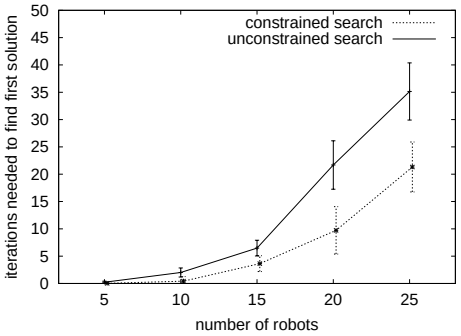

Figure 9: Iteration in which the first solution was found if the planning problem could be solved for the cyclic corridor environment.

4.2.2 Speed-up Obtained by Exploiting the Constraints

The second set of experiments was performed to investigate the ability or our approach to guide the search in the space of all priority schemes. We were especially interested in the question how much the computation time necessary to find a solution can be reduced by constraining the search. During these experiments we increased the values of `maxFlips` and `maxTries` to 10 and evaluated in which iteration the first solution was found if the planning problem could be solved. Figure 9 plots the results obtained for different number of robots in the cyclic corridor environment and Figure 10 shows the same evaluation for the noncyclic environment. As can be seen, the unconstrained search needs significantly more iterations to generate a solution for both environments. Thus, the advantages of our constrained search is two-fold. On one hand, it requires fewer iterations than the unconstrained counter-part. On the other hand, it requires less computation, since the search is restricted to a subset of the robots, which reduces the number of paths that have to be generated in each iteration during the search.

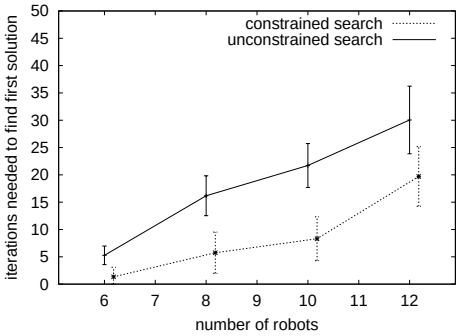

Figure 10: Iterations needed to find first solution in the noncyclic corridor environment.

4.2.3 Influence on the Overall Path Length

The next experiments were carried out to analyze the performance of our algorithm with respect to the overall path length. Since our algorithm in the beginning only considers a restricted set of priority schemes, and after $\mathtt{k}$ iterations explores the whole set of priority schemes, we are especially interested in how long the resulting paths are compared to the unconstrained search. We performed over 100 experiments in the cyclic corridor-environment and determined the average overall move costs at every iteration. The corresponding graphs are shown in Figure 11. This plot contains the average move costs for three different strategies at each iteration. The first data set was obtained for the constrained search which corresponds to $\mathtt{k} = \infty$. Using this strategy we reorder only those robots which lie on a cycle in the constraint graph. The data for the unconstrained search was obtained using $\mathtt{k} = 0$. In this case our algorithm chooses arbitrary priority schemes

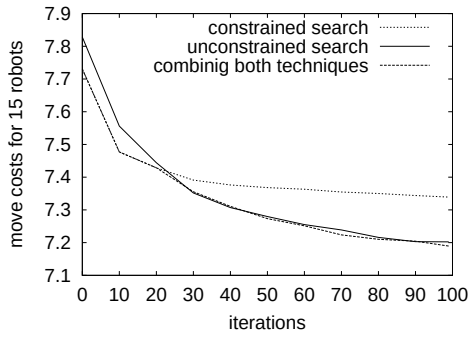

Figure 11: Summed move costs plotted over time averaged over 100 planning problems for 15 robots in the cyclic environment.

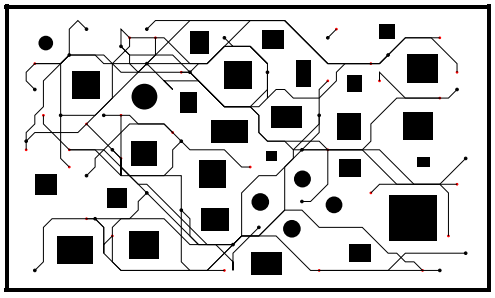

Figure 12: Resulting paths after priority optimization for a team of 30 robots.

regardless of the constraints which were extracted given the task specification. Finally, the third function labeled "combining both techniques" corresponds to the results obtained with our algorithm given $\mathtt{k} = 20$.

Since the constrained search focuses the search on the robots that pose the most serious restrictions to the other robots, it more quickly finds a solution and accordingly has more time to optimize it. Thus, in the beginning, the constrained search outperforms the unconstrained search. After 20 iterations, however, the situation completely changes. Because the unconstrained search can explore many more priority schemes, it more often finds better solutions than the constrained search. Thus, after 20 iterations, the unconstrained search leads to better results than the constrained search. As can be seen from the figure, our approach combines the advantages of both methods. In the beginning, it applies the constraints to focus the search and to quickly find a first solution which is optimized subsequently. After 20 iterations it considers arbitrary priority schemes so that the resulting path length is reduced as in the unconstrained search.

Accordingly, our randomized search that initially uses the constraints to focus the search for a viable solution and afterwards uses the unconstrained search to optimize this solution inherits the advantages of both techniques with respect to efficiency and the overall resulting path length.

4.2.4 Planning Paths for Large Teams of Robots

The final experiment in this paper is designed to illustrate that our system can be used to solve planning problems for even large numbers of robots. Figure 12 shows the paths planned for a team of 30 robots in an unstructured environment. In this particular example our system was able to generate a first solution in less than one second. The paths shown in the figure are the best solution found after 10 restarts with 10 iterations in each round. The paths are 20% shorter than the first solution found.

5 Conclusions

This paper presented an approach to optimize priority schemes for arbitrary decoupled and prioritized path planning methods for groups of mobile robots. Our approach performs a randomized hill-climbing search in the space of priority schemes in order to find a solution and to minimize the overall path length. To guide the search, our approach exploits constraints extracted from the task specification.

The approach has been implemented and tested on real robots. One experiment carried out with the CS-Freiburg RoboCup team demonstrated that our approach can effectively be used for a team of mobile robots. In addition, extensive simulations were performed to complement the physical robot experiments. The experiments suggest that our technique significantly decreases the number of failures in which no solution can be found. Additionally, our approach leads to a significant reduction of the overall path length. A further advantage of our method lies in its general applicability. Although we applied our opti-

mization technique only to two different baseline path-planning techniques in this paper, it is not limited to these two techniques. Rather, it can be used to find and optimize paths generated with arbitrary prioritized path-planning techniques.

6 Acknowledgments

We would like to thank Thilo Weigel for his efforts in carrying out the experiments with the CS-Freiburg RoboCup team robots.

References

[1] K. Azarm and G. Schmidt. A decentralized approach for the conflict-free motion of multiple mobile robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1667–1674, 1996.

[2] J. Barraquand, B. Langois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Robotics and Automation, Man and Cybernetics*, 22(2):224–241, 1992.

[3] J. Barraquand and J. C. Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1990.

[4] Z. Bien and J. Lee. A minimum-time trajectory planning method for two robots. *IEEE Transactions on Robotics and Automation*, 8(3):414–418, 1992.

[5] S. J. Buckley. Fast motion planning for multiple moving robots. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1989.

[6] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[7] C. Ferrari, E. Pagello, J. Ota, and T. Arai. Multirobot motion coordination in space and time. *Robotics and Autonomous Systems*, 25:219–229, 1998.

[8] F. Gravot and R. Alami. An extension of the plan-merging paradigm for multi-robot coordination. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2001.

[9] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic road maps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[10] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991. ISBN 0-7923-9206-X.

[11] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1996.

[12] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical report, Centre for Mathematics and Computer Science, 1989.

[13] S. Leroy, J. P. Laumond, and T. Simeon. Multiple path coordination for mobile robots: A geometric algorithm. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[14] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proc. of the 5th International IPCO Conference*, pages 389–403, 1996.

[15] M. McHenry. *Slice-Based Path Planning*. PhD thesis, University of Southern California, 1998.

[16] H.P. Moravec and A.E. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 116–121, 1985.

[17] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Publisher, Berlin, New York, 1982.

[18] P. A. O'Donnell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1989.

[19] P. Sveska and M. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 1995.

[20] P. Tournassoud. A strategy for obstacle avoidence and its application to multi-robot systems. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 1224–1229, 1986.

[21] C. Warren. Multiple robot path coordination using artificial potential fields. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 500–505, 1990.

[22] S. Zilberstein and S. Russell. Approximate reasoning using anytime algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*. Kluwer Academic Publishers, Dordrecht, 1995.

Economically Coordinated Job Shop Scheduling and Decision Point Bidding - An Example for Economic Coordination in Manufacturing and Logistics -

Wolfram Conen
XONAR GmbH
Wodanstr. 7, 42555 Velbert, Germany
conen@gmx.de

Abstract

We discuss the application of economic coordination mechanisms to scheduling problems in manufacturing and logistics. We use job shop scheduling as a sample problem domain. We study economically augmented job shop problems (EJSP) which comprise valuation information. We demonstrate how instances of EJSP can be mapped to combinatorial job shop auction problems (CJSAP). A discussion of the tractability of approaches that determine optimal solutions identifies the need for heuristics. We suggest a heuristic that combines the merits of economic coordination with the benefits of closely domain-related solution procedures. We conclude that the application of newly suggested and/or well-known economic coordination mechanisms to job shop and related scheduling problems in manufacturing and logistics seem promising.

Keywords: Scheduling, Economic Coordination, Combinatorial Auctions

1 Introduction

The work presented in the following is based on some of the key assumptions of research in economics and game theory. A general description of the scenario under study could be given as follows:

Scenario 1. *A set of actors decides to collaborate. The actors own a set of resources. The primary objective of the collaboration is to utilize this set of resources as economically efficient as possible. Economic efficiency is measured in terms of money. Each actor has individual preferences for resource allocations. This preference information is (to a large extend) private to the actor—it cannot be assumed that the actors reveal their preferences truthfully without an incentive to do so. Paying tribute to the objective and the restrictions, a coordination mechanism*[1] *is designed to repeatedly determine allocations of the available resources to maximize efficiency. Each actor is free to opt out of the collaboration if he is (repeatedly) dissatisfied with the determined outcomes. Each actor acts rationally (within certain limitations) as an utility maximizer.*

To transfer this scenario to a multi agent setting is straightforward: Each actor or each group of actors can be represented by a (potentially computerized) agent. Furthermore, agents may represent parts of the necessary organizational infrastructure (namely the manifested parts of the coordination mechanism). We primarily use agents as a convenient metaphor for the study of coordination issues. In addition, issues relevant for an implementation of the abstract infrastructure, namely computational tractability and communication complexity, are considered. Due to the conceptual proximity of modeled and implemented multi-agent systems, carrying over some of the theoretical results to the implementation is straightforward.[2]

The scenario emphasizes the relevance of the agents' self-interest. We do not assume that the decision to collaborate with other agents implies that an agent is willing to reveal all his private information or that he is willing to deviate dramatically from his utility-maximizing behavior and turns into an altruist. The reasons for the formation of collaborations can be manifold—most relevant to our analysis is the assumption that the actors decide to become part of the collaboration because they consider this to be the economically most beneficial behavioral alternative, given their expectations and their attitude towards risk. The assumption of rational behavior implies that the agents re-assess this evaluation continually and, consequently, may decide to leave the collaboration. This also explains, why an objective of the design of the coordination mechanism is to give incentives to participate in the mechanism continually.

Also note that this scenario is an abstraction of many real-world scenarios in logistics and manufacturing: a number of units compete for the utilization of resources that are owned by the group of units (or by some larger unit that encloses/owns the considered units). Each unit is, to a certain extent, economically independent in its decisions and has information that is often not only local but also private in the strict sense: it is only communicated truthfully to other agents if an incentive to do so is present. This situation can be found in virtual or extended enterprises (consider, for example, a collaboration of small and medium-size transportation companies with overlapping transportation capacities) or even in comparatively small production environments such as a single workshop (groups of workers, sales people, shop-floor managers, planners influence as actors with self-interest and (conflicting) individual objectives the orga-

[1] To explain the use of the term *coordination*: The mechanism is executed to coordinate the interests of the actors and to determine an outcome that implements a coordination of the behaviors of the agents—both with respect to the resources.

[2] For example, if the implementation of a MAS that was modeled as a distributed system is also distributed, the (theoretical) communication complexity results can easily be turned into approximations for communication latency if the details of the underlying networking infrastructure are worked into the formulas.

nization and enactment of work in the workshop). We chose job shop scheduling as an example for this type of coordination problems.

There might be more natural application areas for value-augmented scheduling/planning problems than job shop scheduling problems (JSP). However, both the inherent computational complexity and the structural simplicity make job shop problems a good target for the demonstration of the applicability of economic coordination mechanisms to value-augmented scheduling/planning or, more general, economic resource allocation problems. Additionally, the work that has been done in areas like holonic manufacturing call for an application of the techniques that are outlined below because it allows us

1. to model jobs, machines, production managers, sales officers, planners etc. uniformly as autonomous, self-interested entities with individual tasks to fulfill and individual goals to accomplish.
2. to study directly the economic impact of decisions across all inter-related units of operations and planning (as long as the interdependencies are also mapped into the problem, which they will be if the actors driving the operations get a handle to influence decisions transparently—giving them useful budgets and allowing for active payments is one possible way, see (Adelsberger, Conen, & Krukis 1995)).
3. to apply a large number of relevant results from game theory and microeconomics to the analysis of operational and structural phenomena in manufacturing/logistics systems and beyond. This includes strategic considerations (with the possible goal to design mechanisms to give incentives to the parties interested in the result to communicate their preferences truthfully), economic efficiency (a goal can be to design coordination mechanisms that compute economically efficient solutions), participation constraints (design the mechanism to make sure that the participants feel as being treated in a fair manner to ensure their continued participation) etc.
4. to design mechanisms that scale, both vertically and horizontally, because they speak a language that is understood at every level of a company, between companies, and between companies and customers: the language of value, expressed in terms of money.

To be specific: The goal of our paper is to demonstrate the applicability of economic coordination mechanisms to economically augmented scheduling problems, and to offer solutions for computing economically efficient and/or straightforwardly computable heuristic solutions that are tailored to the problem domain. We also discuss some of the intricacies of economic coordination such as strategic agent behavior, equilibrium considerations and complexity issues.

1.1 Related Work

Related work is numerous, basically all of microeconomic literature is motivated by and related to resource allocation problems. The study of scheduling problems in an economic context has also a significant tradition in AI literature. For an excellent overview with an emphasis on economics, as well as for interesting and significant recent results related to both areas, see (Parkes 2001) or (Wurman 1999). For an instructive overview of methods and problems in the wider context of self-interested agents, consider (Sandholm 1996), whose further work, for example (Sandholm 2002), also contributes significantly to advances in the area. A relevant example for work related to the economics of scheduling is also the work of Wellman et. al (Walsh & Wellman 1998; Wellman *et al.* 2001; Walsh, Wellman, & Ygge 2000). More closely related to the application of economic principles in manufacturing environments are, for example, the work of Van Dyke Parunak (Parunak 1996) or A.D. Baker (Baker 1996).[3] More recently, this has also been discussed in the context of new modes of manufacturing, for example Holonic Manufacturing, as a promising paradigm for controlling scheduling and planning processes in a distributed environment with local goals, private information and general efficiency objectives (see, for example, (Adelsberger & Conen 2000)).

From an economic perspective, a key issue in the study of resource allocation problems is the design of coordination mechanisms that

- enable the computation of economically efficient solutions. In domains where monetary valuations are available (as we assume below), this amounts to determine allocations maximizing the aggregated welfare of the participating agents. Note that in a setting with private information, this requires that the mechanism gives some incentive to the agents to reveal their preferences truthfully.

- satisfy participation constraints for rational agents. A mechanism is considered to be individually rational if the agents can expect participation to be beneficial. It is also relevant that the agents are satisfied with the outcome of the mechanism to ensure further participation. The question of satisfaction boils down to answer the question to what extend it is possible for the agent to realize his most-preferred solution in a given situation.

In view of related work, the work presented here seems justified as most of the results that have been obtained for general resource allocation problems with an emphasis on economics have not been specifically tailored towards shop floor and related scheduling problems in manufacturing or logistics. Furthermore, recent results in studying combinatorial auctions and exchanges show that not all questions have been answered yet (relevant work is mentioned throughout the paper). Also, economically motivated approaches in scheduling/planning literature are sometimes difficult to analyze with respect to the key issues outlined above or tackle simplified settings.

[3] Though the concept of (Pareto) efficiency has been introduced much earlier into the scheduling literature, see (Wassenhove & Gelders 1980).

1.2 Overview

First, the basic notation and terminology for discussing job shop scheduling problems (JSPs) is introduced. We define the class of economically augmented job shop problems (EJSPs), where job agents have utility for schedules, and related them to typical minsum criteria. We map these problem to combinatorial job shop auction problems (CJSAPs). We introduce prices as a means to design individually rational, economically efficient coordination mechanism and discuss the issues of truthful revelation and the Vickrey principle. We show that both EJSP and CJSAP are (strongly) NP"=hard. In view of this result, we suggest a heuristic, economically-driven coordination mechanism that is build on top of a well-known, domain-specific solution procedure, the Giffler and Thompson algorithm. This decision point bidding approach can be applied to a number of search-based solution procedures. It is an example of an economic coordination mechanisms that is tailored to fit the problem domain. This, in turn, may allow participating agents to fully comprehend and purposefully influence the execution of the mechanism by means of bidding. The paper is concluded with a brief discussion of possibilities to extend and apply the obtained results.

2 From Job Shop Scheduling to Economic Coordination

Scheduling allocates resources over time to enable the execution of tasks. An important subclass of scheduling problems are job shop problems. The tasks are given by a set of jobs. Each job consists of a sequence of operations that have to be performed on specific machines in a given order. We base the following on the constraint optimization version of discrete, deterministic Job Shop Scheduling (JSS) as defined in (Ausiello *et al.* 1999).[4]

Definition 1 (Job Shop Scheduling – Basic Setting). *An instance of the class of job shop scheduling problems (JSP) consists of a set $M = \{1, \ldots, m\}$ of m machines, and a set $J = \{1, \ldots, n\}$ of n jobs, each consisting of a set $O_j = \{o_j^1, \ldots, o_j^{n_j}\}$ of n_j operations. For each such operation, o_j^i, a machine $m_j^i \in M$ and a processing time $p_j^i \in \mathbb{N}$ is given. A ready time, $r_j \in \mathbb{N}_0$, denotes the earliest possible start time for the first operation of each job $j \in J$.*

In a straightforward notation, a job j is given as $(r_j, [(m_j^1), p_j^1), \ldots, (m_j^{n_j}, p_j^{n_j}])$. The following example is used throughout the paper:

Example 1. *Job 1:* $(0, [(2,2), (1,3), (3,2)])$
Job 2: $(0, [(3,1), (2,2), (1,2)])$

In the following, we refer to an arbitrary, but fixed JSS problem P. A *potential schedule* for a set of operations is a mapping from the operations to start times. A potential schedule which assigns start times to all operations in P is called *complete*. A potential schedule is called *feasible* if (1) no first operation starts too early, (2) no sequence constraint is violated, and (3) no overlap in the processing times of assigned operations occurs on any machine. A schedule that is complete and feasible with respect to P is called a *valid* schedule or a *solution* of P (s. Fig. 1 for an example). Formally:

Definition 2 (Valid Schedule). *Given an instance P of JSP. A mapping $s : \cup_{j \in J} O_j \to \mathbb{N}_0$ is a* valid schedule *for P iff (1) $o_j^1 \geq r_j$ for all $j \in J$, (2) $s(o_j^i) + p_j^i \leq s(o_j^{i+1})$ for all $j \in J$ and all $i \in \{1, \ldots, n_j - 1\}$, and (3), for every pair of distinct[5] operations o_j^h, o_k^i, either $m_j^h \neq m_k^i$ or $s(o_j^h) + p_j^h \leq s(o_k^i)$ or $s(o_k^i) + p_k^i \leq s(o_j^h)$.*

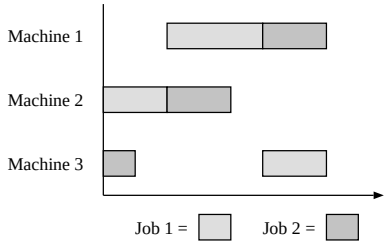

Figure 1: The valid schedule $(0,2,5)_1, (0,2,5)_2$ (schedules are given as n_j-ary sequences of start times for each job j) for the example 1.

We may study a problem P relative to a time horizon $[TS, TE]$, $TS, TE \in \mathbb{N}_0$ and write $P^{[TS,TE]}$. The set of all valid schedules restricted to a specific time horizon is denoted with $S^{[TS,TE]}$. If the time horizon is irrelevant, no index is shown. For a given set S, the subset S_j contains all schedules that are restricted to the operation of a given job j. For a given schedule s, the subschedule s_j denotes the elements of s that schedule operations of job j.

We now turn our attention to comparisons of the *quality* of schedules. Often measures that depend on the completion times of the jobs are used, for example, minimize $C^{max} = \max_{j \in J} C_j$, where $C_j = s(o_j^{n_j}) + p_j^{n_j}$ is the time of completion of the last operation $o_j^{n_j}$ of job j in a given (valid) schedule s. Regularly it is also assumed that each job has a due date d_j which allows to use derivations from this due date as a measure.

In a value-oriented business environment, this way of measuring the quality of schedules is only convincing if the measure has a close correspondence to the cash-flow pattern that is induced by the imposed solutions (usually, it is tried to find a solution that optimizes the chosen measure or that approximates an optimal solution). With respect to the core measure of the success of management and operations, ie. value, such time (or capacity etc.) related measures can only be considered as being surrogates that may render decisions intransparent because they introduce incomparabilities between manufacturing shops competing for resources and give no handle to tie local decisions to the decisions of upstream and downstream manufacturing/logistics units.

[4] Which, in turn, is the optimization version of the decision problem SS18 as defined by Garey and Johnson in their seminal work (Garey & Johnson 1979) – with one difference: the additional constraint of Garey and Johnson which required that each pair of consecutive operations has to be performed on different machines has been dropped.

[5] That is, $j, k \in J$, $h \in \{1, \ldots, n_j\}$, $i \in \{1, \ldots, n_k\}$ and, if $j = k$, $h \neq i$.

This, however, would be an important prerequisite for an unified analysis of the plans, schedules, decisions and operations of all interrelated business units.

But even within one shop, it is usually not trivial to map the goals that are related to jobs to *one* measure only. Some jobs may be produced directly to customer orders (due date is important), some jobs are produced to fill up the stock (minimizing production cost is important) etc. The resulting multi-criteria problem does not seem to be convincingly solvable without introducing a unifying measure. As the *value* related to strategic, tactic, and operational decisions determines the success of business operations, mapping goals to value and measuring quality with money is a natural choice.

We will now augment job shop problems with value information and view them as economic coordination problems.

2.1 JSPs as Economic Coordination Problems

Job shop scheduling problems can be extended to economic coordination problems—the jobs are identified with agents competing for resources. The goods to be traded are slots of machine time. To augment a JSS problem P economically, we assume that each agent has utility for schedules, that is, a value function $v_j^{'} : S \to \mathbb{N}_0$ is available for every job agent j which assigns a non-negative value to each possible valid[6] schedule.

Definition 3 (EJSP). *An instance P (possibly restricted to a time horizon) of the class JSS of job shop scheduling problems and a set V of j value functions $v_j^{'} : S \to \mathbb{N}_0$, one for each job j, defines an instance EP of the class of economically augmented job shop scheduling problems (EJSP).*[7]

The general objective of solving EJSPs is to select a schedule from the set of possible valid schedules that achieves *economic efficiency*:

$$\arg\max_{s \in S} \sum_{j \in J} v_j^{'}(s) \qquad (1)$$

As we show below, this class of scheduling problems encompasses a number of traditional job shop scheduling problems. To do so, we first consider a restricted variant of the EJSP, where, for each agent j, the value of a schedule does only depend on the operations in O_j. In the general variant, an agent j may value two schedules differently though both assign the same start times to the operations in O_j. For example, this allows an agent k to express that she prefers if a machine, say 2, is assigned to agent l instead of agent m in the interval $[3, 5]$. This expressiveness is not always required. [8]

[6]It can also be useful to consider incomplete feasible schedules.

[7]Remember that S is the set of all valid schedules that solve P. If a time horizon is given, this set may be empty. If no time horizon is given, the set is countably infinite. We will therefore usually assume that a time horizon is given (without displaying it).

[8]Though, to map a very common objective, namely max completion time, it is required, see the footnote below.

Definition 4 (EJSP without allocative externalities). *Let EP be given as above. EP is an EJSP* without allocative externalities *iff, for any agent j and any pair of schedules $a, b \in S$, such that the restriction a_j of a to operations of j coincides with the restriction b_j of b, $v_j^{'}(a) = v_j^{'}(b)$.*

Now consider a JSS problem $P^{'}$ and a typical minsum criterion (Hoogeveen, Schuurman, & Woeginger 1998), *total job completion time*. We map this $J||\sum C_j$ problem to a restricted EJSP $EP^{'}$ as follows:

First, determine a valid schedule by timetabling the operations of job 1 as early as possible and without slack and proceed with the operations of job 2, starting with o_2^1 at time C_1. Continue this until all operations are scheduled. This produces (with effort linear in the number of operations) a valid schedule $s^{'}$ of length $l = \sum_{j \in J} \sum_{1 \le i \le n_j} p_j^i$. This determines the time horizon $[0, l]$ to be considered. Now, a value function $v_j^{'}(\cdot)$ for each agent j can be given that determines, with effort linear in the number of operations, the value of any given input schedule:

Function $v_j^{'}$(In: Valid Schedule s, Out: Value v)
$C_j \leftarrow s(o_j^{n_j}) + p_j^{n_j}$;
$v \leftarrow l - C_j$; **return** v;[9]

Proposition 5. *A schedule maximizes the economic efficiency in $EP^{'}$ over all schedules in S if and only if it minimizes the minsum criterion* total completion time[10] *in $P^{'}$ over all schedules in S.*

Proof. Let s be a schedule that maximizes efficiency in $EP^{'}$. Assume that s does not minimize the total completion time in $P^{'}$, that is, there is a schedule $r \in S$ such that $\sum_{j \in J} s(o_j^{n_j}) + p_j^{n_j} > \sum_{j \in J} r(o_j^{n_j}) + p_j^{n_j}$, or, shorter, $\sum_{j \in J} s(o_j^{n_j}) > \sum_{j \in J} r(o_j^{n_j})$. Because s maximizes efficiency in $EP^{'}$, $\sum_{j \in J}(l - s(o_j^{n_j}) + p_j^{n_j}) \ge$

[9]Note that this transformation is polynomial because it makes use of the fact that there is significant structure in the problem. Would we have to enumerate all (or almost all) schedule/value pairs explicitly, the transformation would not be polynomial. However, criteria that would require such an effort are virtually never used in scheduling literature (they would be random in the sense of algorithmic complexity theory, (Li & Vitanyi 1997)).

[10]Similar results can be obtained for restrictions to other minsum criteria (e.g., total tardiness, weighted total completion/tardiness, holding costs, early/tardy penalties). Note also, that EJSP *without allocative externalities* cannot model criteria like $C^{\max}$, because a global optimum for the desired criterion is not obtainable from local considerations. The jobs *cannot* be modeled as being independent in this case. If the value function should reflect the benefit of achieving a minimal $C^{\max}$, they *must* reflect this dependency in their valuation of schedules, or otherwise, self-interest prevents the optimization of the desired criterion. For $C^{\max}$, the jobs should value schedules with an earlier completion time for all jobs higher than, for example, schedules that give them an earlier individual completion time (the time the last *operation* of j finishes) but a later overall completion time (the time the last *job* finishes)—in other words, the value of a schedule does not only depend on the operations in O_j but on all (final) operations (that is, allocative externalities are present).

$\sum_{j\in J}(l - r(o_j^{n_j}) + p_j^{n_j})$, or, written differently, $\sum_{j\in J} l$ $- \sum_{j\in J} s(o_j^{n_j}) - \sum_{j\in J} p_j^{n_j} \geq \sum_{j\in J} l$ - $\sum_{j\in J} r(o_j^{n_j})$ $- \sum_{j\in J} p_j^{n_j}$ respectively $\sum_{j\in J} r(o_j^{n_j}) \geq \sum_{j\in J} s(o_j^{n_j})$, contradicting the assumption. The other direction follows immediately as well. □

Proposition 6. *EJSPs are NP-hard in the strong sense.*

Proof. As above proposition shows, EJSP can be restricted to $J||\sum C_j$. Furthermore, the transformation is polynomial (see above). From the strong NP-hardness of $J||\sum C_j$ (s. (Lawler *et al.* 1992) or (Hoogeveen, Schuurman, & Woeginger 1998)) the proposition follows. □

To adapt the problem to our economic setting with self-interested, (bounded) rational agents, the following assumption are necessary.

In line with our basic motivation, we assume that the value function is *information private to the agent*, that is, *no (central) institution has access to this information without prior consent of the agent*. In addition, *utility is transferable*[11] between agents (ie, a meaningful currency has to be available to express valuations and to transfer payments).

We also assume that no agent can be forced to act against his will. With these assumptions, we can translate economically augmented job shop scheduling problems to a specific subclass of combinatorial auction problems (CAPs), which are currently studied extensively in AI and microeconomic literature (s. (de Vries & Vohra 2001) for a survey).

2.2 Transforming EJSP to CJSAP

Let EP be in EJSP and let $[TS, TE]$ be a time horizon. An economic coordination problem can now be formulated as follows:

Agents: A set of n job agents, $N = J = \{1, \ldots, n\}$, and an arbitrator, 0. The arbitrator is modeled as a *supplier*. The job agents are modeled as *consumers*.

Goods: The set of goods, Ω, is the set of all machine-specific unit intervals within the time horizon (remember that M is the set of machines), that is:

$$\Omega = \{[z, z+1]_i \mid i \in M \wedge z = TS, \ldots, TE - 1\}$$

Any subset $B \subseteq \Omega$ (alternatively writable as $B \in 2^\Omega$, the power set of Ω) is called *bundle*.

Before we can define value functions, a function $A : S \rightarrow 2^\Omega$ that partitions a schedule s into the covered machine-specific unit intervals is required (note that s is a function, and thus a relation, so that it is appropriate to write $(o, t) \in s$ instead of $t = s(o)$).

$$A(s) = \{[z, z+1]_m \mid (o_j^i, t) \in s \wedge m = m_j^i \wedge z \in \{t, \ldots, t + p_j^i - 1\}$$

Value functions: The value function of consumer j, $v_j : 2^\Omega \rightarrow \mathbb{N}_0$ is defined as follows: if at least one $s \in S$ exists such that the schedule s is covered by the unit intervals in B (ie., $A(s) \subseteq B$), the value of B is set to the value (in the underlying EJSP) of the best schedule among the covered schedules, that is, $v_j(B) = v_j^*(B)$ with $v_j^*(B) = \max_{\{s:s\in S\wedge A(s)\subseteq B\}} v_j^{'}(s)$. If no schedule is covered by B, $v_j(B)$ is set to 0.

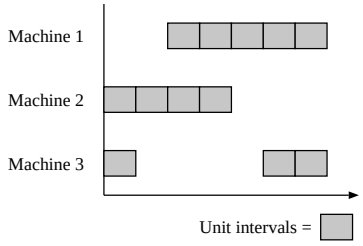

Figure 2: The unit intervals (or *time slots*) of example 1.

Example 2. *Reconsult example 1. The time horizon to be considered is* $[0, 9]$. *The following consideration leads to the preference relations, which underly the value functions: let* j *be a job agent and* s_1 *and* s_2 *be schedules which are complete with respect to* j. *If the last operation of agent* j *in* s_1 *is completed earlier than* s_2*, agent* j *prefers schedule* s_1 *(written as* $s_1 \succ s_2$*). If the completion times are equal, the agent is indifferent between* s_1 *and* s_2 *(written as* $s_1 \sim s_2$*).*

For agent 1 the preference relation given below follows. for convenience, a rank is assigned to the equivalence classes of equally preferred schedules.

1: $(0, 2, 5) \succ$
2: $(0, 2, 6)$, $(0, 3, 6)$, $(1, 3, 6) \succ$
3: $(0, 2, 7)$, $(0, 3, 7)$, $(0, 4, 7)$, $(1, 3, 7)$, $(2, 4, 7) \succ$
4: All invalid schedules.

For agent 2, the preference relation is only partially displayed:

1: $(0, 1, 3) \succ$
2: $(0, 1, 4)$, $(0, 2, 4)$, $(1, 2, 4) \succ \ldots$
3: $(0, 1, 5)$, $(0, 2, 5)$, $(0, 3, 5)$, $(1, 2, 5)$, $(1, 3, 5)$, $(2, 3, 5) \succ$
4: $(0, 1, 6), \ldots \succ$ *5:* $(0, 1, 7), \ldots \succ$ *6: All invalid schedules.*

Agent 1 values the earliest possible completion (at 7) with 20 currency units (CU) and agent 2 (at 5) with 16 CU. a delay per time unit reduces the value of the schedule for agent 1 by 3 CU and for agent 2 by 2 CU, resulting in the following functions $v_j^R(\cdot)$*, which assign values to ranks:*

Agent 1: $v_1^R(1) = 20$, $v_1^R(2) = 17$, $v_1^R(3) = 14$, $v_1^R(4) = 0$.

Agent 2: $v_2^R(1) = 16$, $v_2^R(2) = 14$, $v_2^R(3) = 12$, $v_2^R(4) = 10$, $v_2^R(5) = 8$, $v_2^R(6) = 0$.

For a given bundle B*, its valuation can be determined by mapping* B *with a function* r *to the preference rank which corresponds to the best schedule which is* covered *by* B. *For example for the largest bundle,* Ω*, which covers all schedules that lie within the time horizon, rank 1 is the result of the mapping for both agents, that is* $v_1(\Omega) = v_1^r(r(\Omega)) = v_1^r(1) = 20$ *respectively* $v_2(\Omega) = 16$.

[11] Technically, the value functions have to be quasilinear in money, compare, for example, (Mas-Colell, Whinston, & Green 1995). Quasilinearity allows to interpret the utility for a good (time slots in our case) as the willingness to pay for it.

Proposition 7 (Monotony, Free Disposal). $v_i(A) \leq v_i(B)$ *for all* $A \subseteq B \subseteq \Omega$, $i \in N$.

Proof. This follows immediately from the construction of the value functions: a bundle B that contains at least as many unit intervals as a bundle A covers at least the schedules that A covers, thus its value cannot be smaller than the value of A because the maximum value over all covered schedules in the EJSP instance is at least as large for B as for A. □

This is sometimes also called *Free Disposal* because adding further unit intervals will not reduce value, or, in other words, superfluous goods can be disposed off at no cost.

We also assume that *no budget restrictions* exist: every consumer j is in possession of enough money to be able to pay up to the amount of the valuation for his most preferred bundle. The consumers have no endowment beyond money. All goods belong to the arbitrator. If the consumer does not receive any good, his utility shall be 0 (w.l.o.g).

The objective of the allocation of the goods (from the perspective of the arbitrator) is to maximize the aggregated utility of the consumers. Here, this directly corresponds to maximizing the sum of individual utilities. An allocation[12] $X^* = (X_0^*, \ldots, X_n^*)$ conforms to this objective if and only if the allocation is *efficient*, that is, X^* has to be a maximizer for the following problem

$$\max_X \sum_{j=1}^{n} v_j(X_j) \quad (X \text{ iterates over all allocations}) \quad (2)$$

Example 3. *For the above example, the following allocations are efficient:*
$X_1 \supseteq \{[0,1]_2, [1,2]_2, [2,3]_1, [3,4]_1, [4,5]_1, [5,6]_3, [6,7]_3\}$
$X_2 \supseteq \{[0,1]_3, [2,3]_2, [3,4]_2, [5,6]_1, [6,7]_1\}$
$X_0 = \Omega \backslash (X_1 \cup X_2)$.

Consecutive unit intervals can be consolidated to bundles in an abbreviating notation:
$X_1 \supseteq \{[0,2]_2, [2,5]_1, [5,7]_3\}$
$X_2 \supseteq \{[0,1]_3, [2,4]_2, [5,7]_1\}$
$X_0 = \Omega \backslash (X_1 \cup X_2)$.

Definition 8 (CJSAP). *Let EP be an instance of EJSP. The sets Ω and N, the value functions $v_j(\cdot)$ of the consumers that are obtained by the above transformation of EP, and the objective* (2) *of the arbitrator define an instance C of the class of* Combinatorial Job Shop Auction Problem, *CJSAP.*

We call an allocation that conforms to the objective of the arbitrator a *solution*.

Proposition 9. *A solution exists for every possible instance of a CJSAP.*

Proof. This follows with a straightforward combinatorial argument immediately from the finiteness of the problem (which we assume throughout the paper). □

[12] That is an $(n+1)$-ary partition of the set of goods, Ω, which assigns to agent j the goods in the bundle X_j (some X_j may be empty, so it is not a partition with non-empty subsets, as it is usually assumed).

Proposition 10. *For any $EP \in$ EJSP, if a valid schedule exists, a solution of the corresponding problem $C \in$ CJSAP can be transformed into an optimal schedule for the original, economically augmented job shop scheduling problem EP (and vice versa).*

Proof. Let $X = (X_0, X_1, \ldots, X_n)$ be a tight solution[13] of C. Construct a schedule s^X from X as follows: for each agent $j \in J$, find the best schedule s_j^X that is covered by X_j by simply timetabling the operations as unit intervals are available (due to the tightness of X, this is possible in cost linear to the number of time slots in X). Combine the schedules s_j^X to the schedule s^X. Note that the construction of C ensures that this construction is always possible if a valid schedule exists. Furthermore, as the allocation X is a partition of Ω, no overlap on a machine can occur. The other direction is even simpler: each reservation for an operation in the optimal schedule can be split into the machine-specific unit intervals. The information in the schedule can be used to assign the unit intervals immediately to the correct part of the allocation (cost linear to the processing time). □

We will now turn our attention to the *problems related to finding an efficient allocation*. They can be outlined as follows: (1) a certain amount of value information is necessary to determine an efficient allocation; (2) once the required information is available, the actual computation has to be performed; (3) an incentive has to be given to the agents to report their part of the required information truthfully, or otherwise, efficiency of the solution cannot be guaranteed; and (4) the agents have to be satisfied with the determined outcome. We neglect issue (1), briefly discuss issue (2) and concentrate on (3) and (4). We will, however, return to the issue of complexity later.

2.3 Determination of Efficient Allocations

If we assume for now that the (complete and true) value functions of all agents are known, an efficient allocation of the unit intervals to the job agents can be computed with one of the well-studied methods of winner determination (compare (Sandholm 2002; Sandholm *et al.* 2001; Fujishima, Leyton-Brown, & Shoham 1999)). To save communication, approaches have been suggested that only partially reveal the value functions of the consumers (compare (Parkes 2001) for indirect, iterative auctions and (Conen & Sandholm 2002b) for a progressive direct mechanism based on (Conen & Sandholm 2001b; 2001a)). Note that the winner determination problem is essentially a set-packing problem (Rothkopf, Pekeč, & Harstad 1998; de Vries & Vohra 2001) and that it is NP-hard. We show below that this also holds for CJSAP.

[13] A tight solution is a solution that does not contain time slots that are not used. Note that a tight solution always exists because the value of a bundle is the value of the best covered schedule, and, in consequence, the bundle that covers just the best schedule, is tight and optimal (that is: unused slots cannot add to the value of a bundle due to the construction of the value functions). Further note that is it always possible to tighten a solution with costs less than exponential.

2.4 Prices

In the above example, two core problems remain: first, because only agent 1 can realize his best alternative, agent 2 will envy him. Second, we have tacitly assumed that the agents report their utility truthfully—but why should they do so under our assumption of preferences being private information? Certainly, agent 2 could expect to benefit from over-exaggerating his valuations. If agent 1 would expect agent 2 to over-exaggerate, he would do the same, and so forth. Without an additional way to ensure satisfaction and to make lying unattractive, we cannot expect to compute allocations that are efficient with respect to the true preferences. The way to go is to introduce prices. Each consumer has a net utility function which reflects the impact of prices (negative transfers in the following definition) on the realizable utility.[14]

Definition 11 (Net utility). *The* net utility function $u_j(\cdot) : 2^\Omega \times \mathbb{Z} \rightarrow \mathbb{Z}$ *for each consumer* j *is defined as* $u_j(x,t) = v_j(x) + t$.

To keep every consumer satisfied with the outcome (consisting of allocation and payment), $u_j(X_j, p) \geq u_i(A, p)$ must hold for every consumer j and every bundle $A \subseteq \Omega$,that is, the net utility of the bundle he receives must be at least as good as it would be for any other bundle at the given prices (or, otherwise, the consumer would prefer to receive the bundle that gives him the best net utility).

This leads immediately to the notion of equilibria. An outcome, that is, an allocation and a payment vector (determined by the prices), is an equilibrium if both sides of the market are satisfied with it (the market is *cleared*). In our case this is true if the above condition holds for all consumers and if the allocation is efficient (to satisfy the supplier). There are different restrictions that one can impose on prices—prices for bundles have to be the sum of prices for individual goods (see (Gul & Stacchetti 1999; Kelso & Crawford 1982), prices are independent for every bundle (see (Wurman & Wellman 2000)), prices are determined for the bundles in the efficient allocation and prices for bundles of these bundles are additive (see (Conen & Sandholm 2002a)). Only the independent pricing of all bundles can guarantee the existence of equilibrium price vectors. However, implementing the determined outcome may require enforcement.[15] For the more natural pricing modes, equilibrium price vectors need not exist.[16] We will therefore not discuss (anonymous) equilibrium pricing in detail and turn our attention to a solution concept for which solutions always exists: Vickrey payments. We will demonstrate in the example below that implementing Vickrey payment-based coordination mechanisms give the participating agents no incentive to lie (a very relevant design objective in a private information setting). Please, consider the continued example below.

[14]Note, that we make the usual assumption of quasi-linearity of valuations.

[15]Each agent is only allowed to buy one bundle—thus, if he is interested in AB and we have $p(AB) = 6$, $p(A) = 2$, $p(B) = 2$, he would want to enter the auction with two identities to buy A and B separately.

[16]This negative result holds also for instances of CJSAP.

Example 4. *Bundles on offer in the efficient allocation:*

$$A = \{[0,2]_2, [2,5]_1, [5,7]_3\}$$
$$B = \{[0,1]_3, [2,4]_2, [5,7]_1\}$$

Demand for these bundles:

	$\emptyset$	A	B	AB
Agent 1	*0*	**20**	*0*	*20*
Agent 2	*0*	*0*	**12**	*16*

Vickrey Payments (the vector of payments also fulfill the equilibrium condition if interpreted as prices for the bundles instead of personalized payments):

	A	B	AB
Agent 1's payment	*4*		
Agent 2's payment		*0*	
Equilibrium prices	*4*	*0*	*4*

The payments ensure that there is no (individual) incentive for the agents to lie, as can be seen as follows. First note that the price each agent has to pay does only depend on the reported utilities of the competing agents—it represents the loss of utility that the other agents experiences due to the participation of the former agent. Now, assume that agent 1 would underbid his valuation with, say, 17. Then he risks that agent 2 (or any other agent) would bid just above 17 but below 20, say 18, and would thus receive the good. As the price he has to pay is independent of his own bid and will equal the bid of the other agent, he could have done better by bidding truthfully (exactly 2 = 20 - 18 instead of 0). If he would have known beforehand what the other agent will bid, say x, he would not have a reason to underbid either, because there would be no difference in net value for agent 1 in bidding 20 or $x + \epsilon$ (as the price will be the bid of the other agent anyway). He would also not overbid, because he risks that he receives the overbidded bundle for a price between his true valuation and his bid and would, thus, realize a loss. If he would overbid in the fully-informed situation, he cannot gain any net value from it either. An analogous reasoning applies to agent 2, thus both agents will not have an incentive to misrepresent their valuation if they act rational (this corresponds to an equilibrium in dominant[17] *strategies).*

The general principle invoked here has been mentioned in the example: the payments that an agent has to transfer do not depend on her own bid but captures instead the effect of her participation on the other participating agents.[18] It is intuitively clear that, as soon as there is a dependency for a bundle X_j between the reported utility of agent j and the price j has to pay, j will have an incentive to minimize this price by misrepresenting his utility if possible. To do so, he might start to collect information about the other participating agents (which is not necessary above) to become able to behave *strategically*. This, in turn, may make it impossible for the arbitrator to pick the efficient allocation—all that he could do would be to pick the allocation that is efficient with respect to the *reported* utilities. This brief digression into the issue of *incentive compatibility* may suffice to

[17]Weakly dominant in the case of informed agents.

[18]The principle has been discovered and applied independently by Vickrey (in 1961), Clarke (in 1971) and Groves (in 1973) (see, for example, (Vickrey 1961)).

demonstrate one of the most prevalent problems in environments where the agents have private information: the problem of eliciting their preference truthfully to allow for truly optimal decisions.[19]

Proposition 12 (Existence of Vickrey Outcome). *An outcome consisting of an efficient allocation and a vector of related Vickrey payments exists for every instance of CJSAP.*

Proof. (Sketch) The proof follows from a straightforward combinatorial argument: with finite sets of bundles and agents. A computable solution of the maximization problem (2) (to determine the efficient allocation) and the n (or less) maximization problems following from the initial problem by leaving out, for each non-empty bundle in the allocation, the agent that receives it (to determine the effect of his participation, that is the Vickrey payments), is immediately available from enumerating all possible complete and reduced allocations and picking the optima. □

3 Complexity Issues

Two immediate problems of auction mechanisms that try to solve a CJSAP are (1) that the mechanisms may require communication that is exponential in the number of unit intervals (compare the general result of (Nisan & Segal 2002), which extends to CJSAP) and (2) that solving the actual allocation problem once all required value information is received is NP-hard, as the following corollary demonstrates:

Corollary 13. *CJSAPs are NP-hard in the strong sense.*

Proof. This follows immediately from Proposition 6 and the polynomiality of the transformation given in Proposition 10. □

Note also that CJSAP is a subclass of the combinatorial auction problem CAP which is equivalent to maximum weighted set-packing (compare (de Vries & Vohra 2001)), known to be NP-hard.[20]

Consequently, we cannot expect to obtain optimal solutions for problems of reasonable size with reasonable computational efficiency. Furthermore, the approximability results obtained for maximum weighted set packing (Ausiello *et al.* 1999) are not very encouraging. We also do not expect that CJSAP is an especially well-behaved subclass in this respect due to its proximity to EJSP and, thus, typical job shop scheduling problems. We can now try to modify auction mechanism that determine efficient allocations (like iBundle (Parkes & Ungar 2000) or AkBA (Wurman & Wellman 2000)) to relax the efficiency goal and to restrict the search space heuristically. However, we cannot expect this to be easily justifiable with respect to the properties of the initial problem domain.

[19]For pointers to recent work, see (Bikhchandani *et al.* 2001) or Parkes (Parkes 2001).

[20]If we allow partially ordered sequences of operations and alternative routings, the corresponding variant of EJSP and the resulting CJSAP could be mapped to the general combinatorial auction problem bijectively.

It is the nature of any heuristic modification that no guarantee to obtain economically efficient solutions can be given. Furthermore, other desirable properties like incentive-compatibility may get lost. With respect to a specific problem domain, it seems reasonable to postulate that the heuristic reflects the properties of the domain naturally, allowing users to make justifiable decisions when dealing with the heuristic mechanisms. We suggest a mechanisms that is closely related to a straightforward solution procedure for job shop scheduling, namely the algorithm of Giffler and Thompson (GT). It allows the agents (resp. the represented actors) to base their bidding behavior on an understanding of the consequences of decisions that are made along an execution of GT.

4 Decision Point Bidding

Most solution procedures for JSPs (or for other resource allocation problems) can be viewed as a sequence of decisions—for the GT, this is the selection of the next operation to schedule. Now, instead of competing for reservations directly, the agents bid for the right to stipulate decisions.

As GT constructs an active schedule, we will restrict our analysis to problem instances of EJSP that have the property that the individual agents always (weakly) prefer earlier allocation of the final operation (to mimic the behavior of regular measures).
We give a brief re-collection of the GT prior to outlining a coordination mechanism that allows to solve restricted EJSP by following the sequence of decisions that are characteristic for an execution of GT. Assume that an instance EP of EJSP is given.

(1) Initialize the set SO of schedulable operations to $o_j^1, 1 \leq j \leq n$.
(2) As long as SO is not empty do
(3a) Determine the decision set DS by first picking from SO one of the operations with the earliest completion time, say o_x^y (ie. $\arg\min_{o_j^i \in SO} r_j + p_j^i$) and
(3b) insert into DS all operations from SO that require m_x^y and start before the potential completion time $r_x + p_x^y$ of o_x^y
(4) (**Decision Point**) Choose one of the operations from DS, say o_a^b and make reservations corresponding to its related ready and processing times.
(5) Update the ready time information of all jobs that have operations in DS by setting r_j of any such job to $r_a^b + p_a^b$. Remove o_a^b from SO and add its successor (if it exists, ie. if $b < n_j$), o_a^{b+1} to SO. Empty DS.

Now, the basic mechanism is as follows: the arbitrator announces to run a GT and starts to ask the agents for information about ready times and about the processing times of the operations to be scheduled. It iterates the Giffler and Thompson procedure as usual, requesting bids from the agents to be able to determine a decider for each decision point (see above). Each agent who has currently an operation in the decision set is entitled to

bid. The winner will pay the price of the second highest bid and receives, as a result, the earliest possible reservation of his operation in the decision set (ties in the bids are broken arbitrarily).[21]

Corollary 14. *For any finite EJSP, the mechanism terminates and the resulting allocation corresponds to an active schedule.*

This follows immediately from the construction of the mechanism, which follows the execution pattern of the GT and, thus, inherits its property of computing an active schedule.

Proposition 15. *For the restricted EJSP, the set of possible outcomes of the mechanism contains an efficient allocation.*

Proof. (Sketch) There are $z = \sum_j n_j$ decision points to bid on in each run of the mechanism. The sequence of decisions can be described by a z-ary vector of job agent indices. The set of all sequences that lead to active schedules can be obtained through iterated 0-bidding (no agent bids a non-zero amount of currency units in any iteration). This is due to the non-deterministic tie-breaking and the fact that the mechanism follows the execution path of GT and thus inherits the property of GT to generate all active schedules if all decisions are taken in every possible way. The proposition follows from an adaptation of Theorem 2.1 of (French 1982) (suggested by French in Chap. 10), which states that the class of active schedules contains the optimal schedule for regular measures, and the fact that the restricted EJSP mimics a regular measure. □

Note that each bidding decision is necessarily based on calculations of the expected value of a positive decision (the agent wins the decision point). Each agent knows beforehand that he has to win exactly n_j decision points. He can also calculate the influence of each taken decision on the maximally achievable utility (simply by assuming that he will win all following decision points up to and including an allocation for his last operation, which fixes the reservation times for all remaining operations and, thus, allows us to set a precise upper bound on the realizable utility). His attitude towards risk will influence his bidding decisions, as will his knowledge about the bidding behavior of competing agents in this or in prior runs of the mechanism (if agents participate in multiple allocation problems).[22] A solution concept to study such situations is known as Bayes-Nash Equilibrium, though this is not discussed here. Let us only point out that augmenting solution procedures that show a natural fit to the problem domain may be an interesting opportunity to design economic coordination mechanism that are easily understandable for the participating agents and that give the agents a direct handle to use their understanding to influence the outcome of the mechanism. The decision point approach can be applied to numerous, decision-based solution procedures. Note that obtaining analytical results will not be an easy task, however, advancing this concept looks promising to us due to the similitude of mechanism and domain-specific problem structure.

[21] Instead of the sealed-bid second-price auction suggested here, an open-cry English auction could be chosen, which would convey more information about the competitors to the agents.

[22] The mechanism can be extended to resale to take limited foresight into account.

5 Discussion

This paper has discussed economically augmented job shop problems and their transformation into a specific subclass of combinatorial auction problems. We have briefly discussed the problem of finding efficient solutions and of doing this in a way to ensure that the agents participate and report their valuations truthfully. Motivated by problems of computational efficiency, we have suggested to combine domain-related decision making procedures with bidding. This hybrid approach results in a heuristic that, while remaining problem driven, is applicable in the context of self-interested agents with private information. We consider the study of such application scenarios in the context of scheduling for manufacturing and logistics as important due to the increasing tendencies to (1) restructure companies towards collaborations of (semi-)autonomous units on all levels of granularity and to (2) form (potentially) volatile collaborations between autonomous enterprises. In both cases, monetarian value may turn out to be a key issue for the integration of diverging interests—economically efficient mechanisms that keep the participating agents individually satisfied can contribute to this integration without violating autonomity and information privacy more than necessary.

5.1 Extensions and future work

The mapping of economic job shop problems to combinatorial auction problems can easily be extended to accommodate other classes of job shop problems, for example problems with alternative routings, reservation costs for resources, or sequence-dependent set-up costs.

Further effort is necessary to explore the options and consequences of decision point bidding, with respect to both a theoretical analysis of its effect on efficiency, strategic behavior etc. and a practical or experimental analysis of its applicability. It will also be interesting to apply the basic concept of decision point bidding to other solution procedures for resource allocation problems.

References

Adelsberger, H. H., and Conen, W. 2000. Economic coordination mechanisms for holonic multi agent systems. In *Proc. of HoloMAS Workshop, DEXA 00*.

Adelsberger, H. H.; Conen, W.; and Krukis, R. 1995. Scheduling utilizing market models. In *Proc. of the ICCIM 95*. Singapore: World Scientific.

Ausiello, G.; Crescenzi, P.; Gambosi, G.; Kann, V.; Marchetti-Spaccamela, A.; and Protasi, M. 1999. *Complexity and Approximation*. Springer.

Baker, A. 1996. A case study where agents bid with actual costs to schedule a factory. In Clearwater, S. H., ed., *Market-Based Control*. Singapore: World Scientific. 184–223.

Bikhchandani, S.; de Vries, S.; Schummer, J.; and Vohra, R. V. 2001. Linear programming and Vickrey auctions.

Conen, W., and Sandholm, T. 2001a. Preference elicitation in combinatorial auctions. In *Proc. of the ACM E-commerce conference (ACM-EC01)*. Tampa, FL: ACM. Short Paper.

Conen, W., and Sandholm, T. 2001b. Minimal preference elicitation in combinatorial auctions. In *IJCAI-2001 WS on Economic Agents, Models, and Mechanisms*, 71–80.

Conen, W., and Sandholm, T. 2002a. Coherent pricing of efficient allocations in combinatorial economies. In *Prooceddings of the AAAI-02 Workshop on Game theoretic and Decisiosn Theoretic Agents (GTDT-02)*.

Conen, W., and Sandholm, T. 2002b. Partial-revelation VCG mechanism for combinatorial auctions. In *Proceddings of the National Conference on Artificial Intelligence (AAAI)*.

de Vries, S., and Vohra, R. 2001. Combinatorial auctions: A survey. Draft of 25. Oktober 2001.

French, S. 1982. *Sequencing and Scheduling*. Ellis Horwood Limited.

Fujishima, Y.; Leyton-Brown, K.; and Shoham, Y. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 548–553.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability*. W. H. Freeman and Company.

Gul, F., and Stacchetti, E. 1999. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory* 87:95–124.

Hoogeveen, H.; Schuurman, P.; and Woeginger, G. 1998. Non-approximability results for scheduling problems with minsum criteria. In Bixby, R. E.; Boyd, E. A.; and Rios-Mecado, R. Z., eds., *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization,*, number 1412 in LNCS, 353–366. Springer.

Kelso, A. S., and Crawford, V. 1982. Job matching, coalition formation, and gross substitues. *Econometrica* 50:1483–1504.

Lawler, E.; Lenstra, J.; Kan, A. R.; ; and Shmoys, D. 1992. *Sequencing and Scheduling: Algorithms and Complexity*, volume 4 of *Handbooks in Operations Research and Management Science*. North-Holland.

Li, M., and Vitanyi, P. 1997. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 2nd edition edition.

Mas-Colell, A.; Whinston, M.; and Green, J. R. 1995. *Microeconomic Theory*. Oxford University Press.

Nisan, N., and Segal, I. 2002. The communication complexity of efficient allocation problems. Working paper.

Parkes, D., and Ungar, L. 2000. Iterative combinatorial auctions: Theory and practice. In *Proc. of the AAAI-00*.

Parkes, D. C. 2001. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. Ph.D. Dissertation, Computer and Information Sciences, University of Pennsylvania.

Parunak, H. V. D. 1996. Applications of distributed artificial intelligence in industry. In O'Hare, G., and Jennings, N., eds., *Foundations of Distributed Artificial Intelligence*. New York: Wiley. 139–164.

Rothkopf, M. H.; Pekeč, A.; and Harstad, R. M. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147. Early version: Rutgers Center for Operations Research technical report 13-95.

Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2001. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1102–1108.

Sandholm, T. 1996. *Negotiation among Self-Interested Computationally Limited Agents*. Ph.D. Dissertation, University of Massachusetts, Amherst. Available at http:// www.cs.cmu.edu/ ˜sandholm/ dissertation.ps.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135:1–54.

Vickrey, W. 1961. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance* 16:8–37.

Walsh, W., and Wellman, M. 1998. A market protocol for decentralized task allocation. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS)*, 325–332.

Walsh, W.; Wellman, M.; and Ygge, F. 2000. Combinatorial auctions for supply chain formation. In *ACM Conference on Electronic Commerce 2000: 260-269*.

Wassenhove, L. V., and Gelders, L. 1980. Solving a bicriterion scheduling problem. *Eur.J. Opl. Res.* 4:42–48.

Wellman, M.; Walsh, W.; Wurman, P.; and MacKie-Mason, J. 2001. Auction protocols for decentralized scheduling. *Games and Economic Behavior* 35:271–303.

Wurman, P. R., and Wellman, M. P. 2000. AkBA: A progressive, anonymous-price combinatorial auction. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, 21–29.

Wurman, P. R. 1999. *Market Structure and Multidimensional Design Auction for Computational Economies*. Ph.D. Dissertation, Computer Science and Engineering, University of Michigan.

Taming Numbers and Durations in the Model Checking Integrated Planning System

Stefan Edelkamp
Institut für Informatik, Georges-Köhler-Allee, Gebäude 51,
Albert-Ludwigs-Universität, 79110 Freiburg, Germany
eMail: edekamp@informatik.uni-freiburg.de

September 20, 2002

Abstract

The Model Checking Integrated Planning System (MIPS) has shown distinguished performance in the second and third international planning competitions. With its object-oriented framework architecture MIPS clearly separates the portfolio of explicit and symbolic heuristic search exploration algorithms from different on-line and off-line computed estimates and from the grounded planning problem representation.

In 2002, the domain description language for the benchmark problems has been extended from pure propositional planning to include rational state resources, action durations, and plan quality objective functions. MIPS has been the only system that produced plans in each track of every benchmark domain. This article presents and analyzes the algorithmic novelties necessary to tackle the new layers of expressiveness.

The planner extensions include critical path analysis of sequentially generated plans to generate optimal parallel plans. The linear time algorithm bypasses known NP hardness results for partial ordering with mutual exclusion by scheduling plans with respect to the set of actions *and* the imposed causal structure. To improve exploration guidance approximate plans are scheduled for each encountered planning state.

One major strength of MIPS is its static analysis phase that grounds and simplifies parameterized predicates, functions and operators, that infers single-valued invariances to minimize the state description length, and that detects symmetries of domain objects. The aspect of object symmetry is analyzed in detail.

The paper shows how temporal plans of any planner can be visualized in Gannt-chart format in a client-server architecture. The frontend turns also be appropriate for concise domain visualization.

Contents

1 Introduction

The Model Checking Integrated Planning System MIPS has participated twice in the international planning competition: in the second planning competition at AIPS-2000 in Beckenridge (USA) and in the third planning competition at AIPS-2002 in Toulouse (France). As the name indicates, the MIPS project targets the integration of model checking techniques into a domain-independent action planner.

Model checking (Clarke, Grumberg, & Peled, 1999) is the automated process to verify if a formal model of a system satisfies an specified temporal property or not. As an illustrative example, take an elevator control system together with a correctness property that requires an elevator to eventually stop on every call of a passenger or that guarantees that the door is closed, while the elevator is moving.

Although the success in checking correctness is limited, model checkers found many subtle errors in current hardware and software designs. Models often consists of many concurrent subsystems. Their combination is either synchronous, as often met in hardware design verification, or asynchronous, as frequently given in communication and security protocols, or in multi-threaded programming languages like Java.

Exploration of model checking domains spans very large spaces of all reachable system states. This effect is usually denoted as the *state explosion problem*, even if the sets of generated states rather than the states themselves grow that quickly.

An error that shows a safety property violation, like a deadlock or a failed assertion, corresponds to one of a set of target nodes in the state space graph. Roughly speaking, *something bad has occured*. A liveness property violation refers to a (seeded) cycle in the graph. Roughly speaking, *something good will never occur*. For the case of the elevator example, eventually reaching a target state where a request button was pressed is a liveness property, while certifying closed doors refers to a safety property.

In this paper we refer to safety properties only, since goal achievement in traditional and competition planning problems have yet not been extended with temporal properties. However, temporally extended goals are of increasing research interests (Kabanza, Barbeau, & St-Denis, 1997; Pistore & Traverso, 2001; Lago, Pistore, & Traverso, 2002).

The two main validation processes in model checking are explicit and symbolic search. In explicit-state model checking each state refers to a fixed memory location and the state space graph is implicitly generated by successive expansions of state.

In symbolic model checking (McMillan, 1993; Clarke, McMillan, Dill, & Hwang, 1992), fixed-length binary encodings of states are usually seen as mandatory, so that each state can be represented by its characteristic Boolean function. The function evaluates to true if and only if all state variables are assigned to according bit values. Sets of states are expressed by the disjunct of the individual characteristic functions. On the other hand satisfiability and uniqueness of Boolean formulae is NP hard.

The unique symbolic representation of sets of states as Boolean formulae through binary decision diagrams (BDDs) (Bryant, 1992) is often much smaller than the explicit one. BDDs are (ordered) read-once branching programs with nodes corresponding to variables, edges corresponding to variable outcomes, and each path corresponding to an assignment to the variables with the resulting evaluation at the leaves. One reason of the succinctness of BDDs is that directed acyclic graphs may express exponentially many paths. Since states are encoded in binary, the transition relation is defined on two state variable sets. It evaluates to true, if and only if an operator exists that transforms a state into a valid successor. In some sense, BDDs exploit regularities of the state set and often appear better suited to regular hardware systems, in contrast to many software system that inherit a highly asynchronous and irregular structure, so that the straight use BDD with their fixed variable ordering is probably not flexible enough.

For symbolic exploration a set of states is combined with the transition relation to compute the set of all possible successor states, i.e. the image. Starting with the initial state, iteration

of image computations eventually explores the entire reachable state space. To improve the efficiency of image computations, transition relations are often provided in partitioned form.

The correspondence of action planning and model checking can be roughly characterized as follows. Similar to model checkers, action planners implicitly generate large state spaces, and both exploration approaches base on applying parameterized operators to the current state. States in model checking and in planning problems are both labeled by (propositional state) predicates. The satisfaction of a specified property on the one side and the arrival at a certain goal state on the other, leads to a slight difference in the according search objective. With this respect, the goal in action planning is a safety error and the corresponding (error) trail is interpreted as a plan. In the elevator example, the goal of a planning task is to reach a state, in which the doors are open and the elevator is moving. For a formal treatment on the embedding of planning problems into model checking terminology, we refer the reader to (Giunchiglia & Traverso, 1999).

Model checkers perform either symbolic or explicit exploration. To the contrary MIPS features both and allows to combines symbolic and explicit search planning. It applies heuristic search; a search acceleration technique that has let to considerable gains in both communities. In the last few years, heuristic search planners frequently outperform other domain-independent planning approaches, e.g. (Hoffmann & Nebel, 2001), and heuristic search model checkers turn out to significantly improve state-of-the-art, e.g. (Edelkamp, Leue, & Lluch-Lafuente, 2002).

Including resource variables (like the fuel level of a vehicle or the distance between locations) and action duration (i.e. the time passed during execution of the planning operator) are relatively new aspects for action planning, at least in form of an accepted domain description accessible for competitive planning (Fox & Long, 2001). The competition input format PDDL2.1 is not restricted to variables of finite domain, but also includes specification of rational (floating-point) variables in both precondition and effects. Similar to a set of atoms described by a propositional predicate, a set of numerical quantities can be described by a set of parameters. Through the notation of PDDL2.1, we refer to parameterized numerical quantities as functions. For example, the fuel level might be parameterized by the vehicle that is present in the problem instance description.

In the 2002 competition, domains were provided in different tracks according to different layers of language expressiveness: *i*) pure propositional planning, *ii*) planning with numerical resources, *iii*) planning with numerical resources and constant action duration, *iv*) planning with numerical resources and variable action duration, and, in some cases, *v*) more complex problems usually combining time and numbers in more interesting ways. MIPS competed as a fully automated system and performed remarkably well in all five tracks; it solved a high number of problems and was the only system that produced solutions in each track of every benchmark domain.

In this paper the main algorithmic aspects to *tame* rational numbers, objective functions, and action duration are described. The article is structured as follows. First, we recall the development of the MIPS system and assert its main contributions to the planning community. Then we address the object-oriented heuristic search framework architecture of the system. Subsequently, we fix some terminology that allows to give a formal definition of the syntax and the semantics of a grounded mixed numerical and propositional planning problem instance.

We then introduce the core contributions: critical path scheduling for concurrent plans, and efficient methods for detecting and using symmetry cuts. PERT scheduling produces optimal parallel plans given a sequence of operators and a precedence relation among them in linear time. The paper discusses pruning anomalies and handling of different optimization criteria. We analyze the correctness and efficiency of symmetry detection in detail. Afterwards, a TCP/IP client-server visualization system for sequential and temporal plans is presented. The article closes with related work and concluding remarks.

2 The Development of MIPS

The competing versions of MIPS refer to initial findings (Edelkamp & Reffel, 1999) of heuristic symbolic exploration of planning domains with the μcke model checker (Biere, 1997) that already lead to good performance in puzzle solving (Edelkamp & Reffel, 1998) and in hardware verification (Reffel & Edelkamp, 1999). For general propositional planning, our concise BDD library *StaticBdd*[1] has been used.

During the implementation process we changed the BDD representation to improve performance mainly for small planning examples and chose the public domain `c++` BDD package Buddy (Lind-Nielsen, 1999). In the beginning of the project the variable encodings were provided by hand, while the representation of all possible operator descriptions were established by enumerating all possible parameter instances. Once the encoding and transition relation were fixed, symbolic exploration in form of a reachability analysis of the state-space could be executed. At that time, we were not aware of any other work in BDD-based planning like (Cimatti, Giunchiglia, Giunchiglia, & Traverso, 1997), which is probably the first link to planning via (symbolic) model checking.

Since the above approach was criticized not to be fully automated, we subsequently developed a parser and a static analyzer to cluster atoms into groups in order to minimize the length of the state encoding (Edelkamp & Helmert, 1999). The outcome of the analyzer allowed to specify states and transition functions in Boolean terms, which in turn were included in a bidirectional BDD exploration and solution extraction procedure. In the end, MIPS was the first automated planning system based on symbolic model checking.

In the second international planning competition MIPS (Edelkamp & Helmert, 2001) could handle the STRIPS (Fikes & Nilsson, 1971) subset of the PDDL language (McDermott, 2000) and some additional features from ADL (Pednould, 1989), namely negative preconditions and (universal) conditional effects. MIPS was one of five planning systems to be awarded for "Distinguished Performance" in the fully automated track. The competition version (Edelkamp & Helmert, 2000) already included explicit heuristic search algorithms based on a bit-vector state representation and the relaxed planning heuristic (RPH) (Hoffmann & Nebel, 2001) and symbolic heuristic search based on the HSP-Heuristic (Bonet & Geffner, 2001) and a one-to-one atom RPH-derivate. However, at the end we used breadth-first bi-directional symbolic search in each case the single state heuristic searcher got stuck in its exploration.

In between the planning competitions, explicit (Edelkamp, 2001c) and symbolic pattern databases (Edelkamp, 2002b) were proposed as off-line generated estimators referring to completely explored problem abstractions. Roughly speaking, pattern database abstractions slice the state vector of fluent facts into pieces and adjusts the operators accordingly. The completly explored subspaces then serve as admissible estimate for the overall search and are competetive with the relaxed planning heuristic.

For the 2002's international planning competition new levels of the planning domain description language (Fox & Long, 2001) have been designed to specify problems that include actions with durations and resources. The agreed input language definition is referred to as PDDL 2.1. While Level 1 considers pure propositional planning, Level 2 also includes numerical resources and objective functions to be minimized, and Level 3 additionally allows to specify actions with durations. Consequently, MIPS[2] has been extended to cope with these new forms of expressiveness.

In (Edelkamp, 2001b) first results of MIPS in planning PDDL 2.1 problems are presented. The preliminary treatment exemplifies the parsing process in two simple benchmark domains. Moreover, propositional heuristics and manual branching cuts were applied to accelerate sequential plan generation. This work was extended in (Edelkamp, 2002a), where two approximate exploration techniques to bound and to fix numerical domains, first results on symmetry detec-

[1] See `http://www.informatik.uni-freiburg.de/~edelkamp/StaticBdd`

[2] A recent version of MIPS is available in source code at `www.informatik.uni-freiburg.de/~edelkamp`

tion based on fact groups and critical path scheduling, an any-time wrapper to produce optimal plans and a numerical extension to RPH were presented. Enumerating variable domains and the any-time wrapper were excluded from the competition version of MIPS because of their unpredictable impact on the planner performance.

Our approach to extend RPH with numerical information establishes plans even in challenging numerical domains like *Settlers* and was developed independently from Hoffmann's work on his competing planner *Metric-FF*. Since his planner appears to be more general in its parsing process to generate monotone numerical quantities for the relaxation, we omitted the algorithmic issues for this aspect from this manuscript. The reader is refered to (Hoffmann, 2002a) for further information on this important issue of relaxed plan generation.

Although possible and plausible, in the competition, (S)PDBs estimates were finally not included for plan generation in MIPS, since the integration of numerical state facets and the retrieval of the corresponding relaxed plan operators had not been finished. Hence, the applied heuristic search engine at least in the competition version of MIPS relates to a numerical relaxed plan generator with important pre- and postprocessing aspects.

Hence, we selected the main contributions of this paper to include the following aspects:

- the formal definition of grounded propositional and numerical planning and an index scheme for grounding predicate, functions, and actions;
- the object-oriented framework architecture to choose and combine different heuristics with different search algorithms and storage structures;
- the static analyzer that applies efficient fact-space exploration to distinguish constant from variable atoms and resource variables, that clusters facts into groups and that infers static object symmetries;
- different pruning methods, especially dynamic symmetry detection, hash and transposition cuts, and different strategies for optimizing objective functions and further implementation tricks that made the system efficient;
- a throughout study of dynamic object symmetries, their time and space complexities and a possible trade-off as implemented in MIPS;
- optimal temporal planning enumeration algorithms based on a precedence ralation and PERT scheduling of sequentially generated plans together with a concise analysis of correctness and optimality;
- the integration of PERT scheduling already in the heuristic estimate to guide the search favoring states with smaller parallel plan length;
- the intermediate format of grounded and simplified planning domain instances to serve as an interface for other planners;
- a client-server system for visualization including a wrapper for temporal plans to be presented in Gannt-chart format, and a domain-dependent frontend for executing sequential plans.

3 Terminology

Our running example is the following instance to of a simple PDDL 2.1 problem in *Zeno-Travel* and illustrated in Figure 1. The initial configuration is drawn to the left of the figure and the goal configuration to its right. Some global and local numerical variable assignment are not shown.

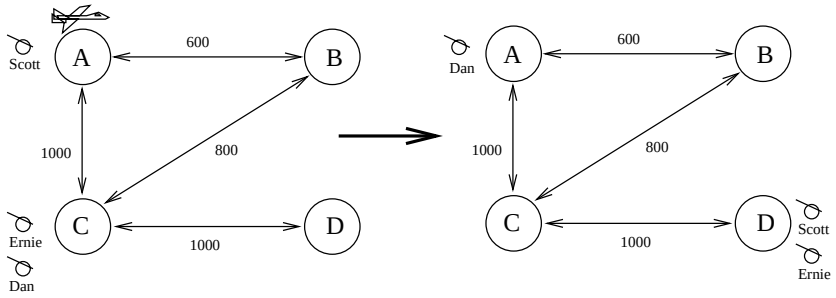

Figure 1: An Instance to the *Zeno-Travel* Domain: Start State (left) and Goal State (right).

In Figures 2 and 3 we provide the according textual domain and problem specification[3]. The instance asks for a temporal plan to fly passengers (`dan`, `scott`, and `ernie`) located somewhere on a small map (including the cities `city-a`, `city-b`, `city-c`, and `city-d`) with an aircraft (`plane`) to their respective target destinations. Boarding and debarking takes a constant amount of time. The plane has a determined capacity of fuel. Fuel and time are consumed according to the distances between the cities and with respect to two different travel speeds. Since fuel can be restored by refueling the aircraft, the total amount of fuel is also maintained as a numerical quantity.

3.1 Sets and Indices

Table 1 displays the basic terminology for sets used in this paper. As most currently successful planning system, MIPS grounds parameterized information present in the domain description.

Set	Descriptor	Example(s)
$\mathcal{OBJ}$	objects	`dan`, `city-a`, `plane`, ...
$\mathcal{TYPE}$	object types	`aircraft`, `person`, ...
$\mathcal{PRED}$	predicates	`(at ?a ?c)`, `(in ?p ?a)`, ...
$\mathcal{FUNC}$	numerical functions	`(fuel ?a)`, `(total-time)`, ...
$\mathcal{ACT}$	parameterized actions	`(board ?a ?p)`, `(refuel ?a)`, ...
$\mathcal{IACT}$	instantiated actions	`(board plane scott)`, ...
$\mathcal{O} \subseteq \mathcal{IACT}$	fluent operators	`(board plane scott)`, ...
$\mathcal{IPRED}$	instantiated predicates	`(at plane city-b)`, ...
$\mathcal{F} \subseteq \mathcal{IPRED}$	fluents	`(at plane city-b)`, ...
$\mathcal{IFUNC}$	instantiated funtions	`(distance city-a city-b)`, ...
$\mathcal{V} \subseteq \mathcal{IFUNC}$	variables	`(fuel plane)`, `(total-time)`, ...

Table 1: Basic Set Definitions.

For all sets we infer a suitable array embedding, indicated by a mapping ϕ from this set to a finite domain and vice versa. This embedding is important to deal with unique identifiers of entities instead of their textual or internal representation. The arrays containing the corresponding information can then be accessed in constant time. Almost all planners that perform grounding prior to the search address instantiations by identifiers.

For sets that occur in the domain or problem specification without any parameterization like $\mathcal{CONST}, \mathcal{PRED}, \mathcal{FUNC}, \mathcal{ACT}, \mathcal{TYPE}, \mathcal{ACT}$, and $\mathcal{OBJ}$, the index ϕ refers to the position of

[3] `[...]` denotes that source fragments were omitted for the sake of brevity. In the given example these are the action definitions for debarking a passenger and flying an airplane..

```
(define (domain zeno-travel)
(:requirements :durative-actions :typing :fluents)
(:types aircraft person city)
(:predicates (at ?x - (either person aircraft) ?c - city)
             (in ?p - person ?a - aircraft))
(:functions (fuel ?a - aircraft) (distance ?c1 - city ?c2 - city)
            (slow-speed ?a - aircraft) (fast-speed ?a - aircraft)
            (slow-burn ?a - aircraft)  (fast-burn ?a - aircraft)
            (capacity ?a - aircraft)   (refuel-rate ?a - aircraft)
            (total-fuel-used) (boarding-time) (debarking-time))
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration boarding-time)
 :condition (and (at start (at ?p ?c))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at end (in ?p ?a))))
[...]
(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
 :condition (and (at start (at ?a ?c1))
                 (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (increase total-fuel-used
                        (* (distance ?c1 ?c2) (fast-burn ?a))))
              (at end (decrease (fuel ?a)
                        (* (distance ?c1 ?c2) (fast-burn ?a))))))
(:durative-action refuel
 :parameters (?a - aircraft ?c - city)
 :duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a)))
 :condition (and (at start (< (fuel ?a) (capacity ?a)))
                 (over all (at ?a ?c)))
 :effect (at end (assign (fuel ?a) (capacity ?a))))
)
```

Figure 2: Zeno-Travel Domain Description in PDDL2.1.

occurrence. Let $k(p)$, $k(f)$, and $k(a)$ denote the arity (the number of parameters) of predicate $p \in \mathcal{PRED}$, function $f \in \mathcal{FUNC}$, and $a \in \mathcal{ACT}$, respectively. The index for an instantiated predicate $(p\ o_1 \ldots o_{k(p)}) \in \mathcal{IPRED}$ is computed as

$$\phi((p\ o_1 \ldots o_{k(p)})) = \psi(p) + \sum_{i=1}^{k(p)} \phi(o_i)|\mathcal{OBJ}|^{i-1},$$

where $\psi(p) = \sum_{i=1}^{\phi(p)-1} |\mathcal{OBJ}|^{k(p_i)}$ is the offset of predicate $p \in \mathcal{PRED}$ and $|\mathcal{OBJ}|$ is the cardinality of set $\mathcal{OBJ}$. Taking $|\mathcal{OBJ}|$ as the radix is a rather coarse value for all parameter instantiations; one could refine the index by using parameter type information.

Indices for instantiated functions $(f\ o_1 \ldots o_{k(f)}) \in \mathcal{IFUNC}$ are determined analogously. Instantiated actions $a \in \mathcal{IACT}$ with parameters $p_1, \ldots, p_{k(a)}$ are consequently addressed by the following index

```
(define (problem zeno-travel-1)
    (:domain zeno-travel)
    (:objects plane - aircraft
              ernie scott dan - person
              city-a city-b city-c city-d - city)
    (:init (= total-fuel-used 0) (= debarking-time 20) (= boarding-time 30)
           (= (distance city-a city-b) 600)  (= (distance city-b city-a) 600)
           (= (distance city-b city-c) 800)  (= (distance city-c city-b) 800)
           (= (distance city-a city-c) 1000) (= (distance city-c city-a) 1000)
           (= (distance city-c city-d) 1000) (= (distance city-d city-c) 1000)
           (= (fast-speed plane) (/ 600 60)) (= (slow-speed plane) (/ 400 60))
           (= (fuel plane) 750)              (= (capacity plane) 750)
           (= (fast-burn plane) (/ 1 2))     (= (slow-burn plane) (/ 1 3))
           (= (refuel-rate plane) (/ 750 60))
           (at plane city-a) (at scott city-a) (at dan city-c) (at ernie city-c))
    (:goal (and (at dan city-a) (at ernie city-d) (at scott city-d)))
    (:metric minimize total-time)
)
```

Figure 3: Zeno-Travel Problem Instance.

$$\phi((a\ p_1 \dots p_{k(a)})) = \psi(a) + \sum_{i=1}^{k(a)} \phi(p_i)|\mathcal{OBJ}|^{i-1}.$$

After static analysis has established a superset of all occurring fluents $\mathcal{F}$, operators $\mathcal{O}$ and variables $\mathcal{V}$, in MIPS the index range is reduced to a minimum, thereby refining ϕ to ϕ'. In the following we keep ϕ as a descriptor, and assume that $\phi(p) \in \{1, \dots, |\mathcal{P}|\}$, $\phi(f) \in \{1, \dots, |\mathcal{V}|\}$, and $\phi(a) \in \{1, \dots, |\mathcal{O}|\}$.

In the following we first give the formal description of a grounded planning problem and then turn to the static analyzer that infers the according and supplementary information.

3.2 Grounded Planning Problem Instances

As many other planners MIPS refers to grounded planning problem representations.

Definition 1 *(Grounded Planning Instance) A* grounded planning instance *is a quadruple* $\mathcal{P} = \langle \mathcal{S}, \mathcal{I}, \mathcal{O}, \mathcal{G} \rangle$*, where* $\mathcal{S}$ *is the set of planning states,* $\mathcal{I} \in \mathcal{S}$ *is the initial state,* $\mathcal{G} \subseteq \mathcal{S}$ *is the set of goal states. In mixed propositional and numerical planning problem the state space* $\mathcal{S}$ *is given by*

$$\mathcal{S} \subseteq 2^{\mathcal{F}} \times I\!R^{|\mathcal{V}|},$$

where $2^{\mathcal{F}}$ *is the power set of* $\mathcal{F}$*. Therefore, a state* $S \in \mathcal{S}$ *is a pair* (S_p, S_n) *with propositional part* $S_p \in 2^{\mathcal{F}}$ *and numerical part* $S_n \in I\!R^{|\mathcal{V}|}$*.*

For the sake of brevity, we assume the operators to be in *normal form*, by means that propositional parts (preconditions and effects) satisfy standard STRIPS notation (Fikes & Nilsson, 1971) and numerical parts are given in form of arithmetic trees t taken from the set of all trees T with arithmetic operations in the nodes and numerical variables and evaluated constants in the leaves. With $\mathit{LeafVariables}(t)$, $t \in T$, we denote the set of all leaf variables in the tree t. However, there is no fundamental difference to more general preconditions and effects representations. The current implementation in MIPS takes a generic precondition tree, thereby including comparison symbols, logical operators (in the nodes) and arithmetic subtrees.

Definition 2 *(Syntax of Grounded Planning Operator) An operator $o \in \mathcal{O}$ in normal form $o = (\alpha, \beta, \gamma, \delta)$ has propositional preconditions $\alpha \subseteq \mathcal{F}$, propositional effects $\beta = (\beta_a, \beta_d) \subseteq \mathcal{F}^2$, numerical preconditions γ, and numerical effects δ. A* numerical precondition *$c \in \gamma$ is a triple $c = (h_c, \otimes, t_c)$, where $h_c \in \mathcal{V}$, $\otimes \in \{\leq, <, =, >, \geq\}$, and $t_c \in T$. A* numerical effect *$m \in \delta$ is a triple $m = (h_m, \oplus, t_m)$, where $h_m \in \mathcal{V}$, $\oplus \in \{\leftarrow, \uparrow, \downarrow\}$ and $t_m \in T$.*

Obviously, $\otimes \in \{\leq, <, =, >, \geq\}$ represents the associated comparison relation, while $\leftarrow$ denotes an assigment to a variable, while $\uparrow$ and $\downarrow$ indicate a respective increase or decrease operation to it. This allows to formalize the application of planning operators to a given state.

Definition 3 *(Semantics of Grounded Planning Operator Application) An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ applied to a state $S = (S_p, S_n)$, $S_p \in 2^{\mathcal{F}}$ and $S_n \in \mathbb{R}^{|\mathcal{V}|}$, yields a successor state $S' = (S'_p, S'_n) \in 2^{\mathcal{F}} \times \mathbb{R}^{|\mathcal{V}|}$ as follows.*

We say that a vector $S_n = (S_1, \ldots, S_{|\mathcal{V}|})$ of numerical variables satisfies *a numerical constraint $c = (h_c, \otimes, t_c) \in \gamma$ if $s_{\phi(h_c)} \otimes eval(S_n, t_c)$ is true, where $eval(S_n, t_c) \in \mathbb{R}$ is obtained by substituting all $v \in \mathcal{V}$ in t_c by $S_{\phi(h_c)}$ followed by a simplification of t_c.*

If $\alpha \subseteq S_p$ and S_n satisfies *all $c \in \gamma$ then $S'_p = S_p \cup \beta_a \setminus \beta_d$ and the vector S_n is* updated *for all $m \in \delta$. We say that the vector $S_n = (S_1, \ldots, S_{|\mathcal{V}|})$ is* updated *to vector $S'_n = (S'_1, \ldots, S'_{|\mathcal{V}|})$ by modifier $m = (h_m, \oplus, t_m) \in \delta$, if*

- $S'_{\phi(h_m)} = eval(S_n, t_m)$ *for* $\oplus = \leftarrow$,
- $S'_{\phi(h_m)} = S_{\phi(h_m)} + eval(S_n, t_m)$ *for* $\oplus = \uparrow$, *and*
- $S'_{\phi(h_m)} = S_{\phi(h_m)} - eval(S_n, t_m)$ *for* $\oplus = \downarrow$.

The propositional update $S'_p = S_p \cup \beta_a \setminus \beta_d$ is defined as in standard STRIPS. The set of goal states $\mathcal{G}$ is often given as $\mathcal{G} = (\mathcal{G}_p, \mathcal{G}_n)$ with a partial propositional state descrition $\mathcal{G}_p \subset \mathcal{F}$, and G_n as a set of numererical preconditions $c = (h_c, \otimes, t_c)$. Moreover, the arithmetic trees t_c usually collaps to simple leaves labeled with numerical constants. Hence, we might assume that $|\mathcal{G}_n| \leq |\mathcal{V}|$.

3.3 Static Analysis

The static analyzer takes the domain and problem instance as an input, grounds its propositional state information and infers different forms of planner independent static information.

- Parsing: Our simple Lisp parser generates a tree of Lisp entities. It reads the input files and recognizes the domain and problem name. To cope with typing we temporarily assert constant typed predicates to be removed together with other constant predicates in a further pre-compiling step. Thereby, we infer a type hierarchy and an associated mapping of objects to types.
- Indexing: Based on the number of counted objects, first indices for the grounded predicates, functions and actions are devised. Since in our example problem we have eight objects and the predicates `at` and `in` have two parameters, we reserve $2 \cdot 8 \cdot 8 = 128$ index positions. Similarly, the function `distance` consumes 64 indices, while `fuel`, `slow-speed`, `fast-speed`, `slow-burn`, `fast-burn`, `capacity`, and `refuel-rate` each reserve eight index positions. For the quantities `total-fuel-used`, `boarding-time`, `debarking-time` only a single fact identifier is needed. Last but not least we interpret duration as an additional quantity `total-time`.
- Flattening Temporal Identifiers: According to our assumption of finite branching in this phase we interpret each action as in integral entity, so that all timed propositional and

numerical preconditions can be merged. Similarly, all effects are merged, independent of their happening. Invariance conditions like `(over all (at ?a ?c))` in the action `board` are included into the precondition set. We will discuss the rationale of this step in Section 5.

- Grounding Propositions: *Fact-space exploration* is a relaxed enumeration of the planning problem to determine a superset of all reachable facts. Algorithmically, a FIFO fact queue is comprised. Successively extracted facts at the front of the queue are matched to the operators. Each time all preconditions of an operator are fulfilled, the resulting atoms according to the positive effect (add) list are determined and enqueued. This allows to distinguish constant from fluent facts, since only the latter are reached by exploration.

- Grouping Atoms: For a concise encoding of the propositional part we group fluent facts in sets of mutually exclusive groups, so that each state in the planning space can be expressed as a conjunct of (possibly trivial) facts drawn from each fact group (Edelkamp & Helmert, 1999). More formally, let $\#p_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n)$ be the number of objects o_i for which the fact $(p\ o_1\ \ldots\ o_n)$ is true. We establish a single-valued invariance at i if $\#p_i(o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n) = 1$. All fix object o_j, $j \neq i$, are representative of the invariance and label the group. To allow for a better encoding, some predicates like `at` and `in` are merged. In the example three groups determine the unique position of the persons (one of five) and one group determines the position of the plane (one of four). Therefore, $3 \cdot \lceil \log 5 \rceil + 1 \cdot \lceil \log 4 \rceil = 11$ bits suffice to encode the encountered 19 fluent facts.

- Grounding Actions: Fact-space exploration also determines all grounded operators. Once all preconditions are met and grounded, the symbolic effect lists are instantiated. In our case we determine 98 instantiated operators, which, by some further simplifications that eliminate duplicates and void operators, are reduced to 43.

- Grounding Functions: Synchronous to fact space exploration of the propositional part of the problem all heads of the numerical formulae in the effect lists are grounded. In the example case only three instantiated formulae are fluent: `fuel plane` with initial value 750 as well as `total-fuel-used` and `total-time` both initialized with zero. All other numerical predicates are in fact constants that can be substituted in the formula-bodies. For example, the numerical effect in `board dan city-a` reduces to `(increase (total-time) 30)`, while `zoom plane city-a city-b` has the following numerical effects: `(increase (total-time) 150)`, `(increase (total-fuel-used) 300))`, and `(decrease (fuel plane) 300)`. Refueling, however, does not reduce to a single rational number, e.g. the effects in `refuel plane city-a` only simplify to `(increase (total-time) (/ (- (750 (fuel plane)) / 12.5)))` and `(assign (fuel plane) 750)`. To evaluate the former assignment variable `total-time` has to be instantiated *on-the-fly*. This is due to the fact that the value of the quantity `fuel plane` is not constant and itself changes over time.

4 Architecture of MIPS

Figure 4 depicts the main components of MIPS and the data flow from the input definition of the domain and the problem instance to the resulting temporal plan in the output.

The planning process can be coarsely grouped into two stages, static analysis and (heuristic search) planning.

The intermediate textual format of the static analyzer in annotated grounded PDDL-like representation serves as an interface e.g. for other planners or model checkers and as an additional resource for plan visualization. Figures 5 and 6 depict an example output for the intermediate representation in the *Zeno-Travel* example.

The object-oriented framework design of MIPS allows different heuristic estimates to be combined with different search strategies, access data structures, and scheduling options.

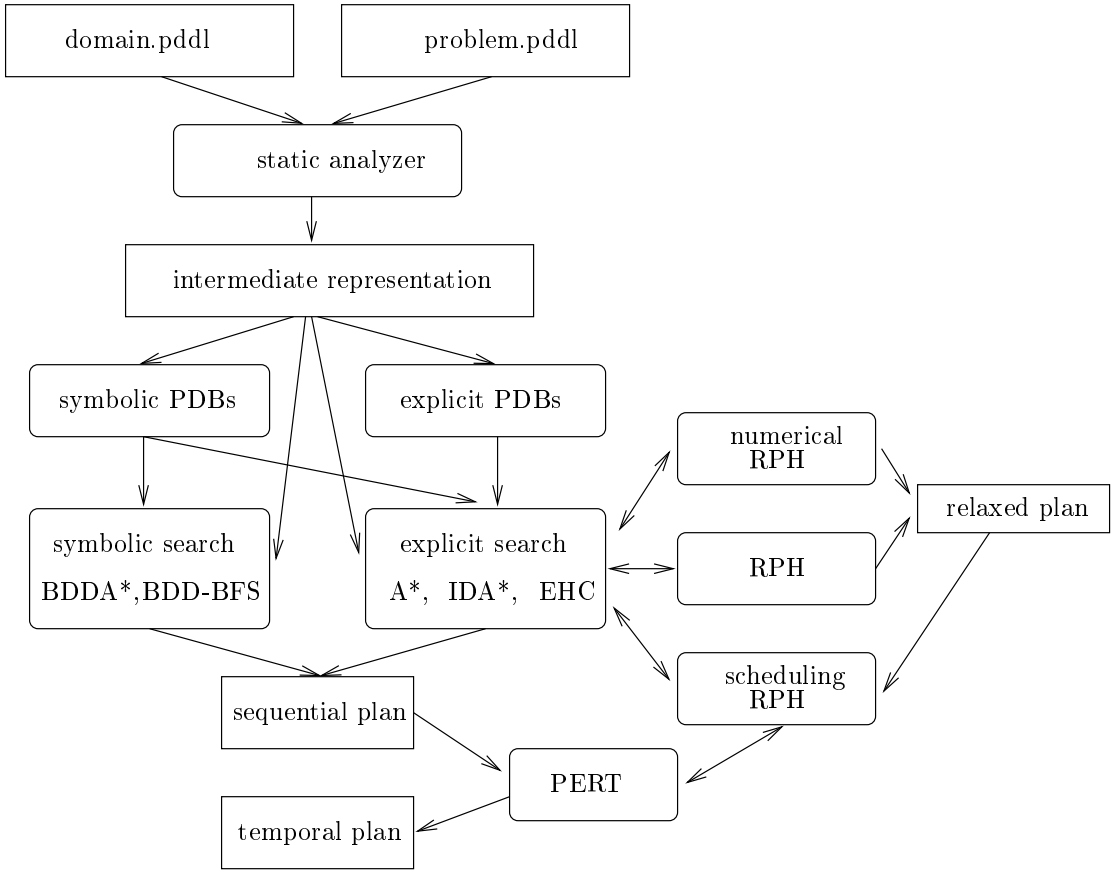

Figure 4: Architecture of MIPS

4.1 Heuristics

MIPS incorporates more than six different estimates.

- Relaxed planning heuristic (RPH): Approximation of the number of planning steps needed to solve the propositional planning problem with all delete effects removed (Hoffmann & Nebel, 2001). The heuristic is constructive, i.e. it returns the set of operators that appear in the relaxed plan.

- Numerical relaxed planning heuristic (numerical RPH): Our numerical extension to RPH is a combined propositional and numerical forward and backward approximation scheme, also allowing for multiple operator application. Our version for integrating numbers into the relaxed planning heuristic is sound, but not as general as Hoffmann's contribution (Hoffmann, 2002a): it restricts to variable-to-constant comparisons, and lacks the simplification of linear constraints.

- Pattern databases heuristic (explicit PDB heuristic): Explicit PDBs were already mentioned in the historical overview of MIPS. The different abstractions are found in a greedy best-fit bin-packing manner, yielding a selection of large PDBs in form of perfect hash tables that fit into main memory. If necessary, PDBs can be designed to be disjoint yielding an admissible estimate (Edelkamp, 2001c).

- Symbolic pattern database heuristic (symbolic PDB heuristic): Symbolic PDBs apply to both explicit and symbolic heuristic search engines. Due to the succinct BDD-representation of sets of states the averaged heuristic estimate can be increased while decreasing the number of nodes to be explored in the overall search. Symbolic PDBs are often orders of

```
(define (grounded zeno-travel-zeno-travel-1)
  (:fluents
     (at dan city-a)   (at dan city-b)   (at dan city-c)   (at dan city-d)
     (at ernie city-a) (at ernie city-b) (at ernie city-c) (at ernie city-d)
     (at plane city-a) (at plane city-b) (at plane city-c) (at plane city-d)
     (at scott city-a) (at scott city-b) (at scott city-c) (at scott city-d)
     (in dan plane)    (in ernie plane)  (in scott plane))
  (:variables (fuel plane) (total-fuel-used) (total-time))
  (:init
     (at dan city-c)   (at ernie city-c) (at plane city-a) (at scott city-a)
     (= (fuel plane) 750) (= (total-fuel-used) 0) (= (total-time) 0))
  (:goal  (at dan city-a)   (at ernie city-d) (at scott city-d))
  (:metric minimize (total-time) )
  (:group dan
    (at dan city-a)    (at dan city-b)   (at dan city-c)    (at dan city-d)
    (in dan plane))
  (:group ernie
    (at ernie city-a)  (at ernie city-b) (at ernie city-c)  (at ernie city-d)
    (in ernie plane))
  (:group plane
    (at plane city-a)  (at plane city-b) (at plane city-c)  (at plane city-d))
  (:group scott
    (at scott city-a)  (at scott city-b) (at scott city-c)  (at scott city-d)
    (in scott plane))
```

Figure 5: Grounded Representation of *Zeno-Travel* Domain.

magnitudes larger than explicit ones. Due to state conversion into a Boolean representation the retrieval of a heuristic estimate is slower than hashing, but still linear in the state description length (Edelkamp, 2002b),

- Scheduling relaxed plan heuristic (scheduling RPH, SRPH): Critical-path analysis by PERT scheduling may also guide the plan finding phase. Different to the RPH heuristics, which computes the length of the greedily extracted plan, SRPH also takes the sequence of operators into account and searches for a good parallel arrangement. Adding PERT-schedules for the path to a state and for the sequence of actions in the relaxed plan is not as accurate as the PERT-schedule of the combined paths. Therefore, the classical merit function of A*-like search engines $f = g + h$ of generating path length g and heuristic estimate h is not immediate. We define the heuristic value of SRPH as the parallel plan length of the combined path minus the parallel plan length of the generating path.

- One suitable combination of the PDB heuristic and RPH heuristics that is also implemented in MIPS, compares the retrieved result of the PDBs with the set of operators in the plan graph that respect the abstraction. The intuition is to slice the relaxed plan graph. If in the backward exploration an add-effect is selected the match will be assigned to its fact group. If the number of matches in an abstraction is smaller than the retrieved PDB value it will be increased by the lacking amount.

In the competition, except for numerical domains we chose pure RPH for sequential plan generation and scheduling PRH for temporal domains. Only in pure numerical problems we used numerical RPH. We have experimented with (symbolic) PDBs with mixed results. Since in our implementation PDBs are purely propositional and do not allow the retrieval of the corresponding operator sets of the optimal abstract plan, we have not included PDB search in the competition version of MIPS.

```
  (:action board dan plane city-a
   :condition
     (and (at dan city-a) (at plane city-a))
   :effect
     (and (in dan plane)  (not (at dan city-a))
        (increase (total-time) (30.000000))))
  [...]
  (:action zoom plane city-a city-b
   :condition
     (and
        (at plane city-a)
        (>= (fuel plane) (300.000000)))
   :effect
     (and (at plane city-b)  (not (at plane city-a))
        (increase (total-time) (60.000000))
        (increase (total-fuel-used) (300.000000))
        (decrease (fuel plane) (300.000000))))
  [...]
  (:action refuel plane city-a
   :condition
     (and
        (at plane city-a)
        (< (fuel plane) (750.000000)))
   :effect
     (and
        (increase (total-time) (/ (- (750.000000) (fuel plane)) (12.500000)))
        (assign (fuel plane) (750.000000))))
 [...]
)
```

Figure 6: Grounded Representation of *Zeno-Travel* Domain (cont.).

4.2 Exploration Algorithms

The algorithm portfolio includes:

- Weighted A* (weighted A*/A*): The A* algorithm (Hart, Nilsson, & Raphael, 1968) can be casted as a derivate of Dijkstra's SSSP exploration on a re-weighted graph. For lower bound heuristics, original A* can be shown to generate optimal plans (Pearl, 1985). Weightening the influence of the heuristic estimate may accelerate solution finding, but also affects optimality (Pohl, 1977). The set of horizon nodes are maintained in a priority queue *Open*, while the settled nodes are kept in *Closed*.

  In MIPS, Weighted A* is implemented with a Dial or a Weak-Heap priority queue data structure (Dial, 1969; Edelkamp & Stiegeler, 2002). The former is used for propositional planning only, while the latter applies to general planning with scheduling estimates. Arrays have been implemented as a dynamic table that double their sizes if they become filled. MIPS stores all generated and expanded states in a hash table. An alternative, yet not implemented, but more flexible storage structure is collection of persistent trees as in the TL planning system (Bacchus & Kabanza, 2000), one for each predicate. In the best case queries and update times to the structure are logarithmic in the number of represented atoms.

- Weighted Iterative-Deepening A* ((W)IDA*): The memory-limited variant of (Weighted) A* is well-suited to large exploration problems with efficient evaluation functions of small integer range (Korf, 1985). In MIPS, IDA* is extended with bit-state hashing (Edelkamp

& Meyer, 2001) to improve duplicate detection with respect to ordinary transposition tables (Reinefeld & Marsland, 1994). This form of partial search effectively trades state-space coverage for completeness. For a further compression of the planning state space, all variables that appear in the objective function are neglected from hash address calculations and state comparisons.

- Enforced Hill Climbing (EHC): The approach is another compromise between exploration and exploitation. EHC searches with an improved evaluation in breadth-first manner and commits established decisions as final (Hoffmann, 2000). EHC is complete in undirected problem graphs and seems to have a slight advantage to Weighted A* when combined with RPH and other pruning cuts. On the other hand, it can be misguided in unstructured planning domains and is likely to get lost in problem graphs with dead-ends.

- Bidirectional Symbolic Breadth-First-Search (BDD-BFS): The implementation performs bidirectional blind symbolic search, choosing the next search direction in favor to the faster executions of the previous iterations (Edelkamp & Helmert, 1999).

- Weighted Symbolic A* (BDDA*): The algorithm performs guided symbolic search and takes a (possibly partitioned) symbolic representation of the heuristic as an additional input. Given a consistent estimate for a uniformly weighted graph, BDDA* performs at most $\mathcal{O}(f^{*^2})$ iterations, where f^* is the optimal solution length, where consistent estimates keep the accumulated f-values on each exploration path monotonical increasing.

- Weak and Strong Planning: These two symbolic exploration algorithms suited to non-deterministic planning have been added to MIPS (Cimatti, Roveri, & Traverso, 1998), but due to the lack of an agreed standard for a domain description language, the implementation was only tested on deterministic samples in which the above symbolic algorithms clearly perform better. The encoding scheme directly transfers to the non-deterministic scenario, where plans were stored in a form of state-action tables.

In the competition we applied Weighted A* with weight 2, e.g. the merit for all states $S \in \mathcal{S}$ was fixed as $f(S) = g(S) + 2 \cdot h(S)$, yielding good but not necessarily optimal plans. In temporal domains we introduced an additional parameter δ to scale the influence between propositional estimates ($f_p(S) = g_p(S) + 2 \cdot h_p(S)$) and scheduled ones ($f_s(S) = g_s(S) + 2 \cdot h_s(S)$). More precisely, we altered the comparison function for the priority queue, so that a comparison of parallel length priorities was invoked if the propositional difference of values was not larger than $\delta \in \mathbb{N}_0$. A higher value of δ refers to a higher influence of the SRPH, while $\delta = 0$ indicates no scheduling at all. In the competition we produced data with $\delta = 0$ (Pure MIPS), and $\delta = 2$ (optimized MIPS).

5 Temporal Planning

PDDL 2.1 domain descriptions include temporal modifiers *at start*, *over all*, and *at end*, where label *at start* denotes the preconditions and effects at invokation time of the action, *over all* refers to an invariance condition and *at end* to the finalization conditions and consequences of the action.

5.1 Temporal Model

In Figure 7 we show two different options to flatten this information back to planning with preconditions and effects to derive its semantic.

In the first case (top right), the compound operator is split into three smaller parts, one for action invocation, one for invariance maintenance, and one for action termination. This is the semantic suggested by (Fox & Long, 2001).

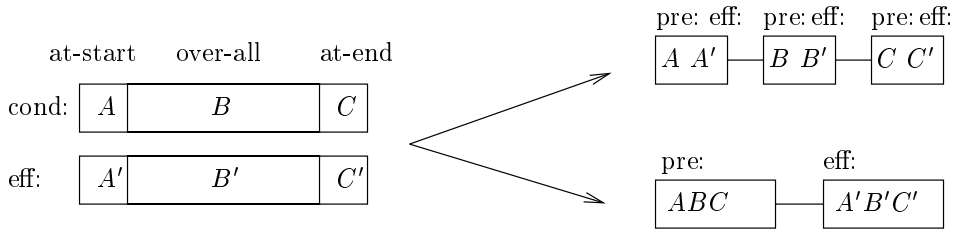

Figure 7: Compiling Temporal Modifiers into Operators.

As expected there are no effects in the invariance pattern, i.e. $B' = \emptyset$. Moreover, we found that in the benchmarks it is uncommon that new effects in `at-start` are preconditioned for termination control or invariance maintenance, i.e. $A' \cap (B \cup C) = \emptyset$.

Therefore, in MIPS the simpler second operator representation model was chosen (bottom right). The intermediate format of the example problem in Figures 5 and 6 implicitly assumed this simpler temporal model. At least for sequential plan finding we have not observed any deficiencies by assuming this temporal model, in which each action starts immediately after a previous one has terminated.

This simple temporal model motivates the definition of the first plan objective: the sequential plan.

Definition 4 *(Sequential Plan) A* sequential plan $\pi_s = (O_1, \ldots, O_k)$ *is an ordered sequence of operators* $O_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, *that transforms the initial state* $\mathcal{I}$ *into one of the goal states* $G \in \mathcal{G}$, *i.e., there exists a sequence of states* $S_i \in \mathcal{S}$, $i \in \{0, \ldots, k\}$, *with* $S_0 = \mathcal{I}$, $S_k = G$ *and* S_i *is the outcome of applying* O_i *to* S_{i-1}, $i \in \{1, \ldots, k\}$.

Minimizing sequential plan length was the only objective in the first and second planning competition. Since *Graphplan*-like planners (Blum & Furst, 1995) like IPP (Koehler, Nebel, & Dimopoulos, 1997) and STAN (Long & Fox, 1998) already produced parallel plans, this was a indeed a limiting aspect to evaluate plan quality. The most important reason for this artificial restriction was that total-ordered plans were easier accessible for automated validation, a necessity for evaluating correctness in a competitive scenario.

5.2 Operator Dependency

The formal definition of operator dependency allows to avoid the transpositioned generation of independent actions and, more importantly, enables optimal schedules of sequential plans with respect to the generated action sequence and its causal structure. If all operators are dependent (or void with respect to the optimizer function), the problem is inherent sequential and no schedule leads to any improvement.

Definition 5 *(Dependency/Mutex Relation) Two grounded operators* $o = (\alpha, \beta, \gamma, \delta)$ *and* $o' = (\alpha', \beta', \gamma', \delta')$ *in* $\mathcal{O}$ *are* dependent/mutex, *if one of the following three conditions holds:*

1. *The propositional precondition set of one operator has a non-empty intersection with the add or the delete lists of the other one, i.e.,* $\alpha \cap (\beta'_a \cup \beta'_d) \neq \emptyset$ *or* $(\beta_a \cup \beta_d) \cap \alpha' \neq \emptyset$.

2. *The head of a numerical modifier of one operator is contained in some condition of the other one, i.e. there exists a* $c' = (h'_c, \otimes, t'_c) \in \gamma'$ *and a* $m = (h_m, \oplus, t_m) \in \delta$ *with* $h_m \in \text{LeafVariables}(t'_c) \cup \{h'_c\}$ *or there exists a* $c = (h_c, \otimes, t_c) \in \gamma$ *and a* $m' = (h'_m, \oplus, t'_m) \in \delta'$ *with* $h'_m \in \text{LeafVariables}(t_c) \cup \{h_c\}$. *Intuitively, an operator modifies variables that appear in the condition of the other. This may be referred to as a* direct conflict.

3. *The head of the numerical modifier of one operator is contained in the formula body of the modifier of the other one, i.e., there exists a* $m = (h_m, \oplus, t_m) \in \delta$ *and* $m' = (h'_m, \oplus, t'_m) \in \delta'$ *with* $h_m \in \text{LeafVariables}(t'_m)$ *or* $h'_m \in \text{LeafVariables}(t_m)$. *This may be referred to as an* indirect conflict.

The dependence relation may be refined according to the PDDL 2.1 guidelines for mutual exclusion (Fox & Long, 2001), but for our purposes for improving sequential plans this approach is sufficient. In our implementation (at least for temporal and numerical planning) the dependence relation is computed beforehand and tabulated for constant time access. To improve the efficiency of pre-computation, the set of leaf variables is maintained in an array, once the grounded operator is constructed.

To detect domains for which any parallelization leads to no improvement, a planning domain is said to be *inherently sequential* if all operators in any sequential plan are dependent or instantaneous (i.e. with zero duration). The static analyzer checks this by testing each operator pair. While some benchmark domains like *DesertRats* and *Jugs-and-Water* are inherently sequential, others like *ZenoTravel* and *Taxi* are not.

Operator independence also indicates transpositions of two operators o_1 and o_2 to safely prune exploration in sequential plan generation.

Definition 6 *(Parallel Plan) A* parallel plan $\pi_c = ((O_1, t_1), \ldots, (O_k, t_k))$ *is a* schedule *of operators* $O_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, *that transforms the initial state* $\mathcal{I}$ *into one of the goal states* $G \in \mathcal{G}$, *where* O_i *is executed at time* t_i.

(Bäckström, 1998) clearly distincts partial ordered plans $(O_1, \ldots, O_k, \preceq)$, with the relation $\preceq \subseteq \{O_1, \ldots, O_k\}^2$ being a partial order (reflexive, transitive, and antisymmetric), from parallel plans $(O_1, \ldots, O_k, \preceq, \#)$, with $\# \subseteq (\preceq \cup \preceq^{-1})$ (irreflexive, symmetric) expressing, which actions must not be executed in parallel.

Definition 7 *(Precedence Ordering) A ordering* $\preceq_d$ *induced by the set of operators* $\{O_1, \ldots, O_k\}$ *and a dependency relation is given by* $O_i \preceq_d O_j$, *if* O_i *and* O_j *are dependent and* $1 \leq i < j \leq k$.

Precedence is not a partial ordering, since it is neither reflexive nor transitive. By computing the transitive closure of the relation, however, precedence could be extended to a partial ordering. A sequential plan $O_1, \ldots, O_k$ produces an acyclic set of precedence constraints $O_i \preceq_d O_j$, $1 \leq i < j \leq k$, on the set of operators. It is also important to observe, that the constraints are already topologically sorted according to $\preceq_d$ by taking the node ordering $\{1, \ldots, k\}$.

Definition 8 *(Respecting Precedence Ordering in Parallel Plan) Let* $d(O)$ *for* $O \in \mathcal{O}$ *be the* duration *of operator* O *in a sequential plan. For a parallel plan* $\pi_c = ((O_1, t_1), \ldots, (O_k, t_k))$ *that respect* $\preceq_d$, *we have* $t_i + d(O_i) \leq t_j$ *for* $O_i \preceq_d O_j$, $1 \leq i < j \leq k$.

For optimizing plans (Bäckström, 1998) defines *parallel execution time* as $\max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$, so that if $O_i \preceq O_j$, then $t_i + d(O_i) \leq t_j$, and if $O_i \# O_j$, then either $t_i + d(O_i) \leq t_j$ or $t_j + d(O_j) \leq t_i$. These two possible choices in # are actually not apparent in practice, since we already have a precedence relation at hand and just seek the optimal arrangement of operators. In contrast we assert that only one option, namely $t_i + d(O_i) \leq t_j$ can be true, reducing # to $\preceq_d$. More importantly, (Bäckström, 1998)'s work introduces unnecessary time complexity, since optimized scheduling a set of fixed-timed operators is already an NP-complete problem.

Definition 9 *(Optimal Parallel Plan) An optimal parallel plan with respect to a sequence of operators* $O_1, \ldots, O_k$ *and precedence ordering* $\preceq_d$ *is a parallel plan* $\pi^* = ((O_1, t_1), \ldots, (O_k, t_k))$ *with minimal parallel execution time* $OPT = \max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$ *among all parallel plans* $\pi_c = ((O_1, t'_1), \ldots, (O_k, t'_k))$ *that respect* $\preceq_d$.

Procedure *Critical-Path*
Input: *Sequence of operators* $O_1, \ldots, O_k$, precedence ordering $\preceq_d$
Output: Optimal parallel plan length $\max\{t_i + d(O_i) \mid O_i \in \{O_1, \ldots, O_k\}\}$
for all $i \in \{1, \ldots, k\}$
$e(O_i) = d(O_i)$
for all $j \in \{1, \ldots, i-1\}$
if $(O_j \preceq_d O_i)$
if $e(O_i) < e(O_j) + d(O_i)$
$e(O_i) \leftarrow e(O_j) + d(O_i)$
return $\max_{1 \leq i \leq k} e(O_i)$

Table 2: Algorithm to Compute Critical Path Length.

Many algorithms have been suggested to convert sequential plans into partial ordered ones (Pednault, 1986; Regnier & Fade, 1991; Veloso, Pérez, & Carbonell, 1990). Most of them interpret a total ordered plan as a maximal constrained partial ordering $\preceq = \{(O_1, O_j) \mid 1 \leq i < j \leq k\}$ and search for least constraint plans. However, the problem of minimum constraint "deordering" has also been proven to be NP-hard, except if the so-called validity check is polynomial (Bäckström, 1998), where deordering maintains validity of the plan by lessening its constraintness, i.e. $\preceq' \subseteq \preceq$ for a new ordering $\preceq'$.

Since we have an explicit model of dependency and time, optimal parallel plans will not change the ordering relation $\preceq_d$ at all.

5.3 Critical Path Analysis

The *Project Evaluation and Review Technique* (PERT) is a critical path analysis algorithm usually applied to project management problems. The critical path is established, when the total time for activities on this path is greater than any other path of operators. A delay in any tasks on the critical path leads to a delay in the project. The heart of PERT is a network of tasks needed to complete a project, showing the order in which the tasks need to be completed and their dependencies between them. As shown in Table 2, PERT scheduling reduces to a derivate of Dijkstra's single shortest path algorithm within acyclic graphs (Cormen, Leiserson, & Rivest, 1990).

In the algorithm, $e(O_i)$ is the tentative earliest end time of operator O_i, $i \in \{1, \ldots, k\}$, while the earliest starting times t_i for all operators in the optimal plan are given by $t_i = e(O_i) - d(O_i)$.

Theorem 1 *(PERT Scheduling) Given a sequence of operators* $O_1, \ldots, O_k$ *and a precedence ordering* $\preceq_d$ *an optimal parallel plan* $\pi^* = ((O_1, t_1), \ldots, (O_k, t_k))$ *can be computed in optimal time* $\mathcal{O}(k + \mid \preceq_d \mid)$.

Proof: The proof is done by induction on $i \in \{1, \ldots, k\}$. The induction hypothesis is that after iteration i the value $e(O_i)$ is correct, e.g. $e(O_i)$ is the earliest end time of operator O_i. This is clearly true for $i = 1$, since $e(O_1) = d(O_1)$. We now assume that the hypothesis is true $1 \leq j < i$ and look at iteration i. There are two choices. Either there is a $j \in \{1, \ldots, i-1\}$ with $(O_j \preceq_d O_i)$. For this case after the inner loop is completed, $e(O_i)$ is set to $\min\{e(O_j) + d(O_j) \mid O_j \preceq_d O_i, j \in \{1, \ldots, i-1\}\}$. On the other hand, $e(O_i)$ is optimal, since O_i cannot start earlier than $\min\{e(O_j) \mid O_j \preceq_d O_i, j \in \{1, \ldots, i-1\}\}$, since all values $e(O_j)$ are already the smallest possible by induction hypothesis. If there is no $j \in \{1, \ldots, i-1\}$ with

$(O_j \preceq_d O_i)$, then $e(O_i) = d(O_i)$ as in the base case. Therefore, at the end $\max_{1 \leq i \leq k} e(O_i)$ is the optimal parallel path length.

The time and space complexities of the algorithm *Critical-Path* are clearly in $\mathcal{O}(k^2)$, where k is the length of the sequential plan. Using an adjacency list representation these efforts can be reduced to time and space proportional to the number of vertices and edges in the dependence graph, which are of size $\mathcal{O}(k + | \preceq_d |)$. The bound is optimal, since the input consists of $\Theta(k)$ operators and $\Theta(| \preceq_d |)$ dependencies among them. □

5.4 Graphplan Distances

In this section we restrict the planning model to *valid STRIPS plans* as in the original article of *Graphplan* (Blum & Furst, 1995), where the execution cost of each operator is 1 and the semantics of a parallel plan are as follows.

For each time step i, $i \in \{1, \ldots, l\}$, a state $S_i \in \mathcal{S}$ is generated by applying all operators with time stamp $i - 1$ to S_{i-1}, where $S_0 = \mathcal{I}$. An optimal parallel plan is a parallel plan of minimal length l. The name *dependency* is borrowed from the notion of partial order reduction in explicit-state model checking (Clarke et al., 1999), where two operators O_1 and O_2 are *independent* if for each state $S \in \mathcal{S}$ the following two properties hold:

1. *Enabledness* is preserved, i.e. O_1 and O_2 do not disable each other.
2. O_1 and O_2 are *commutative*, i.e. executed in any order O_1 and O_2 lead to the same state.

Two actions *interfere*, if they are dependent. The original *Graphplan* definition is very closed to ours, which fixes interference as $\beta'_d \cap (\beta_a \cup \alpha) \neq \emptyset$ and $(\beta'_a \cup \alpha') \cap \beta_d \neq \emptyset$.

Lemma 1 *If $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, operator inference in the* Graphplan *model is implied by the propositional MIPS model of dependence.*

Proof: If $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, for two independent operators $o = (\alpha, \beta)$ and $o' = (\alpha', \beta')$: $\alpha \cap (\beta'_a \cup \beta'_d) = \emptyset$ implies $\beta_d \cap (\beta'_a \cup \beta'_d) = \emptyset$, which in turn yields $\beta_a \cap \beta'_d = \emptyset$. The condition $\beta'_a \cap \beta_d = \emptyset$ can be inferred analogously by exchanging primed and unprimed variables. □

Theorem 2 *Two independent STRIPS operators $o = (\alpha, \beta)$ and $o' = (\alpha', \beta')$ in $\mathcal{O}$ with $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$ are* enabledness preserving *and* commutative, *i.e. for all states in $S \subseteq 2^{|A|}$ we have $o(o'(S)) = o'(o(S))$.*

Proof: Since $\beta_d \subseteq \alpha$ and $\beta'_d \subseteq \alpha'$, we have $\beta_a \cap \beta'_d = \emptyset$ and $\beta'_a \cap \beta_d = \emptyset$ by Lemma 1. Let S' be the state $((S \setminus \beta_d) \cup \beta_a)$ and let S'' be the state $((S \setminus \beta'_d) \cup \beta'_a)$. Since $(\beta'_a \cup \beta'_b) \cap \alpha = \emptyset$, o is enabled in S'', and since $(\beta_a \cup \beta_b) \cap \alpha' = \emptyset$, o' is enabled in S'. Moreover,

$$
\begin{aligned}
o(o'(S)) &= (((S \setminus \beta'_d) \cup \beta'_a) \setminus \beta_d) \cup \beta_a \\
&= (((S \setminus \beta'_d) \setminus \beta_d) \cup \beta'_a) \cup \beta_a \\
&= S \setminus (\beta'_d \cup \beta_d) \cup (\beta'_a \cup \beta_a) \\
&= S \setminus (\beta_d \cup \beta'_d) \cup (\beta_a \cup \beta'_a) \\
&= (((S \setminus \beta_d) \setminus \beta'_d) \cup \beta_a) \cup \beta'_a \\
&= (((S \setminus \beta_d) \cup \beta_a) \setminus \beta'_d) \cup \beta'_a = o'(o(S))
\end{aligned}
$$

□

A less restrictive notion of independence, in which several actions may occur at the same time even if one deletes an add-effect of another is provided in (Knoblock, 1994).

All three models of valid plans are restrictive, since they assume that for each parallel plan there exist at least one corresponding total ordered plan. In general, however, this is not true. Consider the simple STRIPS planning problem domain with $\mathcal{I} = \{B\}$, $\mathcal{G} = \{\{A, C\}\}$, and $\mathcal{O} = \{(\{B\}, \{A\}, \{B\}), (\{B\}, \{C\}, \{B\})\}$. Obviously, both operators are needed for goal achievement, but there is no sequential plan of length 2, since B is deleted in both operators. However, a parallel plan could be executed, since all precondition are fulfilled at the first time step.

5.5 Full Enumeration Algorithms

Even though full state-space enumaration is far from being practical they provide a basis for heurisitic search engines. In optimal parallel plans, each operator either starts or ends at the start or end time of another operator. Therefore, for a fixed number of operators, we can assume a possibly exponential but finite number of possible parallel plans.

This immediately leads to the following plan enumeration algorithm *ENUM-1*. For all $|\mathcal{O}|^i$ operator sequences of length i, $i \in \mathbb{N}$, generate all possible partial orderings, check for each individual schedule if it transforms the initial state into one of the goals, and take the sequence with smallest parallel plan length. Since all parallelizations are computed we have established the following result.

Theorem 3 *If the number of operators for an optimal parallel plan is bounded, ENUM-1 is complete and computes optimal parallel plans.*

Note that the first i with a matching solution does not necessarily yield an optimal parallel path, since longer operator sequences might rise better parallel solutions. ENUM-1 can also generate non-valid plans in the *Graphplan* model. For a better distinction between the objectives for parallel plans, we keep the notion of *validity* in this section.

Assuming only valid plans implies that each parallel plan corresponds to at least one (possible many) sequential ones. Viewed from the opposite side, each partial-ordered plan can be established by generating a totally-ordered plan first and then apply a scheduling algorithm to it to find its best partial-order. Therefore, the next two enumeration schemes produce valid plans only.

Enumeration algorithm *ENUM-2* generates all feasible sequential plans of length i with increasing $i \in \mathbb{N}$, and computes their optimal schedule with respect to the number of operators and dependency property. Since optimal parallelization of all valid operator sequences are computed we have established the following theorem.

Theorem 4 *If the number of operators for an optimal parallel plan is bounded, ENUM-2 is complete and computes a valid optimal parallel plan.*

A complete enumeration scheme of all sequential plans that transform the initial state into one goal state is also still computationally expensive, but ruling out impossible operator applications drastically reduces the vast number of $|\mathcal{O}|^i$ operator sequences of length i. Bäckström's result for deriving partial orders has shown, that given the sequence of operators in a sequential plan, to infer an optimized partial order that respects a set of mutexes is NP-hard, so that even for the second phase no polynomial-time algorithm is to be expected. Therefore, at least for STRIPS we have restricted the PSPACE-hard planning task (Bylander, 1994) to an NP-hard problem for each generated sequential plan.

When the concept of mutual exclusion is extended to a precedence relation between operators, there exist at least one sequential plan that respects the set of operators and the set of precedence constraints. From the opposite point of view, for each sequential plan there exist at least one parallel plan that respects both the number of operators and the imposed set of precedence constraints.

Algorithm *ENUM-3* is a straight variant of *ENUM-2* that simply applies PERT scheduling for finding the optimal parallel plan, with the main difference that it additionally maintains the causal structure.

We have seen that *ENUM-1* may generate parallel plans that *ENUM-2* cannot produce. Are there also valid plans that *ENUM-2* can produce, but *ENUM-3* cannot? The answer is no. If *ENUM-2* terminates with an optimal schedule, we generate a corresponding sequential plan while preserving the causal structure. Optimal PERT-scheduling of this plan with respect to the set of operators and the imposed precedence relation will yield back the optimal parallel plan. Since all sequential paths are eventually generated, the given partial will also be found by *ENUM-3*. This proves the following result.

Theorem 5 *If the number of operators for an optimal parallel plan is bounded, ENUM-3 is complete and computes a valid optimal parallel plan.*

In the following, we interpret optimized parallel plans as nodes in a weighted directed graph $G = (V, E, w)$. Edges correspond to possible extensions of the plans with an additional operator, which can be found by a sequentialization of the parallel plan followed by a PERT scheduling operation. The weight function denotes the difference in parallel plan length. Since the set of operators and the precedence set is enlarged, all weights will be greater than or equal to 0. If only a finite number of actions can be executed in parallel, then any infinite path in G has unbounded cost. Therefore, we can traverse G in shortest path ordering using Dijkstra's algorithm to finally yield an optimal parallel plan.

The argument for optimality is that Dijkstra's algorithm is complete, i.e., it cannot exit with failure, since if the horizon list becomes becomes empty there is no solution at all. If the horizon is not empty, there is at least one node on an optimal solution path, which has to be selected before any goal node with larger cost.

Theorem 6 *If only a finite number of actions can be executed in parallel, Dijkstra's shortest path enumeration is complete and computes a valid optimal parallel plan.*

5.6 Heuristic Search Enumeration

The enumeration algorithms in the previous section are sound, complete and optimal in theory. On the other hand enumeration schemes do not contradict known undecidability results in numerical planning (Helmert, 2002). If we have no additional information like a bound to the maximal number of actions in a plan or on the number of actions that can be executed in parallel, we cannot say if the enumeration will terminate or not.

The main drawback of the above approaches is that they are seemingly too slow for practical planning. Heuristic search algorithms like A* and IDA* reorder the traversal of states in the planning problem, and an admissible estimate does not affect completeness and optimality. The reason for completeness in finite graphs is that the number of acyclic paths in G is finite and with every node expansion, A* adds new links to its traversal tree. Each newly added link represents a new acyclic path, so that the reservoir of path must eventually be exhausted. The argument is valid for any best-first strategy that prunes cyclic paths, but by their move-committing nature, hill-climbing algorithms are not complete.

(Pearl, 1985) has shown that A* is complete even on infinte graphs, demanding that the cost of every infinite path is unbounded. A deeper investigation shows that given an admissible estimate there must always be a node in the current search horizon with optimal priority. Actually to preserve this condition for admissible but not necessarily consistent estimates, already expanded node may have to be reconsidered (re-opening). Hence, A* must also terminate with an optimal solution.

Theorem 7 *If the cost of every infinite plan is unbounded, A* enumeration with an admissible parallel plan length estimate computes a valid optimal parallel plan.*

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
130: (board ernie plane city-c) [30]
160: (refuel plane city-c) [40]
200: (zoom plane city-c city-a) [100]
300: (debark dan plane city-a) [20]
320: (board scott plane city-a) [30]
350: (refuel plane city-a) [40]
390: (zoom plane city-a city-c) [100]
490: (refuel plane city-c) [40]
530: (zoom plane city-c city-d) [100]
630: (debark ernie plane city-d) [20]
650: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (board ernie plane city-c) [30]
100: (refuel plane city-c) [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c) [40]
420: (zoom plane city-c city-d) [100]
520: (debark ernie plane city-d) [20]
520: (debark scott plane city-d) [20]
```

Figure 8: A Sequential Plan for *Zeno-Travel* (left) and its PERT Schedule (right).

Note that the assumption of unbounded sequential plan costs is not true in all benchmark problems, since there may be an infinite sequence of instantaneous events that do not contribute to the plan objective. For example, loading and unloading tanks in *DesertRats* does not affect `total-fuel` consumption, which is to be minimized in one benchmark instance.

As a matter of fact, informative admissible parallel plan length estimates are not easy to obtain. This was the reason in MIPS to chose sequential plan generation first, because very effective heuristics are known to generate sequential plans quickly. With the SRPH we choose a parallel plan length approximation, but since it extends PRH, it is known to be not admissible.

5.7 Pruning Anomalies

Other acceleration techniques like sequential plan hashing, symmetry and transposition cuts have to be chosen carefully to maintain parallel plan length optimality.

Take for example sequential state memorization, i.e. the memorization of states in the sequential plan generation process. This approach does affect parallel optimality, as the following example shows.

Consider the sequences

`(zoom city-a city-c plane)`, `(board dan plane)`, `(refuel plane)`,
`(zoom city-c city-a plane)`, `(board scott)`, `(debark dan)`, `(refuel plane)`,

and

`(board scott)`,`(zoom city-a city-c plane)`,`(board dan plane)`,
`(refuel plane)`, `(zoom city-c city-a plane)`, `(debark dan)`, `(refuel plane)`

in the *Zeno-Travel* problem. The set of operators is the same and so is the resulting (sequentially generated) state.

However, the PERT schedule for the first sequence is shorter than the schedule for the second one, since in the previous case the time for boarding `scott` is compensated by the remaining two operators.

For small problems, such anomalies can be avoided by avoided duplicate pruning at all. As an example Figure 8 depicts a sequential plan for the example problem instance and its PERT schedule, which turns out to be the overall optimal parallel plan.

In order to generate sequential solutions for large planning problem instances, in the competition version of MIPS we have introduced cuts that affect optimality but reduce the number of expansions significantly.

```
  0: (board scott plane city-a) [30]
 30: (fly plane city-a city-c) [150]
180: (board ernie plane city-c) [30]
180: (board dan plane city-c) [30]
210: (fly plane city-c city-a) [150]
360: (debark dan plane city-a) [20]
360: (refuel plane city-a) [53.33]
413.33: (fly plane city-a city-c) [150]
563.33: (fly plane city-c city-d) [150]
713.33: (debark ernie plane city-d) [20]
713.33: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (board ernie plane city-c) [30]
100: (refuel plane city-c) [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (fly plane city-a city-c) [150]
430: (fly plane city-c city-d) [150]
580: (debark ernie plane city-d) [20]
580: (debark scott plane city-d) [20]
```

Figure 9: Optimized Plans in *Zeno-Travel* according to different Plan Objectives.

5.8 Arbitrary Plan Objectives

In PDDL 2.1 different plan metrics can be devised. In Figure 9 we depict two plans found by MIPS when modifying the objective function from minimizing `total-time` to minimize `total--fuel-used`, and to minimize the compound `(+ (* 10 (total-time)) (* 1 (total-fuel--used)))`.

For the first case we computed an optimal value of 1,333.33, while for the second case we established 7,666.67 as the optimized merit. When optimizing time, the ordering of board and zoom actions is important. When optimizing *total-fuel* we reduce speed to save fuel consumption to 333.33 per flight but we may board the first passenger immediately. We also save two refuel actions with respect to the first case.

When increasing the importance of time we can trade refueling actions for time, so that both zooming and flight actions are chosen for the complex minimization criterion.

We first thought, that we could simply substitute the plan objective in the PERT scheduling process. However, the results did not match with the ones produced by the validator (Long & Fox, 2001a), in which the final time is substituted in the objective function after the plan has been build.

The way we evaluate objective functions that include time is as follows. First we schedule the (relaxed or final) sequential plan. Then we temporarily substitute the `total-time` value in the state with the parallel plan length and evaluate the formula to get the objective function value. To avoid conflicts in subsequent expansions, afterwards we set the value `total-time` back to the optimal one in the sequential plan.

6 Symmetry

An important feature of parameterized predicates, functions and action descriptions in the domain specification file is that actions are transparent to different bindings of parameters to objects. Disambiguating information is present in the problem instance file.

In case of typed domains, many planners, including MIPS, compile all type information into additional predicates, attach additional preconditions to actions and enrich the initial states by suitable object-to-type atoms.

As a consequence, a symmetry is viewed as a permutation of objects that is present in the current state, in the goal representation, and transparent to the set of operators.

There are $n!$, $n = |\mathcal{OBJ}|$, possible permutations of the set of objects. Taking into account all type information reduces the number of all possible permutation to

$$\binom{n}{t_1, \ldots, t_k} = \frac{n!}{t_1! \cdot \ldots \cdot t_k!}.$$

where t_i is the number of objects with type i, $i \in \{1, \ldots, k = |\mathcal{TYPES}|\}$. In a moderate sized logistic domain with 10 cities, 10 trucks, 5 airplanes, and 15 packages, this results in $40!/(10! \cdot 10! \cdot 5! \cdot 15!) \geq 10^{20}$ permutations.

To reduce the number of potential symmetries to a tractable size we restrict symmetries to object transpositions, for which we have at most $n(n-1)/2 \in \mathcal{O}(n^2)$ candidates. Including type information this number further reduces to

$$\sum_{i=1}^{k} \binom{t_i}{2} = \sum_{i=1}^{k} t_i(t_i - 1)/2.$$

In the following, the set of typed object transpositions is denoted by $\mathcal{SYMM}$. For the example, we have $|\mathcal{SYMM}| = 45 + 45 + 10 + 105 = 205$.

6.1 Static Symmetries

We generate a set of object pairs $(o, o') \in \mathcal{SYMM}$, indistinguishable with respect to the set of instantiated operators and the goal specification.

Definition 10 *(Object Transpositions for Fluents, Variables, and Operators) A transposition of objects $(o, o') \in \mathcal{SYMM}$ applied to a fluent $f = (p\ o_1, \ldots, o_{k(p)}) \in \mathcal{F}$, written as $f[o \leftrightarrow o']$, is defined as $(p\ o'_1, \ldots, o'_{k(p)})$, with $o'_i = o_i$ if $o_i \notin \{o, o'\}$, $o_i = o'$ if $o_i = o$, and $o_i = o$ if $o_i = o'$, $i \in \{1, \ldots, k(p)\}$. Object transpositions $[o \leftrightarrow o']$ applied to a variable $v = (f\ o_1, \ldots, o_{k(f)}) \in \mathcal{V}$ or to an operator $O = (a\ o_1, \ldots, o_{k(a)}) \in \mathcal{O}$ are defined analogously.*

By definition we have

Lemma 2 *For all $f \in \mathcal{F}$, $v \in \mathcal{V}$, $O \in \mathcal{O}$, and $(o, o') \in \mathcal{SYMM}$: $f[o \leftrightarrow o'] = f[o' \leftrightarrow o]$, $v[o \leftrightarrow o'] = v[o' \leftrightarrow o]$, $O[o \leftrightarrow o'] = O[o' \leftrightarrow o]$, $f[o \leftrightarrow o'][o \leftrightarrow o'] = f$, $v[o \leftrightarrow o'][o \leftrightarrow o'] = v$, and $O[o \leftrightarrow o'][o \leftrightarrow o'] = O$.*

The time complexity for checking $f[o \leftrightarrow o']$ is of order $\mathcal{O}(k(p))$. By precomputing a $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{F}|)$ sized table containing the index of $f' = f[o \leftrightarrow o']$ for all $(o, o') \in \mathcal{SYMM}$, this time complexity can be reduced to $\mathcal{O}(1)$.

Definition 11 *(Object Transpositions for States) An* object transposition $[o \leftrightarrow o']$ *applied to state $S = (S_p, S_n) \in \mathcal{S}$ with $S_n = (v_1, \ldots, v_k)$, $k = |\mathcal{V}|$, written as $S[o \leftrightarrow o']$, is equal to $(S_p[o \leftrightarrow o'], S_n[o \leftrightarrow o'])$ with*

$$S_p[o \leftrightarrow o'] = \{f' \in \mathcal{F} \mid f \in S_p \ \wedge \ f' = f[o \leftrightarrow o']\}$$

and $S_n[o \leftrightarrow o'] = (v'_1, \ldots, v'_k)$ with $v_i = v'_j$ if $\phi^{-1}(i)[o \leftrightarrow o'] = \phi^{-1}(j)$ for $i, j \in \{1, \ldots, k\}$.

The time complexity to compute $S_n[o \leftrightarrow o']$ is $\mathcal{O}(k)$, since testing $\phi^{-1}(i)[o \leftrightarrow o'] = \phi^{-1}(j)$ is available in time $\mathcal{O}(1)$ by building another $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{V}|)$ sized precomputed look-up table. We summarize the complexity issues as follows.

Lemma 3 *The time complexity to compute $S[o \leftrightarrow o']$ for state $S = (S_p, S_n) \in \mathcal{S}$ and $(o, o') \in \mathcal{SYMM}$ is $\mathcal{O}(|S_p| + |\mathcal{V}|)$ using $\mathcal{O}(|\mathcal{SYMM}| \cdot (|\mathcal{F}| + |\mathcal{V}|)$ space.*

Definition 12 *(Object Transpositions for Domains) A planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is symmetric with respect to the object transposition $[o \leftrightarrow o']$, abbreviated as $\mathcal{P}[o \leftrightarrow o']$, if $\mathcal{I}[o \leftrightarrow o'] = \mathcal{I}$ and for all $G \in \mathcal{G}$: $G[o \leftrightarrow o'] \in \mathcal{G}$.*

Applying Lemma 3 for all $(o, o') \in \mathcal{SYMM}$ yields

Theorem 8 *Assuming a description complexity $\mathcal{O}(|\mathcal{G}_p| + |\mathcal{V}|)$ for the set of goals $\mathcal{G}$, checking whether a planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is symmetric with respect to the object transpositions $[o \leftrightarrow o']$, with $(o, o') \in \mathcal{SYMM}$ can be done in time $\mathcal{O}(|\mathcal{SYMM}| \cdot (|\mathcal{G}_p| + |\mathcal{I}_p| + |\mathcal{V}|)$.*

Lemma 4 *If operator O is applicable in S and $S = S[o \leftrightarrow o']$ then $O[o \leftrightarrow o']$ is applicable in S and*

$$O(S)[o \leftrightarrow o'] = O[o \leftrightarrow o'](S)$$

Proof: If O is applicable in S the $O[o \leftarrow o']$ is applicable in $S[o \leftarrow o']$. Since $S = S[o \leftrightarrow o']$, $O[o \leftrightarrow o']$ applicable in S, and

$$O[o \leftrightarrow o'](S) = O[o \leftrightarrow o'](S[o \leftrightarrow o']) = O(S)[o \leftrightarrow o'].$$

□

Lemma 4 indicates how symmetry will be used to reduce exploration. If a planning problem with current state $\mathcal{C} \in \mathcal{S}$ is symmetric with respect to the operator transposition $[o \leftrightarrow o']$ then either the application of operator $O \in \mathcal{O}$ or the application of operator $O[o \leftrightarrow o']$ is neglected, significantly reducing the branching factor.

6.2 Dynamic Symmetries

One problem is that symmetries that are present in the initial state may vanish or reappear during exploration. In the *DesertRats* domain, for example, the initial set of supply tanks is indistinguishable so that only one should be loaded into the truck. Once the fuel level of the supply tanks decrease or tanks are transported to another location, formerly existing symmetries are broken. However, when two tanks in one location are emptied they can once more be considered as being symmetric.

In a forward chaining planner goal conditions do not change over time, only the initial state $\mathcal{I}$ transforms to the current state $\mathcal{C}$. Therefore, in a precompiling phase we refine the set $\mathcal{SYMM}$ to

$$\mathcal{SYMM}' := \{(o, o') \in \mathcal{SYMM} \mid \forall\, G \in \mathcal{G} : G[o \leftrightarrow o'] = G\}\,.$$

Usually $|\mathcal{SYMM}'|$ is by far smaller than $|\mathcal{SYMM}|$. For the *Zeno-Travel* instance, the symmetries left in $\mathcal{SYMM}'$ are the ones of the location of `scott` and `ernie`.

Theorem 9 *Checking whether an induced planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle$ with current state $\mathcal{C} = (\mathcal{C}_p, \mathcal{C}_n) \in \mathcal{S}$ is symmetric with respect to the object transpositions $[o \leftrightarrow o']$, $(o, o') \in \mathcal{SYMM}'$, can be performed in time $\mathcal{O}(|\mathcal{SYMM}'| \cdot (|\mathcal{C}_p| + |\mathcal{V}|))$.*

Therefore, we can efficiently compute set

$$\mathcal{SYMM}''(\mathcal{C}) := \{(o, o') \in \mathcal{SYMM}' \mid \mathcal{C}[o \leftrightarrow o'] = \mathcal{C}\}$$

of symmetries that are present in the current state. In the initial state of the example problem of *Zeno-Travel* $\mathcal{SYMM}''(\mathcal{C}) = \emptyset$, but once `scott` and `ernie` share the same location in a state $\mathcal{C} \in \mathcal{S}$ this object pair would be included in $\mathcal{SYMM}''(\mathcal{C})$.

By precomputing a $\mathcal{O}(|\mathcal{SYMM}| \cdot |\mathcal{O}|)$ sized table the index of operator $O' = O[o \leftrightarrow o']$ can be determined in time $\mathcal{O}(1)$ for each $(o, o') \in \mathcal{SYMM}'$.

Let $\Gamma(S)$ be the set of operators that are applicable in state $S \in \mathcal{S}$.

Definition 13 *The* pruning set $\Delta(S, \mathcal{SYMM}''(\mathcal{C})) \subset \Gamma(S)$ *is defined as the set of all operators that have a symmetric operator and that are not of minimal index, i.e.,* $\Delta(S, \mathcal{SYMM}''(\mathcal{C})) =$

$$\{O \in \Gamma(S) \mid \exists\, O' \in \Gamma(S) : \phi(O') > \phi(O) \text{ and } \exists\, (o, o') \in \mathcal{SYMM}''(\mathcal{C}) : O' = O[o \leftrightarrow o']\}\,.$$

The symmetry reduction *of $\Gamma(S, \mathcal{SYMM}''(\mathcal{C})) \subseteq \Gamma(S)$ with respect to the set $\mathcal{SYMM}''(\mathcal{C})$ is defined as $\Gamma(S, \mathcal{SYMM}''(\mathcal{C})) = \Gamma(S) \setminus \Delta(S, \mathcal{SYMM}''(\mathcal{C}))$.*

To shorten notation, in the following we write $\Gamma'(\mathcal{C})$ for $\Gamma(S, \mathcal{SYMM}''(\mathcal{C}))$ and $\Delta'(\mathcal{C})$ for $\Delta(S, \mathcal{SYMM}''(\mathcal{C}))$. Determining $\Gamma'(\mathcal{C})$ can be performed in time $\mathcal{O}(|\Gamma(S)|)$, since finding $O' = O[o \leftrightarrow o']$ and the indices $\phi(O')$ and $\phi(O)$ are all available in constant time.

Theorem 10 *Reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma(S, \mathcal{SYMM}''(\mathcal{C}))$ during the exploration of planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ preserves completeness and sequential optimality for all expanded states $\mathcal{C}$.*

Proof: Suppose that for some expanded state $\mathcal{C}$, reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma'(\mathcal{C})$ during the exploration of planning problem $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ does not preserve completeness and sequential optimality. Furthermore, let $\mathcal{C}$ be the state with this property that is maximal in the exploration order.

Then there is a sequential plan $\pi = \{O_1 \ldots, O_k\}$ in $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle$ with intermediate state sequence $S_0 = \mathcal{C}, \ldots, S_k \subseteq \mathcal{G}$. Obviously, $O_i \in \Gamma(S_{i-1})$, $i \in \{1, \ldots, k\}$. By the choice of $\mathcal{C}$ we have $O_1 \notin \Gamma'(S_0)$. Since $O_1 \notin \Gamma'(S_0)$ but $O_1 \in \Gamma(S_0)$ we have that $O_1 \in \Delta(S_0, \mathcal{SYMM}''(S_0))$. By the definition of the pruning set $\Delta'(S_0)$ there exists O_1', $\phi(O_1') > \phi(O_1)$ and $(o, o') \in \mathcal{SYMM}''(S_0)$ with $O_1' = O_1[o \leftrightarrow o'] \in \Gamma'(S_0)$ that is applicable in S_0. By Lemma 4 we have $O_1'(S_0) = S_1[o \leftrightarrow o']$.

Since $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{G} \rangle = \mathcal{P}[o \leftrightarrow o'] = \langle \mathcal{S}, \mathcal{O}, \mathcal{C}[o \leftrightarrow o'] = \mathcal{C}, \mathcal{G}[o \leftrightarrow o'] = \mathcal{G} \rangle$, we have a sequential plan $O_1[o \leftrightarrow o'], \ldots, O_k[o \leftrightarrow o']$ with state sequence $S_0[o \leftrightarrow o'] = S_0, S_1[o \leftrightarrow o'], \ldots, S_k[o \leftrightarrow o'] = S_k$ that reaches the goal $\mathcal{G}$.

Sequential plan objectives are devised on parameterized predicates and functions, so that any cost function on $O_1[o \leftrightarrow o'], \ldots, O_k[o \leftrightarrow o']$ will be the same as on $O_1, \ldots, O_k$. This contradicts the assumption that reducing the operator set $\Gamma(\mathcal{C})$ to $\Gamma'(\mathcal{C})$ does not preserve completeness and optimality for all $\mathcal{C}$. □

If the plan objective is defined on instantiated predicates and objects, it can be symmetry breaking and to preserve optimality should be checked as an additional requirement similar to $\mathcal{G}$ and $\mathcal{I}$.

6.3 Symmetry Reduction in MIPS

The main purpose of the refined implementation in MIPS is to reduce the time for dynamic symmetry detection from $\mathcal{O}(|\mathcal{SYMM}'| \cdot (|\mathcal{C}_p| + |\mathcal{V}|))$ to time $\mathcal{O}(|\mathcal{C}_p| + |\overline{\mathcal{SYMM}'}| \cdot |\mathcal{V}|)$ by loosing some but not all structural properties.

The key observation is that symmetries are also present in fact groups according to their object representatives. Fact groups $G_i \subseteq \mathcal{F}$, $i \in \{1, \ldots, l\}$ implicitly define projections $\mathcal{P}|_i$ of the (propositional) planning space $\mathcal{P}$ by $\mathcal{P}|_i = \langle \mathcal{S}|_i, \mathcal{O}|_i, \mathcal{I}|_i, \mathcal{G}|_i \rangle$, with $\mathcal{S}|_i = G_i$, $\mathcal{I}|_i = \mathcal{I} \cap G_i$, $\mathcal{G}|_i = \bigcup_{G \in \mathcal{G}} G \cap G_i$, and $\mathcal{O}|_i = \{(\beta_a, \beta_b) \in \mathcal{O} \mid (\beta_a \cup \beta_b) \cap \mathcal{S}|_i \neq \emptyset\}$. By construction for all $S \in \mathcal{S}$ we have exactly one fact in each group true, such that S can be partitioned into $\{S_1, \ldots, S_l\}$, with $S_i \in \mathcal{S}|_i$, $i \in \{1, \ldots, l\}$.

Let $R_i \subseteq \mathcal{OBJ}$ be the set of object representatives for group G_i. If $S[o \leftrightarrow o'] = S$ then $\mathcal{S}|_i[o \leftrightarrow o'] = \mathcal{S}|_j$ in a group G_j with representative $R_i[o \leftrightarrow o']$. Hence, in MIPS we devise a symmetry relation $\overline{\mathcal{SYMM}}$ not on objects but on fact groups, i.e.

$$\overline{\mathcal{SYMM}} = \{(i, j) \mid 1 \leq i < j \leq l : R_i[o \leftrightarrow o'] = R_j\}.$$

Many objects, e.g. the objects of type `city` in *ZenoTravel*, were not selected as representatives for a single attribute invariance to build a group. These were neglected in MIPS, since we expect no symmetry on them. This reduces the set of objects $\mathcal{OBJ}$ that MIPS considers to a considerably smaller subset $\mathcal{OBJ}' = \bigcup_{\{1 \leq i \leq l\}} R_i$. In the example problem $|\mathcal{OBJ}| = 7$, and $|\mathcal{OBJ}'| = 4$.

It may also happen that more than one group has a representative $o \in \mathcal{OBJ}'$. However, if all fluent predicates p have arity $k(p) \leq 2$, which is frequently met in the benchmark domains, all

$|R_i|$ were equal to one for all i, so for all objects we get a finite partitioning into representatives, i.e. $\mathcal{OBJ}' = \dot{\bigcup}_{i \in \{1,\ldots,l\}} R_i$.

MIPS takes this conservative assumption and may leave other symmetries uncaught. It computes $\overline{\mathcal{SYMM}}$ by analyzing the subproblem structures $\mathcal{P}|_i$, $i \in \{1,\ldots,l\}$ instead of $\mathcal{P}$ itself. In case of an object symmetry $[R_i \leftrightarrow R_j]$ the groups G_i and G_j necessarily have to be isomorphic, and we can establish a bijective mapping $\psi : \mathcal{P}|_i \to \mathcal{P}|_j$ with subcomponents $\psi_{\mathcal{S}} : \mathcal{S}|_i \to \mathcal{S}|_j$ and $\psi_{\mathcal{O}} : \mathcal{O}|_i \to \mathcal{O}|_j$.

As above, static symmetries based on non-matching goal predicates were excluded, yielding a refinement $\overline{\mathcal{SYMM}'}$ of $\overline{\mathcal{SYMM}}$. Dynamic symmetries are detected for each expanded state S. The current state representation is mapped to the subgraphs $\mathcal{P}|_i$. The list of possibly symmetric groups $\overline{\mathcal{SYMM}'}$ is traversed to select pairs which obey the current instantiation. MIPS marks the groups with larger index as *visited*. This guarantees that operators of at least one group are executed. The complexity of this phase is bounded by $|S_p|$ and by $\overline{\mathcal{SYMM}'}$ and yields a list $\overline{\mathcal{SYMM}''}$.

Testing the propositional part $S_p = (S_1,\ldots,S_l)$ of a state S for all symmetries reduces to test, whether $\psi_{\mathcal{S}}(S_i) = S_j$ for each $(i,j) \in |\overline{\mathcal{SYMM}'}|$ and can be performed in time $\mathcal{O}(|S_p| + |\overline{\mathcal{SYMM}'}|)$. The comparison of variables $v \in \mathcal{V}$ is implemented as described in the previous section such that for the numerical part S_n we check the remaining symmetries, for total time $\mathcal{O}(|S_p| + |\overline{\mathcal{SYMM}'}| \cdot |\mathcal{V}|)$ to fix $\overline{\mathcal{SYMM}''}$ in form of visited markings.

For each expanded state S and each matching operator $O \in \Gamma(S)$ the algorithm checks, whether an applied operator is present in a visited group, in which case it is pruned. The time complexity is in $\mathcal{O}(|\Gamma(S)|)$, since operator group containment can be preprocessed and checked in constant time.

7 Visualization

For visualization of plans we extended an existing animation system for our purposes.

Vega (Hipke, 2000) is implemented as a Client-Server architecture that runs a annotated algorithm on the server side to be visualized on the client side in a Java frontend. Annotation are visualization requests that (minorly) extend the existing source code by (simple) commands like *send point*(x,y) or *send circle*(x,y,r). In the system visualization objects can be associated with hierarchical structured identifiers. The client is used both as the user front-end and the visualization engine. Thus, it allows server and algorithm selection, input of data, running and stopping algorithms, and customization of the visualization.

It can be used to *i*) manipulate scenes with hierarchically named objects — either in the view or in an object browser that displays the object tree, *ii*) view algorithm lists at the server and display algorithm information, *iii*) apply algorithms to selected data in a view, control the algorithm execution using a VCR-like panel or the execution browser, *iv*) adjust view attributes directly or using the object browser, show several algorithms simultaneously in multiple scenes and open different views for a single scene, and *v*) load and save single scenes, complete runs, and attribute lists, export scenes in xfig or gif format.

Vega allows on-line and off-line presentations. The main purpose of the server is to make algorithms accessible through TCP/IP. The server is able to receive commands from multiple clients at the same time. It allows the client to choose from the list of available algorithms, to retrieve information about the algorithm, to specify input data, to start it and to receive output data. The server maintains a list of installed algorithms. This list may be changed without the need of stopping and restarting the server.

We have extended Vega with two respects, and call it Vepa for *Visualization of Efficient Planning Algorithms* to emphasize the planning aspect. It can be run as an interactive applet available at `www.informatik.uni-freiburg.de/~mmips/visualization`.

The first program that we added is *VepaServer* which wraps plan execution and visualizes

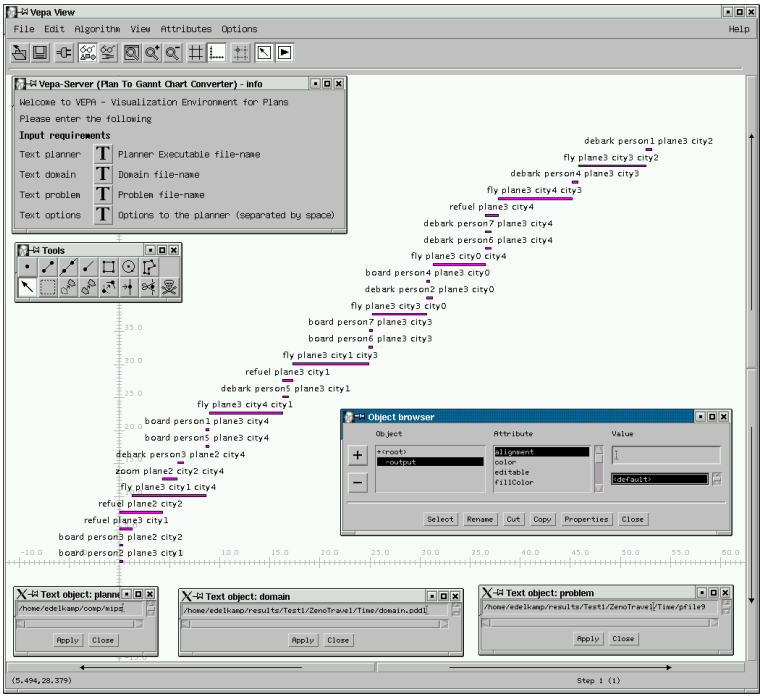

Figure 10: Visualization a Plan in Gannt Chart Format.

Gannt Charts of plans (cf. Figure 10). Gannt Charts are a well known representation for schedules in which a horizontal open oblong is drawn against each activity indicating estimated duration. The tool can be adapted to any planner that writes plans in planning competition format to standard I/O. The user may choose the planner, the domain, and the problem file. Hence, the algorithm is fully planner independent. To avoid security conflicts in our web presentation of the visualizer we have incorporated a slightly different call panel that allows to select the planner in a pull-down menu and to include domain and problem by cut and paste.

The second program (suite) is *VepaDomain* for domain-dependent visualization of sequential plans. Figure 10 shows an example for the *Settlers* domain. *VepaDomain* includes instance independent visualizations for all competition domains all with less than 100 lines of code. The figures for the objects were collected with an image web search engine. We used *Google* (cf. `www.google.de`) in the advanced setting to search for small GIFs. *VepaDomain* works in cooperation with the MIPS system as follows. The planner writes propositional and numeric state facets and (unscheduled) action sequences into a file, which in turn is read by the domain dependent visualizer. Therefore, it is not difficult to adapt other planners to the domain visualizer.

8 Related Work

Solving planning problems with numerical preconditions and effects as allowed in Level 2 and Level 3 problems is undecidable in general (Helmert, 2002). However, the structures of the provided benchmark problems are simpler than the general problem class, so that these problems are in fact solvable.

8.1 Problem Classes and Methods

According to the PDDL-hierarchy we indicate three problem classes:

1. Propositional Planning. STRIPS problems have been tackled with different planning techniques, most notably by SAT-planning, e.g. (Kautz & Selman, 1996), IP-planning,

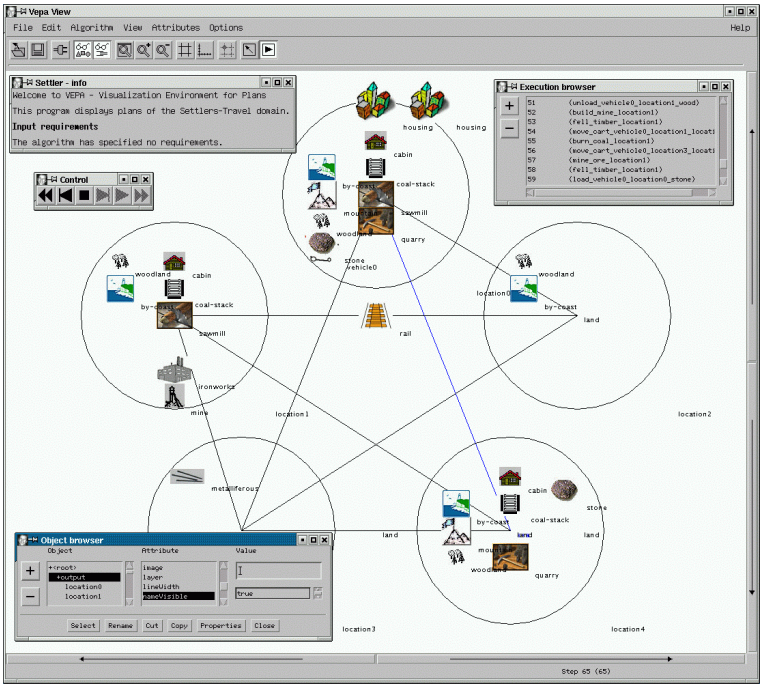

Figure 11: Visualization of a Planning Problem Instance of Settlers.

e.g. (Kautz & Walser, 1999), CSP-planning, e.g. (Rintanen & Jungholt, 1999), graph relaxation, e.g. (Blum & Furst, 1995), and heuristic search planing, e.g. (Bonet & Geffner, 2001). The major quality measurements are the numbers of sequential and parallel steps. ADL generalizations (Pednould, 1989) like conditional effects and negative preconditions are more expressive in general, but can usually be resolved during grounding.

2. Numerical Effects. Numerical variables in the effect lists can include time and resources. If numerical effects do not bound integral values, infinite state spaces are likely to be generated. However, by assuming finitely many interesting events the problem class becomes tractable and is effectively dealt by schedulers that usually minimize the *make-span* of concurrent actions.

3. Numerical Preconditions. We distinguish finite and infinite branching problems. With finite branching, execution time of an action is not parameterized, while with infinite branching, an infinite number of actions can be applied. These problems have ever since been confronted to model checking. Some subclasses of infinite branching problems like timed automata exhibit a finite partitioning through a symbolic representation of states (Pettersson, 1999). By the technique of shortest-path reduction a unique and reduced normal form can be obtained. We have implemented this constraint network data structure, since this is the main data structure when exploring timed automata as done by the model checker Uppaal (Pettersson, 1999). For this to work, all constraints must have the form $x_i - x_j \leq c$ or $x_i \leq c$. For example, the set of constraints $x_4 - x_0 \leq -1$, $x_3 - x_1 \leq 2$, $x_0 - x_1 \leq 1$, $x_5 - x_2 \leq -8$, $x_1 - x_2 \leq 2$, $x_4 - x_3 \leq 3$, $x_0 - x_3 \leq -4$, $x_1 - x_4 \leq 7$, $x_2 - x_5 \leq 10$, and $x_1 - x_5 \leq 5$ has the shortest-path reduction $x_4 - x_0 \leq -1$, $x_3 - x_1 \leq 2$, $x_5 - x_2 \leq -8$, $x_0 - x_3 \leq -4$, $x_1 - x_4 \leq 7$, $x_2 - x_5 \leq 10$, and $x_1 - x_5 \leq 5$. If the constraint set is over-constraint, the algorithm will determine unsolvability, otherwise a feasible solution is returned. The absence of partitioning is current research (Wolper & Boigelot, 1998).

Critical path analysis for timed precedence networks is one of the simpler cases for scheduling. We have achieved a simplification by solving the sequential path problem first. Several scheduling techniques apply the presented critical path analyses as a subcomponent (Syslo, Deo, & Kowalik, 1983).

Most previously achieved results in symmetry reduction, e.g. (Guéré & Alami, 2001), neglect the combinatorial explosion problem and tend to assume that the information on existing symmetries in the domain is supplied by the user. Our approach shares similarities with the approach of (Fox & Long, 1999, 2002) in inferring symmetry information automatically, which bases on the TIM inference module (Fox & Long, 1998). Since no additional information on the current symmetry level in form of matrix is stored, our approach consumes less space per state. Moreover, we can give correctness proofs and efficiency guarantees.

8.2 Competing Planners

The on-line presentation of IPC-3[4] provides aspects of the input language, domains, results and other resources, e.g links to competing planners and the history of the event. In the following we briefly present the successful approaches at AIPS-2002. In AIPS-1998 most successful planners besides HSP (Bonet & Geffner, 2001) and *Satplan* (Kautz & Selman, 1996) were *Graphplan* derivates, e.g. IPP (Koehler et al., 1997) and STAN (Long & Fox, 1998). In 2000, the field was dominated by the success of heuristic search planning as in FF (Hoffmann & Nebel, 2001), HSP-2 (Haslum & Geffner, 2000), and in some hybrids, like STAN4 (Long & Fox, 2001b), and MIPS. System R (Lin, 2001) used backward regression.

Metric-FF (Hoffmann, 2002a) extends FF (Hoffmann & Nebel, 2001) and is a forward chaining heuristic state space planner. It performed best in the numerical track and was the only system besides MIPS that solved instances to *Settlers*. The main heuristic of relaxed plans bases on the HSP-heuristic (Bonet & Geffner, 2001). Metric-FF deals with PDDL 2.1 level 2, combined with ADL. The key difference is the definition of the relaxation. In STRIPS, the task is relaxed by ignoring all delete lists. However, numerical constraints are not monotonic: while one constraint (e.g. $x > 2$) might prefer higher values of a variable x, another constraint (e.g. $x < 2$) might prefer lower values. Opposed to that, the conditions in the purely logical case all prefer *higher* values of the propositional variables: negative conditions are compiled away as a pre-process, and thus it is always preferable to have more propositional facts true. The observation exploited in Metric-FF is that the same methodology can be applied in the numerical setting, at least in a subset of the language. The task is pre-processed such that all numerical constraints are monotonic, i.e., for any constraint c, if c is true in a state S then c is true in any state S' where, for all variables x, $x(S') \geq x(S)$. The relaxation is then simply to ignore all effects that decrease the value of the affected variable, and the relaxed task can be solved in Graphplan-style. To achieve the monotonicity property, one needs, in the numerical constraints and effects, expressions that are monotonic in all variables. In the current implementation, Numerical-FF restricts to linear expressions which obviously have this property.

LPG (Local search for Planning Graphs) (Gerevini & Serina, 2002) is the only planner that was competitive with MIPS at AIPS-2002 in the temporal domains. It bases on local search and planning graphs that handles PDDL 2.1 domains involving numerical quantities and durations. The system can solve both plan generation and plan adaptation problems. The basic search scheme of LPG was inspired by Walksat, an efficient procedure to solve SAT-problems. The search space of LPG consists of *action graphs* (Gerevini & Serina, 1999), particular subgraphs of the planning graph representing partial plans. The search steps are certain graph modifications transforming an action graph into another one. LPG exploits a compact representation of the planning graph to define the search neighborhood and to evaluate its elements using a parametrized function, where the parameters weight different types of inconsistencies in the current partial plan, and are dynamically evaluated during search using discrete Lagrange multipliers. The evaluation function uses some heuristics for estimate the *search cost* and the *execution cost* of achieving a (possibly numeric) precondition. Action durations and numerical quantities (e.g., fuel consumption) are represented in the actions graphs, and are modeled in the evaluation function. In temporal domains, actions are ordered using a *precedence graph* that is

[4] http://www.dur.ac.uk/d.p.long/competition.html

maintained during search, and that took into account the mutex relations of the planning graph.

TP4 (Haslum & Geffner, 2001) is in fact a scheduling system based on grounded problem instances. For these cases all formula trees in numerical conditions and assignments reduce to constants. Utilizing admissible heuristics TP4 minimize the plan objective of optimal parallel plan length. Our planner has some distinctive advantages: it handles numerical preconditions, instantiates numerical conditions on the fly and can cope with complex objective functions. Besides input restriction, in the competition, TP4 was somewhat limited by is focus to produce optimal solutions only.

SAPA (Do & Kambhampati, 2001) is a domain-independent time and resource planner that can cope with metrics and concurrent actions. It adapts the forward chaining algorithm of (Bacchus & Ady, 2001). Both planning approaches instantiate actions on the fly and can, therefore, in principle be adapted to at least mixed propositional and numerical planning problems. The search algorithm of SAPA extends partial concurrent plans. It uses a relaxed temporal planning graph for the yet unplanned events for different heuristic evaluation functions. In the competition SAPA was the only system besides MIPS that produced plans for the complex domains, which was the only one it submitted solutions to.

8.3 Symbolic Model Checking based Planners

In the 2000 competition, two other symbolic planner took part: PropPlan (Fourman, 2000), and BDDPlan (Hölldobler & Stör, 2000). Although they were not awarded for performance, they show interesting properties. PropPlan performs symbolic forward breadth first search to explore propositional planning problems with propositions for generalized action preconditions and generalized action effects. It performed well in the full ADL Micsonic-10 elevator domain (Koehler, 2000). ProbPlan is written in the Poly/ML implementation of SML and the standart C-BDD library[5]. BDD-Plan bases on solving the entailment problem in the fluent calculus with BDDs. At that time the authors acknowledged that a concise domain encoding and symbolic heuristic search as found in MIPS provides a large space for improvements.

In the Model-Based Planner, MBP[6], the paradigm of planning as symbolic model checking (Giunchiglia & Traverso, 1999) has been implemented for *non-deterministic* planning domains (Cimatti et al., 1998), which classifies in weak, strong, and strong-cyclic planning, with plans that are represented as complete state-action tables. For *partial observable* planning, exploration faces the space of belief states; the power set of the original planning space. Therefore, in contrast to the successor set generation based on action application, observations introduce "And" nodes into the search tree (Bertoli, Cimatti, Roveri, & Traverso, 2001b). Since the approach is a hybrid of symbolic representation of belief states and explicit search within the "And"-"Or" search tree, simple heuristic have been applied to guide the search. The need for heuristics that trade information gain for exploration effort is also apparent need in *conformant* planning (Bertoli, Cimatti, & Roveri, 2001a). Recent work (Bertoli & Cimatti, 2002) proposes improved heuristic for belief space planning. MBP has not yet participated in a planning competition, but plan to do in 2004.

The UMOP system parses a non-deterministic agent domain language that explicitly defines a controllable system in an uncontrollable environment (Jensen & Veloso, 2000). The planner also applies BDD refinement techniques such as automated transition function partitioning. New result for the UMOP system extends the setting of weak, strong and strong cyclic planning to adversarial planning, in which the environment actively influences the outcome of actions. In fact, the proposed algorithm joins aspects of both symbolic search and game playing. UMOP has not participated yet in a planning competition.

More recent developments in symbolic exploration are expected to influence automated planning in near future. With SetA*, (Jensen, Bryant, & Veloso, 2002) provide an improved imple-

[5] http://www-2.cs.cmu.edu/ modelcheck/bdd.html

[6] http://sra.itc.it/tools/mbp

mentation of the symbolic heuristic search algorithm BDDA* (Edelkamp & Reffel, 1998) and Weighted BDDA* (Edelkamp, 2001a). One major surplus is to maintain a finer granularity of the sets of states in the search horizon kept in a matrix according to matching g- and h- values. This contrasts the plain bucket representation of the priority queue based on f-values. The heuristic function is implicitly encoded with value differences of grounded actions. Since sets of states are to be evaluated and some heuristics are state rather than operator dependent it has still to be shown how general this approch is. As above the considered planning benchmarks are seemingly simple for single-state heuristic search exploration (Hoffmann, 2002b; Helmert, 2001). (Hansen, Zhou, & Feng, 2002) also re-implemented BDDA* and suggest that symbolic search heuristics and exploration algorithms are probably better to be implemented with algebraic decision diagrams (ADDs), as available in Somenzi's CUDD package. Although the authors achieved no improvement to (Edelkamp & Reffel, 1998) to solve the $(n^2 - 1)$-Puzzle, the established generalization to guide a symbolic version of the LAO* exploration algorithm (Hansen & Zilberstein, 2001) for *probabilistic* (MDP) planning, results in a remarkable improvement to the state-of-the-art (Feng & Hansen, 2002).

9 Conclusions

With the competing planning system MIPS, we have contributed an object-oriented architecture for a forward chaining, heuristic explicit and symbolic search planner that finds plans in finite-branching numerical problems. The planner parses, pre-compiles, solves, and schedules all current benchmark problem instances include complex ones with duration, resource variables and different objective functions.

Model checking aspects have always been influencing to the development of MIPS, e.g in the static analysis to minimize the state description length, in symbolic exploration and plan extraction, in the dependence relation for PERT schedules according to a given partial order, in bit-state hashing for IDA*, etc. The successes of planning with MIPS were also exported back to model checking, as the development of a heuristic search explicit-state model checker HSF-SPIN (Edelkamp et al., 2002) indicates.

MIPS instantiates numerical pre- and postconditions on-the-fly and produces optimized parallel plans. Essentially planning with numerical quantities and durative actions is planning with time and resources. The given framework of mixed propositional and numerical planning problems problems and the presented intermediate format can be seen as a normal form for temporal and metric planning.

For temporal planning, MIPS generates sequential (totally ordered) plans and efficiently schedules them with respect to the set of actions and the imposed causal structure, without falling into known NP-hardness traps for optimized partial-ordering of sequentially generated plan. For smaller problems the enumeration approach guarantees optimal solutions. To improve solution quality in approximate enumeration, the (numerical) estimate for the number of operators was substituted by scheduling the relaxed plan in each state.

Other contributions besides the new expressivity were refined static analysis techniques to simplify propositionally grounded representation and to minimize state encoding, automated state-based dynamic symmetry detection, as well as effective hashing and transposition cuts.

In the main part of paper we have analyzed completeness and optimality of different forms of exploration and have given a troughout theoretical treatment of PERT scheduling and symmetry detection, proving correctness results and studying run-time complexities.

References

Bacchus, F., & Ady, M. (2001). Planning with resources and concurrency: A forward chaning approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 417–

424.

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*, 123–191.

Bäckström, C. (1998). Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, *9*, 99–137.

Bertoli, P., & Cimatti, A. (2002). Improving heuristics for planning as search in belief space. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Bertoli, P., Cimatti, A., & Roveri, M. (2001a). Heuristic search symbolic model checking = efficient conformant planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 467–472.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2001b). Planning in nondeterministic domains under partial observability via symbolic model checking. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 473–478.

Biere, A. (1997). μcke - efficient μ-calculus model checking. In *Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science, pp. 468–471. Springer.

Blum, A., & Furst, M. L. (1995). Fast planning through planning graph analysis. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1636–1642.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, *24*(3), 142–170.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 165–204.

Cimatti, A., Giunchiglia, E., Giunchiglia, F., & Traverso, P. (1997). Planning via model checking: A decision procedure for AR. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 130–142. Springer.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, pp. 875–881.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press.

Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L. J. (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, *98*(2), 142–170.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press.

Dial, R. B. (1969). Shortest-path forest with topological ordering. *Communication of the ACM*, *12*(11), 632–633.

Do, M. B., & Kambhampati, S. (2001). Sapa: a domain-independent heuristic metric temporal planner. In *European Conference on Planning (ECP)*, pp. 109–120.

Edelkamp, S. (2001a). Directed symbolic exploration and its application to AI-planning. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, pp. 84–92.

Edelkamp, S. (2001b). First solutions to PDDL+ planning problems. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, pp. 75–88.

Edelkamp, S. (2001c). Planning with pattern databases. In *European Conference on Planning (ECP)*. 13-24.

Edelkamp, S. (2002a). Mixed propositional and numerical planning in the model checking integrated planning system. In *International Conference on AI Planning & Scheduling (AIPS), Workshop on Temporal Planning*.

Edelkamp, S. (2002b). Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 135–147. Springer.

Edelkamp, S., & Helmert, M. (2000). On the implementation of MIPS. In *Artificial Intelligence Planning and Scheduling (AIPS)–Workshop on Model Theoretic Approaches to Planning*, pp. 18–25.

Edelkamp, S., & Helmert, M. (2001). The model checking integrated planning system MIPS. *AI-Magazine*, 67–71.

Edelkamp, S., Leue, S., & Lluch-Lafuente, A. (2002). Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology (STTT)*.

Edelkamp, S., & Meyer, U. (2001). Theory and practice of time-space trade-offs in memory limited search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science, pp. 169–184. Springer.

Edelkamp, S., & Reffel, F. (1998). OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, pp. 81–92.

Edelkamp, S., & Reffel, F. (1999). Deterministic state space planning with BDDs. In *European Conference on Planning (ECP), Preprint*, pp. 381–382.

Edelkamp, S., & Stiegeler, P. (2002). Implementing HEAPSORT with $n \log n - 0.9n$ and QUICKSORT with $n \log n + 0.2n$ comparisons. *ACM Journal of Experimental Algorithmics*.

Feng, Z., & Hansen, E. (2002). Symbolic heuristic search for factored markov decision processes. In *National Conference on Artificial Intelligence (AAAI)*.

Fikes, R., & Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fourman, M. P. (2000). Propositional planning. In *Artificial Intelligence Planning and Scheduling (AIPS)-Workshop on Model-Theoretic Approaches to Planning*, pp. 10–17.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Artificial Intelligence Research*, *9*, 367–421.

Fox, M., & Long, D. (1999). The detection and exploration of symmetry in planning problems. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 956–961.

Fox, M., & Long, D. (2001). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Tech. rep., University of Durham, UK.

Fox, M., & Long, D. (2002). Extending the exploitation of symmetries in planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Gerevini, A., & Serina, I. (1999). Fast planning through greedy action graphs. In *National Conference of Artificial Intelligence (AAAI)*.

Gerevini, A., & Serina, I. (2002). LPG: a planner based on local search for planning graphs with action costs. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Giunchiglia, F., & Traverso, P. (1999). Planning as model checking. In *European Conference on Planning (ECP)*, pp. 1–19.

Guéré, E., & Alami, R. (2001). One action is enough to plan. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Hansen, E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*, 35–62.

Hansen, E. A., Zhou, R., & Feng, Z. (2002). Symbolic heuristic search using decision diagrams. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics*, *4*, 100–107.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 140–149.

Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In *European Conference on Planning (ECP)*, pp. 121–132.

Helmert, M. (2001). On the complexity of planning in transportation domains. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 349–360. Springer.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Hipke, C. A. (2000). *Verteilte Visualisierung von Geometrischen Algorithmen*. Ph.D. thesis, University of Freiburg.

Hoffmann, J. (2000). A heuristic for domain independent planning and its use in an enforced hill climbing algorithm. In *ISMIS*, Lecture Notes in Computer Science, pp. 216–227. Springer.

Hoffmann, J. (2002a). Extending FF to numerical state variables. In *European Conference on Artificial Intelligence*.

Hoffmann, J. (2002b). Local search topology in planning benchmarks: A theoretical analysis. In *Artificial Intelligence Planning and Scheduling (AIPS)*.

Hoffmann, J., & Nebel, B. (2001). Fast plan generation through heuristic search. *Artificial Intelligence Research*, *14*, 253–302.

Hölldobler, S., & Stör, H.-P. (2000). Solving the entailment problem in the fluent calculus using binary decision diagrams. In *Artificial Intelligence Planning and Scheduling (AIPS)-Workshop on Model-Theoretic Approaches to Planning*, pp. 32–39.

Jensen, R. M., Bryant, R. E., & Veloso, M. M. (2002). SetA*: An efficient BDD-based heuristic search algorithm. In *National Conference on Artificial Intelligence (AAAI)*.

Jensen, R., & Veloso, M. M. (2000). OBDD-based universal planning for synchronized agents in non-deterministic domains. *Artificial Intelligence Research*, *13*, 189–226.

Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reachtive agents. *Artificial Intelligence*, *95*(1), 67–113.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, pp. 1194–1201.

Kautz, H., & Walser, J. (1999). State-space planning by integer optimization. In *National Conference on Artificial Intelligence (AAAI)*.

Knoblock, C. (1994). Generating parallel execution plans with a partial order planner. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 98–103.

Koehler, J. (2000). Elevator control as planning problem. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 331–338.

Koehler, J., Nebel, B., & Dimopoulos, Y. (1997). Extending planning graphs to an adl subset. In *European Conference on Planning (ECP)*, pp. 273–285.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*(1), 97–109.

Lago, U. D., Pistore, M., & Traverso, P. (2002). Planning with a language for extended goals. In *National Conference on Artificial Intelligence (AAAI)*.

Lin, F. (2001). System r. *AI-Magazine*, 73–76.

Lind-Nielsen, J. (1999). Buddy: Binary decision diagram package, release 1.7. Technical Univeristy of Denmark. jln@itu.dk.

Long, D., & Fox, M. (1998). Efficient implementation of the plan graph in STAN. *Artificial Intelligence Research*, *10*, 87–115.

Long, D., & Fox, M. (2001a). Encoding temporal planning domains and validating temporal plans. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*.

Long, D., & Fox, M. (2001b). Hybrid stan: Identifying and managing combinatorial optimisation sub-problems in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 445–452.

McDermott, D. (2000). The 1998 AI Planning Competition. *AI Magazine*, *21*(2).

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Press.

Pearl, J. (1985). *Heuristics*. Addison-Wesley.

Pednault, E. P. D. (1986). Formulating multiagend, dynamic-world problems in the classical framework. In *Reasoning about Action and Plans*, pp. 47–82. Morgan Kaufmann.

Pednould, E. (1989). ADL: Exploring the middleground between Strips and situation calculus. In *Knowledge Representation (KR)*, pp. 324–332. Morgan Kaufman.

Pettersson, P. (1999). *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. Ph.D. thesis, Department of Computer Systems, Uppsala University.

Pistore, M., & Traverso, P. (2001). Planning as model checking for extended goals in non-deterministic domains. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. *Machine Intelligence*, *8*, 55–72.

Reffel, F., & Edelkamp, S. (1999). Error detection with directed symbolic model checking. In *World Congress on Formal Methods (FM)*, pp. 195–211.

Regnier, P., & Fade, B. (1991). Détermination du parallélisme maximal et optimisation temporelle dans les plans d'actions linéaires. *Revue d'intelligence artificielle*, *5*(2), 67–88.

Reinefeld, A., & Marsland, T. (1994). Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(7), 701–710.

Rintanen, J., & Jungholt, H. (1999). Numeric state variables in constraint-based planning. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, pp. 109–121. Springer.

Syslo, M. M., Deo, N., & Kowalik, J. S. (1983). *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall.

Veloso, M. M., Pérez, M. A., & Carbonell, J. G. (1990). Nonlinear planning with parallel resource allocation. In *Innovative Approaches to Planning, Scheduling and Control*, pp. 207–212.

Wolper, P., & Boigelot, B. (1998). Verifying systems with infinite but regular state spaces. In *Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pp. 88–97. Springer.

Decidability and Undecidability Results for Planning with Numerical State Variables

Malte Helmert
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 052, 79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

These days, propositional planning can be considered a quite well-understood problem. Good algorithms are known that can solve a wealth of very different and sometimes challenging planning tasks, and theoretical computational properties of both general STRIPS-style planning and the best-known benchmark problems have been established.

However, propositional planning has a major drawback: The formalism is too weak to allow for the easy encoding of many genuinely interesting planning problems, specifically those involving numbers. A recent effort to enhance the PDDL planning language to cope with (among other additions) numerical state variables, to be used at the third international planning competition, has increased interest in these issues.

In this contribution, we analyze "STRIPS with numbers" from a theoretical point of view. Specifically, we show that the introduction of numerical state variables makes the planning problem undecidable in the general case and many restrictions thereof and identify special cases for which we can provide decidability results.

Introduction

In recent years, we have seen significant progress on the solution of propositional planning tasks. Fully automated planning systems such as FF (Hoffmann & Nebel 2001) and systems using hand-tailored control knowledge such as **TALplanner** (Doherty & Kvarnström 2001) show impressive performance on the problem suite of the second international planning competition. Many researchers have seen this as an indication that it is time to move on to richer – and perhaps more interesting – domain definition languages than the STRIPS or ADL subsets of PDDL, and consequently, the PDDL language has been extended to allow for numerical state variables, explicit models of concurrency, and specification of other metrics of plan quality than sequential length.

The result of this effort, the PDDL2.1 language defined by Fox and Long (2001), has the potential to become *the* representation language for planning problems involving numerical conditions and effects, especially since it is going to be used for the third international planning competition at AIPS 2002.

From a practician's point of view, this means that it is time some new planning algorithms capable of handling numerical features were developed, and indeed a number of recent publications in this area can be found (Haslum & Geffner 2001; Do & Kambhampati 2001).

From a theoretician's point of view, this is a good opportunity to look into decidability and complexity issues that arise in the context of numerical state variables. There is no work that we are aware of in the planning literature that addresses these issues, so in this paper we make a first attempt at closing this gap.

In the following analysis, we restrict ourselves to what Fox and Long have called "level 2" of PDDL2.1, not considering the PDDL2.1 formulation of concurrency or plan metrics. There are five main reasons behind that decision.

First, only investigating one new language feature at a time makes it easier to judge the impact of the addition of that feature. If we presented results for "full" PDDL2.1 instead, it might not be immediately obvious which new features actually contribute to the increased hardness of the task.

Second, related to this, restricting ourselves to one feature makes the results of our analysis more accessible, easier to understand, and allows for providing a more in-depth picture than we would have been able to give if we had chosen a panorama view of PDDL2.1.

Third, we think that the addition of numerical state variables is the feature of PDDL2.1 that will have the highest number of supporters, because it is immediately plausible and there are few competing models. In contrast, Fox and Long themselves have presented an interesting alternative model for concurrent planning domains with their PDDL+ language, which favors modeling actions of the planning agent as instantaneous and modeling change over time through environmental processes. It is not obvious if and how decidability results for PDDL2.1 with concurrency carry over to PDDL+, and vice versa.

Fourth, some of the other additions, especially concerning plan metrics, are only of importance when we are concerned with *optimal* or *near-optimal* planning. Our analysis is concerned with the more fundamental problem of finding *any* plan, because we feel that at this time, we cannot reasonably expect to optimally solve bigger problems with numerical features. While there are first results for optimally solving

planning problems of this kind (Haslum & Geffner 2001), just finding some plan for these domains seems a goal that is ambitious enough at the moment.

Last, we will see in the following sections that the addition of numerical features to the representation language is already sufficient to witness a sharp increase in the hardness of the planning task. Thus, we feel no immediate need to investigate even harder planning tasks.

The rest of this paper is structured as follows: In the following two sections, we introduce a formal notion of planning with numerical state variables and a convenient way of specifying restrictions to the domain definition language. After that, we prove some undecidability and decidability results for the general case and restrictions thereof. Finally, we discuss the implications of our analysis, have a look at related work, and conclude.

Planning with Numbers

To get started, we must introduce a notion of planning with numerical state variables. Our planning formalisms are based on propositional STRIPS, enhanced with numerical preconditions, numerical goal conditions, and numerical effects. We will look into a variety of different planning formalisms, each sharing the same basic propositional planning properties, but differing in what kind of numerical features they exhibit. For that reason, we will identify planning formalisms with their numerical features.

Definition 1 ***Planning formalism***
*A **planning formalism** is a triple $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, where $\mathcal{G}, \mathcal{P}, \mathcal{E} \subseteq \bigcup_{n \geq 1}(\mathbb{Q}^n \to \mathbb{Q})$. The three components of $\mathcal{F}$ are called its **numerical goal condition functions**, **numerical (operator) precondition functions**, and **numerical effect functions**, respectively.*

The sets of functions that define a planning formalism characterize the calculations that are allowed to be part of a condition or effect evaluation. To see how this works, we first need some auxiliary definitions.

Definition 2 ***Some terminology***
*For the rest of this definition, let V_P and V_N be disjoint finite sets called **propositional state variables** and **numerical state variables**, respectively.*

*The set of **states** (of (V_P, V_N)) is defined as $S = \{\ (\alpha, \beta) \mid \alpha : V_P \to \{\bot, \top\}, \beta : V_N \to \mathbb{Q}\ \}$.*

*A **propositional condition** is given by a propositional variable $v \in V_P$. It is written as $v = \top$. It is **satisfied** by state (α, β) iff $\alpha(v) = \top$.*

*For a set of rational functions (in one or more variables) $\mathcal{C}$, a **numerical condition** over $\mathcal{C}$ is given by an n-ary function $f \in \mathcal{C}$, n numerical state variables $v_1, \ldots, v_n \in V_N$ and a relational operator $\mathbf{relop} \in \{=, \neq, <, \leq, \geq, >\}$. It is written as $f(v_1, \ldots, v_n)\ \mathbf{relop}\ 0$, and it is satisfied by state (α, β) iff $f(\beta(v_1), \ldots, \beta(v_n))\ \mathbf{relop}\ 0$.*

*A **propositional effect** is given by a propositional variable $v \in V_P$ and a truth value $t \in \{\top, \bot\}$. It is written as $v \leftarrow t$ and v is called its **affected variable**.*

*For a set of rational functions (in one or more variables) $\mathcal{E}$, a **numerical effect** over $\mathcal{E}$ is given by an n-ary function $f \in \mathcal{E}$ and n numerical state variables $v_1, \ldots, v_n \in V_N$. It is written as $v_1 \leftarrow f(v_1, \ldots, v_n)$ and v_1 is its affected variable.*

*An **operator** over sets of rational functions $\mathcal{C}$ and $\mathcal{E}$ is a pair (Pre, Eff), where Pre is a finite set of propositional conditions and numerical conditions over $\mathcal{C}$ and Eff is a finite set of propositional effects and numerical effects over $\mathcal{E}$ such that different effects have different affected variables.*

This is all we need to define planning tasks:

Definition 3 ***Planning task***
*A **planning task** over a planning formalism $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$ is a 5-tuple $T = (V_P, V_N, Init, Goal, Ops)$, such that $Init$ is a state, $Goal$ is a finite set of conditions over $\mathcal{G}$, and Ops is a finite set of operators over $\mathcal{P}$ and $\mathcal{E}$, with V_P and V_N being the underlying state variables (propositional and numerical, respectively).*

*$Init$ is called the **initial state**, $Goal$ the set of **goal conditions**, and Ops the **operator set** of T.*

The next definition captures the semantics of applying an operator to a state.

Definition 4 ***State transition graph***
*Let $T = (V_P, V_N, Init, Goal, Ops)$ be a planning task over a planning formalism $(\mathcal{G}, \mathcal{P}, \mathcal{E})$, and let S denote the set of states of T. The **state transition graph** of T is defined as the digraph with vertex set S containing the arc $((\alpha, \beta), (\alpha', \beta'))$ iff there is an operator $(Pre, Eff) \in Ops$ such that:*

(α, β) satisfies all conditions in Pre.

For all propositional effects $v \leftarrow t \in Eff$: $\alpha'(v) = t$.

For all $v \in V_P$ that are not affected by any effect in Eff: $\alpha'(v) = \alpha(v)$.

For all numerical effects $v_1 \leftarrow f(v_1, \ldots, v_n) \in Eff$: $\beta'(v_1) = f(\beta(v_1), \ldots, \beta(v_n))$.

For all $v \in V_N$ that are not affected by any effect in Eff: $\beta'(v) = \beta(v)$.

We complete our formal definitions by specifying the family of decision problems we want to analyze.

Definition 5 PLANEX-$\mathcal{F}$
For a planning formalism $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, the decision problem PLANEX-$\mathcal{F}$ is defined as follows:
***Given:** A planning task T over $\mathcal{F}$.*
***Question:** In the state transition graph of T, is there a directed path from the initial state to some state (α, β) satisfying all goal conditions?*

To provide some examples, PLANEX-$(\emptyset, \emptyset, \emptyset)$ is plan existence for propositional STRIPS (while there *are* numerical state variables according to our definition, they are not referred to at all and hence can be ignored), and PLANEX-$(\emptyset, \{id\}, \{inc, dec\})$, where $id : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x$, $inc : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x + 1$, and $dec : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x - 1$, is the plan existence problem for a planning formalism where no numerical conditions can be evaluated as part of the goal test, preconditions can test numerical state variables against zero, and effects can increase or decrease numerical state variables by one.

A Hierarchy of Planning Formalisms

Having defined the framework, we now have to answer the question which planning formalisms we are interested in analyzing. It is evident that we have to pose some restrictions on the sets of functions that define a planning formalism. If, for example, we included functions that are not computable, such as the *busy beaver* function (Boolos & Jeffrey 1989), we could easily prove undecidability results without getting any real insight into the problem of planning with numerical state variables.

It seems natural to restrict ourselves to functions that can be expressed by only using the usual arithmetic operators $\{+, -, \cdot, /\}$, which is in fact the approach of PDDL2.1. For our purposes, the problem of that approach is that many functions that make use of the division operator are not total, because division by zero is not possible.

Partial functions are forbidden in our definition of a planning formalism for a reason: Consider an operator which updates the value of state variable v by setting $\beta'(v) = \beta(v)^2/\beta(v)$. While this looks like a no-op, it is in fact a *hidden precondition* for the operator, because this calculation can only be performed if $\beta(v) \neq 0$. We do not want operator effects to play the part of operator preconditions because this is not what they are meant for. Thus, we further restrict the class of functions we want to analyze by only allowing division *by constants* other than zero, rather than division by values of state variables. This means that the most general class of functions we will investigate for operator effects is $\mathbb{Q}[x_1, x_2, x_3, \ldots]$, the class of multi-variable polynomials over the rational numbers.

Although there is no immediate need to be equally restrictive with regard to the sets of functions allowed for goal conditions and operator preconditions, we will do so for a simple reason: A condition that uses division arbitrarily can easily be transformed into a set of polynomial preconditions by multiplying with the squares of all denominators[1] and adding preconditions stating that the denominators must be different from zero. For example, the precondition $2/(x-1)+1 > 0$ can be replaced by the two polynomial preconditions $2(x-1)+(x-1)^2 > 0$ and $x-1 \neq 0$.

We will see that even with these restrictions, the PLANEX problem is undecidable, and thus we will investigate more specialized planning formalisms to identify the boundary of decidability. Specifically, we will use the following classes of functions for numerical goal conditions and numerical preconditions[2]:

- $\mathcal{C}_\emptyset = \emptyset$: No numerical conditions.
- $\mathcal{C}_0 = \{x \mapsto x\}$: Compare with zero. Numerical conditions are of the type v **relop** 0.
- $\mathcal{C}_c = \{\, x \mapsto x - c \mid c \in \mathbb{Q} \,\}$: Compare with a constant. Numerical conditions are of the type $v - c$ **relop** 0, for which we will use the alternative notation v **relop** c.
- $\mathcal{C}_= = \{(x_1, x_2) \mapsto x_1 - x_2\}$: Compare with another numerical state variable. Numerical conditions are of the type $v_1 - v_2$ **relop** 0, for which we will use the alternative notation v_1 **relop** v_2.
- $\mathcal{C}_p = \mathbb{Q}[x]$: Compare a polynomial of a state variable with zero. Numerical conditions are of the type $p(v)$ **relop** 0, where p is a polynomial.
- $\mathcal{C}_{p+} = \mathbb{Q}[x_1, x_2, x_3, \ldots]$: Compare a polynomial of many state variables to zero. Numerical conditions are of the type $p(v_1, \ldots, v_n)$ **relop** 0, where p is a polynomial.

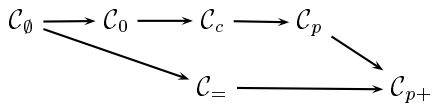

Figure 1: Containment for the function classes used in conditions. Arrows indicate subset relations.

For numerical effects, we will investigate the following classes of functions:

- $\mathcal{E}_\emptyset = \emptyset$: No numerical effects.
- $\mathcal{E}^{=c} = \{\, x \mapsto c \mid c \in \mathbb{Q} \,\}$: Assign a constant. Numerical effects are of the type $v \leftarrow c$.
- $\mathcal{E}_{+1} = \{x \mapsto x + 1\}$: Increase by one. Numerical effects are of the type $v \leftarrow v + 1$.
- $\mathcal{E}^{=c}_{+1} = \mathcal{E}^{=c} \cup \mathcal{E}_{+1}$. Numerical effects are either of the type $v \leftarrow c$ or of the type $v \leftarrow v + 1$.
- $\mathcal{E}_{\pm 1} = \{\, x \mapsto x + c \mid c \in \{-1, +1\} \,\}$: Increase or decrease by one. Numerical effects are either of the type $v \leftarrow v + 1$ or of the type $v \leftarrow v - 1$.
- $\mathcal{E}^{=c}_{\pm 1} = \mathcal{E}^{=c} \cup \mathcal{E}_{\pm 1}$. Numerical effects are either of the type $v \leftarrow c$, or of the type $v \leftarrow v+1$, or of the type $v \leftarrow v-1$.
- $\mathcal{E}_{+c} = \{\, x \mapsto x + c \mid c \in \mathbb{Q}_+ \,\}$: Add a positive constant. Numerical effects are of the type $v \leftarrow v + c$ for $c \in \mathbb{Q}_+$.
- $\mathcal{E}^{=c}_{+c} = \mathcal{E}^{=c} \cup \mathcal{E}_{+c}$. Numerical effects are either of the type $v \leftarrow c$ for $c \in \mathbb{Q}$ or of the type $v \leftarrow v + c$ for $c \in \mathbb{Q}_+$.
- $\mathcal{E}_{\pm c} = \{\, x \mapsto x + c \mid c \in \mathbb{Q} \,\}$: Add an arbitrary constant. Numerical effects are of the type $v \leftarrow v + c$.
- $\mathcal{E}^{=c}_{\pm c} = \mathcal{E}^{=c} \cup \mathcal{E}_{\pm c}$. Numerical effects are either of the type $v \leftarrow c$ or of the type $v \leftarrow v + c$.
- $\mathcal{E}_p = \mathbb{Q}[x]$: Assign a polynomial of the old value. Numerical effects are of the type $v \leftarrow p(v)$, where p is a polynomial.
- $\mathcal{E}_{p+} = \mathbb{Q}[x_1, x_2, x_3, \ldots]$: Assign a polynomial of the old values of multiple state variables. Numerical effects are of the type $v_1 \leftarrow p(v_1, \ldots, v_k)$, where p is a polynomial.

With six options each for goal conditions and operator preconditions and twelve options for operator effects, there is a total of 432 different planning formalisms that we can define using combinations of the above classes of functions (not all of them interesting, of course). The following result shows that they need not be considered in isolation:

[1] We use squares in order to be sure not to multiply with a negative number, which is important if the relational operator is from the set $\{<, \leq, \geq, >\}$.

[2] We write $x \mapsto f(x)$ for the function in $\mathbb{Q} \to \mathbb{Q}$ which maps x to $f(x)$, and $(x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n)$ for the function in $\mathbb{Q}^n \to \mathbb{Q}$ which maps $(x_1, \ldots, x_n)$ to $f(x_1, \ldots, x_n)$.

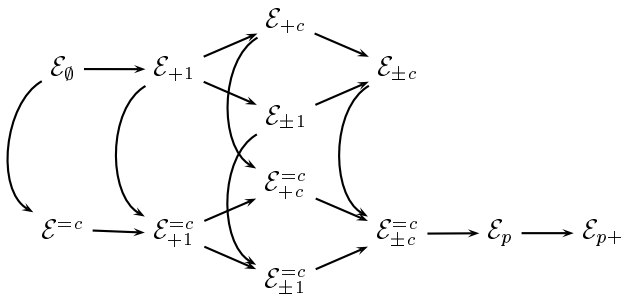

Figure 2: Containment for the function classes used in effects. Arrows indicate subset relations.

Proposition 6 ***Generalization/specialization***
For planning formalisms $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, $\mathcal{F}' = (\mathcal{G}', \mathcal{P}', \mathcal{E}')$ satisfying $\mathcal{G} \subseteq \mathcal{G}'$, $\mathcal{P} \subseteq \mathcal{P}'$, and $\mathcal{E} \subseteq \mathcal{E}'$, the following holds: PLANEX-$\mathcal{F} \leq$ PLANEX-$\mathcal{F}'$.

This is immediately obvious because if these containment conditions hold, the one decision problem is just a special case of the other one, and the identity function can be used as a membership-preserving mapping. This proposition makes it worthwhile to look at the containment graphs for the function classes we have defined. They are illustrated in Figure 1 for the conditions and Figure 2 for the effects.

Undecidability Results

We have now finished our preparations and can start proving results. In this section, we prove undecidability for five of the planning formalisms introduced. Together with Proposition 6, this allows us to conclude that for 258 of the 432 different variants of planning with numbers, complete algorithms for plan generation cannot be found. The remaining 174 variants will then be analyzed in the following section.

Our first result shows that general goals, with individual goal conditions referring to polynomials of multiple state variables, lead to undecidability even if no numerical operator preconditions and only a very weak type of numerical effects are present. The proof is based on the decision problem DIOPHANT$_{\mathbb{N}_0}$, which asks if a given Diophantine equation (polynomial in multiple variables) has a solution in the natural numbers. This problem is known to be undecidable (Cutland 1980).[3]

Theorem 7 PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_\emptyset, \mathcal{E}_{+1})$ ***is undecidable.***
Proof: Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial. The equation $p(x_1, \ldots, x_n) = 0$ has a solution in the natural numbers iff the planning task T described below is solvable:
$T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$, where
$V_P = \emptyset$,
$V_N = \{v_1, \ldots, v_n\}$,
$\mathit{Init} = (\emptyset, \{(v_1, 0), \ldots, (v_n, 0)\})$,
$\mathit{Goal} = \{p(v_1, \ldots, v_n) = 0\}$, and
$\mathit{Ops} = \{\ (\emptyset, \{v_i \leftarrow v_i + 1\}) \mid i \in \{1, \ldots, n\}\ \}$.

This is not hard to see: If $(m_1, \ldots, m_n)$ is a solution to p, then applying the operator affecting v_1 m_1 times, applying the operator affecting v_2 m_2 times, and so on, results in a plan satisfying the goal. On the other hand, the values of the state variables at the end of the execution of any plan that solves the task form a solution to the Diophantine equation.

It is evident that this transformation is computable and that the resulting planning task obeys the constraints of the planning formalism. Thus, DIOPHANT$_{\mathbb{N}_0} \leq$ PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_\emptyset, \mathcal{E}_{+1})$, proving the theorem. ∎

Note that the planning tasks constructed in the proof even obey some further constraints: There are no operator preconditions whatsoever, there is only one goal condition, and there is only one effect per operator. This emphasizes the fact that goal conditions referring to multiple numerical state variables are indeed a great challenge.

Corresponding results could be shown for multi-variable polynomials in the preconditions rather than the goal, or for multi-variable polynomials in the numerical effects and tests against 0 as the only valid goal conditions, with only a few adjustments to the proof. However, we do not provide these proofs since the following theorems already entail those undecidability results.

While the previous result is not too limiting – polynomials in multiple variables are quite powerful mathematical objects and thus it does not seem unreasonable to disallow them –, the following result uncovers more severe restrictions. But first we have to introduce the decision problem that the reduction is based on.

Definition 8 01-MPCP
Given: *A modified PCP (MPCP) over the alphabet {0, 1}, i. e. non-empty words $a_1, \ldots, a_n, b_1, \ldots, b_n$ over {0, 1}.*
Question: *Is there a solution for the MPCP, i. e. a sequence of indices $i_1, \ldots, i_m$ such that the concatenation of the words $a_{i_1}, \ldots, a_{i_m}$ equals the concatenation of the words $b_{i_1}, \ldots, b_{i_m}$, and such that $i_1 = 1$?*

This problem is undecidable, even if n is fixed to be 7 (Matiyasevich & Senizerguez 1996), a special case we denote as 01-MPCP[7].

Theorem 9 PLANEX-$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_p)$ ***is undecidable.***
Proof: Let $((a_1, b_1), \ldots, (a_7, b_7))$ be a binary MPCP. We assume that a_1 and b_1 start with the same digit (otherwise the problem is trivially unsolvable and can be mapped to a trivially unsolvable planning task), and that this digit is 1 (otherwise 0 and 1 are flipped in all words). We write $\#w$ to denote the number that has the word $w \in \{0, 1\}^*$ as its binary notation, e. g. $\#110$ equals 6. The MPCP can be solved iff the planning task T described below has a solution:
$T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$, where
$V_P = \emptyset$,
$V_N = \{a, b\}$,
$\mathit{Init} = (\emptyset, \{(a, \#a_1), (b, \#b_1)\})$,
$\mathit{Goal} = \{a = b\}$, and

[3] The reference only gives a proof of undecidability for solutions in the *integers* rather than in the natural numbers, but it is easy to see that $p(x_1, \ldots, x_n) = 0$ has an integer solution if and only if one of the polynomials that can be obtained from p by substituting some of the variables x_i by $-x_i$ has a solution in the natural numbers, providing a Turing reduction.

$Ops = \{\ op_i \mid i \in \{1, \ldots, 7\}\ \}$, where
$op_i = (\emptyset, \{a \leftarrow 2^{|a_i|}a + \#a_i, b \leftarrow 2^{|b_i|}b + \#b_i\})$.

Again, it is easy to see that this transformation is a computable function and that it maps to a planning instance satisfying the requirements for the chosen planning formalism (note that $2^{|a_i|}$ and $2^{|b_i|}$ are *constants*, as a_i and b_i are *not* state variables).

To see that the mapping is solution-preserving, observe that after applying an operator sequence $op_{i_2}, \ldots, op_{i_m}$ for $i_j \in \{1, \ldots, 7\}$ to the initial state, the state variables a and b, written down as binary numbers, equal the concatenation of $a_{i_1}, \ldots, a_{i_m}$ and $b_{i_1}, \ldots, b_{i_m}$, respectively, for $i_1 = 1$.

Thus, if $i_1, \ldots, i_m$ is a solution to the MPCP, then applying $op_{i_2}, \ldots, op_{i_m}$, in sequence, to the initial state solves the planning task, and vice versa.

This shows 01-MPCP[7] $\leq$ PLANEX-$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_p)$, proving the theorem. ■

This result is complemented by the following:

Theorem 10 PLANEX-$(\mathcal{C}_0, \mathcal{C}_\emptyset, \mathcal{E}_p)$ ***is undecidable.***
Proof: In this case, we only need to slightly adjust the mapping from the previous proof to accommodate for the different type of goal. We only describe the changes to the previous mapping. First, we introduce two propositional variables, $Work$, with an initial value of $\top$, and $Finish$, with an initial value of $\bot$. The goal is adjusted to require a and b to both have values of 0, rather than just be equal.

We make $Work = \top$ a precondition of each of the operators op_i, $i \in \{1, \ldots, 7\}$, and introduce two new operators, $op_S = (\{Work = \top\}, \{Work \leftarrow \bot, Finish \leftarrow \top\})$ and $op_D = (\{Finish = \top\}, \{a \leftarrow a - 1, b \leftarrow b - 1\})$.

It is not hard to see that this is again a solution-preserving reduction. If the MPCP is solvable, then applying a sequence of operators over $\{op_1, \ldots, op_7\}$ leads to equal values for a and b. Then, op_S can be applied once and op_D a number of times until both values are 0, meeting the goal. On the other hand, it is easy to see that valid plans can be transformed into MPCP solutions in the same way as in the previous proof if the trailing instances of op_S and op_D are removed.

Thus, 01-MPCP[7] $\leq$ PLANEX-$(\mathcal{C}_0, \mathcal{C}_\emptyset, \mathcal{E}_p)$, proving the theorem. ■

Similar results hold for preconditions (rather than goal conditions) from $\mathcal{C}_=$ and $\mathcal{C}_0$, but again, those are entailed by results to come. Note from the two proofs that the results still hold if only polynomials of degree 1 are allowed in numerical effects. This basically shows that from the effect function sets in our hierarchy, everything that is beyond assignment, addition and subtraction of constants is too hard, even for the most simple types of pre- and goal conditions. In fact, the proof of Theorem 9 even shows undecidability for the very restricted case of no preconditions, only seven operators, and only two state variables, if polynomials of degree 1 are allowed for effects.

For the last two results in this section, we first have to define abacus programs.

Definition 11 ***Abacus program***
*An **abacus program** is a 5-tuple (V, L, l_0, l_H, P), where V and L are disjoint finite sets called **variables** and **labels**, respectively, $l_0 \in L$ is called the **initial label**, $l_H \in L$ is called the **halt label**, and $P : L \setminus \{l_H\} \rightarrow C_{V,L}$ is called the **program**. $C_{V,L}$ is defined as follows:*
$C_{V,L} = \{\ \textbf{INC}\ v; \rightarrow l \mid v \in V, l \in L\ \}$
$\quad \cup\ \{\ \textbf{DEC}\ v; \rightarrow l_>, l_= \mid v \in V, l_>, l_= \in L\ \}$

We cannot afford the space to formally introduce the semantics of abacus programs, so we hope that the following informal account will suffice.

The variables of the program correspond to registers of a virtual machine which can hold arbitrary natural numbers. They are initialized to zero. Initially, the current label is l_0. If at any time the current label is l_H, execution terminates. Otherwise, if the current label is l, and $P(l) = \textbf{INC}\ v; \rightarrow l'$, the value of variable v is increased by one and l' becomes the new current label. If $P(l) = \textbf{DEC}\ v; \rightarrow l_>, l_=$, the value of variable v is inspected. If it is greater than zero, it is decreased by one and $l_>$ becomes the current label. Otherwise, it is not changed and $l_=$ becomes the current label. The process is repeated with the new current label.

Abacus machines are as powerful as Turing Machines (Boolos & Jeffrey 1989), which implies that it is algorithmically impossible to decide whether execution of a given abacus program stops or not, i. e. the halting problem for abacus programs, HALT-ABACUS, is undecidable, following from a well-known theorem about Turing Machines (Boolos & Jeffrey 1989). This leads to the following result:

Theorem 12 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_0, \mathcal{E}_{\pm 1})$ ***is undecidable.***
Proof: Let (V, L, l_0, l_H, P) be an abacus program. Then the planning task T is defined as follows:
$T = (V_P, V_N, Init, Goal, Ops)$, $Init = (\alpha_I, \beta_I)$, where
$V_P = L$,
$V_N = V$,
$\alpha_I(l_0) = \top$; $\alpha_I(l) = \bot$ for all other $l \in L$,
$\beta_I(v) = 0$ for all $v \in V$,
$Goal = \{l_H = \top\}$, and
Ops contains the following operators:
For l, l', v such that $P(l) = \textbf{INC}\ v; \rightarrow l'$:
$(\{l = \top\}, \{l \leftarrow \bot, l' \leftarrow \top, v \leftarrow v + 1\})$.[4]
For l, $l_>$, $l_=$, v such that $P(l) = \textbf{DEC}\ v; \rightarrow l_>, l_=$:
$(\{l = \top, v > 0\}, \{l \leftarrow \bot, l_> \leftarrow \top, v \leftarrow v - 1\})$ and
$(\{l = \top, v = 0\}, \{l \leftarrow \bot, l_= \leftarrow \top\})$.

To understand the mapping, note that states of the planning task correspond to execution states of the abacus program, where propositional variable l is true iff l is the current label. It is easy to verify that plans for this task correspond to execution traces of the abacus program, and that the goal will be met iff the halt label ever becomes the current label, i. e. iff the abacus program halts.

Thus, HALT-ABACUS$\leq$ PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_0, \mathcal{E}_{\pm 1})$, proving the theorem. ■

We consider it interesting to point out that the values of the numerical state variables in a goal state correspond to

[4] If $l = l'$, the propositional effects are omitted. Similar comments apply to the other operators if $l = l_>$ or $l = l_=$, respectively.

the values of the abacus program variables upon termination. This close correspondence means that we could employ a planning formalism as a programming language, computing functions of the natural numbers. The previous proof shows that, viewed as a programming language, the formalism is Turing-complete (like abacus programs).

The last result in this section is a minor variation of the previous theorem.

Theorem 13 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_=, \mathcal{E}_{+1})$ ***is undecidable.***
Proof: We use the same reduction as in the proof of the preceding theorem, with two changes. First, for each numerical state variable $v \in V$, we introduce another numerical state variable v^-, also initialized to 0. Second, in the operators corresponding to decrement statements in the abacus program, the preconditions that compare v against zero are changed to compare v against v^-, and the effect that decreases v is changed to an effect that increases v^-.

Again, we can verify that plans correspond to abacus program execution traces. To understand this, observe that the value of the abacus program variable v in state (α, β) can be calculated as $\beta(v) - \beta(v^-)$. Thus, following the same reasoning as above, HALT-ABACUS$\leq$ PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_=, \mathcal{E}_{+1})$, proving the theorem. ∎

This completes our list of undecidability results. The following section will provide some good news.

Decidability Results

We begin the discussion of decidable planning formalisms in our numerical planning framework by mentioning two trivial results.

Theorem 14 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_\emptyset, \mathcal{E}_{p+})$ ***is decidable.***
Proof: Since the planning formalism does not feature numerical goal conditions or operator preconditions, all numerical state variables and effects can simply be ignored. The problem therefore reduces to plan existence for propositional STRIPS, which is known to be decidable, in fact **PSPACE**-complete (Bylander 1994). ∎

The preceding result is not of much relevance and was only included for sake of completeness. The next result is a bit more interesting, but still obvious.

Theorem 15 PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_{p+}, \mathcal{E}^{=c})$ ***is decidable.***
Proof: In this planning formalism, only constants are assigned to numerical state variables. Since there is only a finite set C of such constants in each planning task, no numerical state variable can take on more than $|C|+1$ different values, namely its initial value and the values from C. The set of states reachable from the initial state is thus finite and the reachable part of the state transition graph can be explicitly constructed and the problem then solved using standard graph search techniques. ∎

Having considered the trivial cases, we will now look at more interesting planning formalisms and provide some simplification results. To do so, we must first introduce a new term.

Definition 16 ***Scalable function sets***
A set of rational functions $\mathcal{F}$ is called ***scalable*** *if for each $q \in \mathbb{Q}_+$ and for each n-ary function $f \in \mathcal{F}$ there exists some function $f_{[q]} \in \mathcal{F}$ such that for all $x_1, \ldots, x_n \in \mathbb{Q}$, $\mathrm{sgn}(f(x_1, \ldots, x_n)) = \mathrm{sgn}(f_{[q]}(qx_1, \ldots, qx_n))$.*[5]

Some examples of scalable function sets:
$\mathcal{C}_p$: $(x \mapsto p(x))_{[q]} = (x \mapsto p(\frac{x}{q}))$
$\mathcal{C}_c$: $(x \mapsto x - c)_{[q]} = (x \mapsto x - qc)$
$\mathcal{C}_=$: $((x_1, x_2) \mapsto x_1 - x_2)_{[q]} = ((x_1, x_2) \mapsto x_1 - x_2)$

Our first simplification result shows that in certain cases we can restrict the domain of numerical state variables to the set of integers.

Algorithm 17 ***Domain simplification***
Let T be a planning task over $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ such that $\mathcal{G}$ and $\mathcal{P}$ are scalable and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}, \mathcal{E}_{\pm c}, \mathcal{E}^{=c}_{\pm c}\}$. Then the following "domain simplification" algorithm translates T into an equivalent task over $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ such that all numerical effects are of the type $v \leftarrow v + c$ or $v \leftarrow c$ for $c \in \mathbb{Z}$ and all initial values of numerical state variables are integers. Thus, for each reachable state (α, β), the domain of β is a subset of $\mathbb{Z}$.

Let $T = (V_P, V_N, Init, Goal, Ops)$, $Init = (\alpha_I, \beta_I)$.

Let $M = \{\, \beta_I(v) \mid v \in V_N \,\} \cup$
$\{\, c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N : v \leftarrow c \in Eff \,\} \cup$
$\{\, c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N : v \leftarrow v + c \in Eff \,\}$.

In words, M contains all the constants that appear in the initial state and operator effects.

Let $d \in \mathbb{N}_+$ be a common denominator of all $q \in M$.

Replace, for all $v \in V_N$, initial values $\beta_I(v)$ by $d\beta_I(v)$, effects of type $v \leftarrow c$ by $v \leftarrow dc$, effects of type $v \leftarrow v + c$ by $v \leftarrow v + dc$, and conditions $f(v_1, \ldots, v_n)$ $\mathbf{rel}$ 0 by $f_{[d]}(v_1, \ldots, v_n)$ $\mathbf{rel}$ 0.

This satisfies the requirements, which can be shown by verifying that the function which maps (α, β) to $(\alpha, d\beta)$ is a state isomorphism from the original to the modified planning task, i.e. initial and goal states are mapped to initial and goal states, and $((\alpha, \beta), (\alpha', \beta'))$ is an arc in the original state transition graph iff $((\alpha, d\beta), (\alpha', d\beta'))$ is an arc in the modified state transition graph.

Furthermore, the requirements of integrality and staying within $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ are met.

Our second simplification result, which is applicable under the same conditions as domain simplification, shows that in many cases numerical conditions using polynomials can be replaced by simple comparisons.

Theorem 18 ***Condition simplification***
Let $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ be a planning formalism such that $\mathcal{G}$ and $\mathcal{P}$ are scalable and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}, \mathcal{E}_{\pm c}, \mathcal{E}^{=c}_{\pm c}\}$. Then
PLANEX-$(\mathcal{G}, \mathcal{P}, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}', \mathcal{P}', \mathcal{E})$
for $\mathcal{G}' = (\mathcal{G} \setminus \mathcal{C}_p) \cup \mathcal{C}_c$ and $\mathcal{P}' = (\mathcal{P} \setminus \mathcal{C}_p) \cup \mathcal{C}_c$.
Proof: The following algorithm provides the required Turing reduction. First, apply domain simplification to the input

[5] sgn denotes the signum, or sign function. To be strict, we must also require that there is an algorithm that calculates a representation of $f_{[q]}$ from representations of f and q. However, as the most general class of functions we are concerned with in this paper are polynomials, we can safely assume that this is the case.

planning task. We can now assume all numerical state variables to only take on integral values, which allows to translate conditions of the type $p(v)$ $\mathbf{relop}$ 0 for polynomials p into disjunctions of simpler conditions of type v $\mathbf{relop}_i$ c_i in the following way.[6]

Using standard numerical algorithms, calculate integers l and u such that for all x_0 solving $p(x_0) = 0$, $l < x_0 < u$ (we ignore the trivial case $p(x) = 0$ for all x). Because of the intermediate value theorem and the assumption $\beta(v) \in \mathbb{Z}$ for all states (α, β), $p(v)$ $\mathbf{relop}$ 0 is equivalent to

$(v \leq l \wedge p(l)$ $\mathbf{relop}$ $0)$
$\vee$ $(v = l + 1 \wedge p(l + 1)$ $\mathbf{relop}$ $0)$
$\vee \ldots$
$\vee$ $(v = u - 1 \wedge p(u - 1)$ $\mathbf{relop}$ $0)$
$\vee$ $(v \geq u \wedge p(u)$ $\mathbf{relop}$ $0)$,

where the subexpressions $p(k)$ $\mathbf{relop}$ 0 can be evaluated statically to simplify the expression to a simple disjunction.

Once this has been done for all polynomial conditions, operators with disjunctive preconditions can be translated to sets of standard operators using well-known techniques (Nebel 1999), and the goal can be translated into a set of conjunctive goals analogously. We can then use the subroutine for PLANEX-$(\mathcal{G}', \mathcal{P}', \mathcal{E})$ to check if at least one of those conjunctive goals can be satisfied. If so, we succeed, otherwise we fail. ■

Our last simplification result shows that in many cases numerical operator preconditions can be compiled away completely if numerical state variables can only be assigned constants or increased.

Theorem 19 ***Precondition elimination***
Let $\mathcal{G}$ be a scalable function set and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}\}$. Then PLANEX-$(\mathcal{G}, \mathcal{C}_p, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}, \mathcal{C}_\emptyset, \mathcal{E})$.
Proof: Because the previous theorem implies the relationship PLANEX-$(\mathcal{G}, \mathcal{C}_p, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}, \mathcal{C}_c, \mathcal{E})$, we only need to eliminate operator preconditions that compare to constants. First, we apply domain simplification, obtaining a task $T = (V_P, V_N, Init, Goal, Ops)$ over $(\mathcal{G}, \mathcal{C}_c, \mathcal{E})$, with $Init = (\alpha_I, \beta_I)$. Assume $V_N \neq \emptyset$ (otherwise, this is trivial).

Let $l = \min(\{\ \beta_I(v) \mid v \in V_N\ \} \cup$
$\{\ c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N : v \leftarrow c \in Eff\ \})$.

l is the least initial value of any numerical state variable, or the least value that can be assigned to a state variable, whichever is lower. Values of numerical state variables will never be below l.

Let $u = 1 + \max\{\ c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N$
$\exists \mathbf{relop} \in \{=, \neq, <, \leq, \geq, >\} : v\ \mathbf{relop}\ c \in Pre\ \}$
(assuming this set is non-empty, otherwise there is nothing to do). u is one greater than the highest number that is compared to in any numerical precondition. For evaluating numerical preconditions, values beyond u need not be distinguished. Our key idea is to keep track of numerical state variables having values between l and u by using new *propositional* variables $eq_{v,k}$ for each $v \in V_N$, $k \in \{l, \ldots, u\}$[7], where $eq_{v,k}$ is to be read as "the value of v is k" with the exception of $k = u$, where it means "the value of v is at least u".

The initial value of these new propositional variables is defined by $\alpha_I(eq_{v,k}) = \top \leftrightarrow \beta_I(v) = k$. Their meaning is maintained throughout operator applications by adjusting operator definitions as follows:

For operators having an effect $v \leftarrow c$, add the effects $eq_{v,c} \leftarrow \top$ and $eq_{v,k} \leftarrow \bot$ for each $k \in \{l, \ldots, u\} \setminus \{c\}$.

For operators having an effect $v \leftarrow v + c$, for each $old, new \in \{l, \ldots, u\}$ add a *conditional* propositional effect **IF** $eq_{v,old} = \top$ **THEN** $eq_{v,new} \leftarrow t$, where $t = \top$ if the equation $\min(old + c, u) = new$ is true, and $t = \bot$ otherwise.

Once this has been done, each numerical precondition v $\mathbf{relop}$ c can be replaced by the following disjunctive propositional precondition:
$\bigvee_{\{k \in \{l,\ldots,u\}\ \mid\ k\ \mathbf{relop}\ c\ \}} eq_{v,k} = \top$.

Finally, disjunctive preconditions and conditional effects can be compiled away (Nebel 1999). ■

Now that we have provided a number of simplification results, it is time we presented the main result of this section. In the following, let $T = (V_P, V_N, Init, Goal, Ops)$ be a planning task over the formalism $\mathcal{F} = (\mathcal{C}_c \cup \mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}^{=c}_{\pm c})$. Note that there are no numerical preconditions in this formalism.

Definition 20 ***Propositional states***
For states (α, β), α is called a ***propositional state****.*

For propositional states α and α', an operator sequence is called a ***propositionally acyclic path from*** *α* ***to*** *α' if there is a corresponding path in the state transition graph from (α, β) to (α', β') (for some β, β') such that all nodes in the path have different propositional states.*

For propositional states α, a non-empty operator sequence is called a ***propositional cycle in*** *α if it corresponds to a path in the state transition graph from (α, β) to (α, β') such that all nodes in the path except the start and end node have different propositional states.*

The planning task T has $2^{|V_P|}$ propositional states, a finite number. Without numerical preconditions, knowing the propositional state is sufficient for deciding if an operator sequence is applicable in a given state, and the values of the numerical state variables are only important for checking goal conditions. Thus, we will also talk about an operator or sequence of operators being *applicable in a propositional state*.

Note that the sets of propositionally acyclic paths and propositional cycles can be computed by analyzing the (finite) transition graph of the planning task obtained by deleting all numerical state variables, effects and goal conditions from T. Only operator sequences of length at most $2^{|V_P|}$ must be taken into account.

[6] Note that we need domain simplification: If x can take on any rational value, than conditions like $x^2 - 2 \geq 0$ cannot be translated into a finite number of simple comparisons of x to rational numbers.

[7] For simplicity of notation, we assume $l < u$, which can be enforced by increasing u if needed.

Definition 21 ***Additive and assigning effects***
For operators $o = (Pre, Eff)$ and numerical state variables v, the ***increase*** *of v by o is defined as follows:*
$incr(o, v) = c$ if $v \leftarrow v + c \in Eff$.
$incr(o, v) = 0$ if no effect in Eff affects v.
$incr(o, v)$ is undefined if $v \leftarrow c \in Eff$ for some $c \in \mathbb{Q}$.
In the formalism $\mathcal{F}$, exactly one of those conditions must be true for each o and v. If the third holds, we say that o has an ***assigning effect****, otherwise an* ***additive effect*** *on v.*

For calculating the value of a numerical state variable v at the end of plan execution, we can ignore all plan steps that precede an operator with an assigning effect on v. We say that v becomes *active* in that plan after the last assignment to it, or with the first planning step, if it is never assigned a new value. Plan steps that cause some state variable to become active are called *activating steps*. The sequence of activating steps is called the *activation sequence*. It can comprise no more than $|V_N|$ operators, because each numerical state variable is activated at most once.

The key idea of our algorithm is to "guess" the activation sequence and flesh out the details of the plan by inserting subplans between steps of the activation sequence, and before the first and after the last activating step.

We will use a "guess" operation in the description of the algorithm a number of times. The algorithm makes a number of guesses that can be bounded by a computable function in the size of the instance, and in each case, the set of possible guesses is finite and can be generated systematically. Thus, the algorithm can be made deterministic by systematically exploring the space of possible guesses. If the task is solvable, there is a sequence of guesses that leads to success. If it is unsolvable, all possible guesses lead to failure.

Algorithm 22 ***Cycle counting, part 1***
First, guess a natural number $k \in \{0, \ldots, |V_N|\}$ and an activation sequence $aseq = (o_1, \ldots, o_k) \in Ops^k$. Verify that $aseq$ is a valid activation sequence by checking that each o_i has an assigning effect on some numerical state variable that is not assigned to by any operator o_j for $j > i$.

For each $v \in V_N$, the ***activation time*** *of v is defined as $atime(v) = \max\{\, i \in \{1, \ldots, k\} \mid o_i$ assigns to $v \,\}$, or 0 if this set is empty. The* ***activation value*** *$aval(v)$ of v is defined as the value that v is assigned by $o_{atime(v)}$, or the initial value of v, if $atime(v) = 0$.*

Next, guess a sequence $(\alpha_1, \ldots, \alpha_k)$ of propositional states such that each o_i is applicable in states with propositional part α_i. Calculate the propositional states α'_i that result from applying o_i in a state with propositional part α_i. We assume that each o_i is applied in a state with propositional part α_i. We define $\alpha'_0 = \alpha_I$, the propositional part of the initial state, and guess another propositional state α_{k+1} satisfying all propositional goal conditions, assuming that this is the propositional part of some reachable goal state.

What remains to be done is finding (possibly empty) operator sequences π_i for $i \in \{0, \ldots, k\}$ such that each π_i is applicable in the propositional state α'_i and results in propositional state α_{i+1} and such that $\pi_0 o_1 \pi_1 o_2 \ldots \pi_n$ also satisfies the numerical parts of the goal. We call such a sequence π_i the i-th *episode* of the plan.

Episodes cannot contain arbitrary operators: In episode i, operators that have assigning effects for any numerical state variable with activation time less than or equal to i are forbidden, because these state variables are active in that part of the plan.

If we make sure that these operators are ruled out, we can safely ignore assigning effects within an episode: All state variables that are assigned a value will be assigned a different value later, when they become active. So all relevant numerical effects are additive, which is important because addition is commutative and associative: We do not need to know the episode exactly, we only need the information which operators are executed and how often they are executed.

Each operator sequence linking propositional nodes α and α' can be partitioned, by iteratively removing propositional cycles, into a propositionally acyclic path and a number of propositional cycles. If for each episode we know its acyclic path and the number of times each propositional cycle in it is traversed, we can calculate the values of all numerical state variables in the goal.

The correct acyclic paths π_i^{ac} can be guessed, as there is only a finite number of candidates. We can also guess the sets C_i of propositional cycles that are traversed at least once in the episode, as there is only a finite number of possible choices here, too. As noted above, for both π_i^{ac} and C_i, we must restrict ourselves to sequences containing no operators with assigning effects that are illegal for the i-th episode.

Not every choice for C_i is feasible: We need to check for each chosen cycle that it touches a propositional node traversed by π_i^{ac}, or a propositional node on some other feasible cycle. This can be done with a fixpoint reachability test as described in the following part of the algorithm.

Algorithm 22 (continued) ***Cycle counting, part 2***
For all $i \in \{0, \ldots, k\}$, guess a propositionally acyclic path π_i^{ac} from α'_i to α_{i+1} and a set of propositional cycles C_i.

Verify that π_i^{ac} and the cycles in C_i contain no operators with assigning effects for numerical state variables with activation time i or less. Check that the choice for C_i is valid as follows:

First, label all cycles in C_i which pass through propositional nodes on π_i^{ac} as feasible. Then, label all cycles in C_i passing through propositional nodes on some feasible cycle as feasible. Iterate this step until a fixpoint is reached, and reject the choice of C_i if some cycle in C_i has not been labeled.

The only important piece of information which is still lacking is the *number of times* each cycle in C_i is executed in episode i. We cannot use a systematic enumeration technique here, because any positive integer would be a valid choice and thus, choices cannot be explored exhaustively. For this reason, we introduce a variable x_i^π for each cycle $\pi \in C_i$, representing the number of times this cycle is traversed in the i-th episode. The planning task is solvable if and only if there is a function which maps each such variable x_i^π to a positive integer such that the final values of the numerical state variables, which can be computed from the guessed information and the values of the x_i^π variables,

match the requirements of the goal.

Algorithm 22 (continued) ***Cycle counting, part 3***
Write down the following linear equations in the rational variables $goal_v$ for $v \in V_N$ and integer variables x_i^π for $i \in \{0, \ldots, k\}$ and $\pi \in C_i$:[8]

$$goal_v = aval(v) + \sum_{i \in \{atime(v)+1,\ldots,k\}} incr(o_i, v) + \sum_{i \in \{atime(v),\ldots,k\}} \sum_{o \in \pi_i^{ac}} incr(o, v) + \sum_{i \in \{atime(v),\ldots,k\}} \sum_{\pi \in C_i} \sum_{o \in \pi} incr(o, v) \cdot x_i^\pi$$

Additionally, write down the following formulas:
For each $i \in \{0, \ldots, k\}$, $\pi \in C_i$: $x_i^\pi \geq 1$.
For goal conditions v_1 $\mathbf{relop}$ v_2: $goal_{v_1}$ $\mathbf{relop}$ $goal_{v_2}$.
For goal conditions v $\mathbf{relop}$ c: $goal_v$ $\mathbf{relop}$ c.
Taking these together, we obtain a mixed integer program in the variables $goal_v$ and x_i^π that has a solution if and only if the planning task is solvable. A mixed integer program solver can be used to generate such a solution or prove that none exists (Bixby et al. *2000).*

Putting this algorithm and the preceding results together, we obtain the following corollary.

Corollary 23 ***Some decidability results***
PLANEX *is decidable for these planning formalisms:*
$(\mathcal{C}_p, \mathcal{C}_\emptyset, \mathcal{E}_{\pm c}^{=c})$ *(Theorem 18 and Algorithm 22)*
$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_{\pm c}^{=c})$ *(Algorithm 22)*
$(\mathcal{C}_p, \mathcal{C}_p, \mathcal{E}_{\pm c}^{=c})$ *(Theorem 19, Theorem 18 and Algorithm 22)*
$(\mathcal{C}_=, \mathcal{C}_p, \mathcal{E}_{+c}^{=c})$ *(Theorem 19 and Algorithm 22)*

Combining these four results with the two trivial results stated at the beginning of the section and the generalization/specialization relationships in our framework, we see that plan existence is decidable for all planning formalisms that were not covered in the previous section. This completes our analysis.

Discussion

Let us summarize the results of our analysis. We have defined and investigated a family of decision problems related to planning with numerical state variables. Many of these were undecidable, due to the fact that the state space is infinite. The results are repeated in Figure 3.

Type of effects	Decidability status
$\mathcal{E}_\emptyset, \mathcal{E}^{=c}$	Always decidable
$\mathcal{E}_{+1}, \mathcal{E}_{+c}, \mathcal{E}_{+1}^{=c}, \mathcal{E}_{+c}^{=c}$	Decidable iff $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} \notin \{\mathcal{C}_=, \mathcal{C}_{p+}\}$
$\mathcal{E}_{\pm 1}, \mathcal{E}_{\pm c}, \mathcal{E}_{\pm 1}^{=c}, \mathcal{E}_{\pm c}^{=c}$	Decidable iff $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} = \mathcal{C}_\emptyset$
$\mathcal{E}_p, \mathcal{E}_{p+}$	Decidable iff $\mathcal{G} = \mathcal{P} = \mathcal{C}_\emptyset$

Figure 3: Results for the variants of PLANEX-$(\mathcal{G}, \mathcal{P}, \mathcal{E})$ in the hierarchy.

[8] We use the notation $o \in \pi$ to traverse the operators in an operator sequence. This is *not* supposed to mean that π is viewed as a set: Operators that appear multiple times in π *must* be considered multiple times.

Some of the undecidability results, as mentioned before, even apply to very restricted special cases, such as a constant number of operators, no operator preconditions, no propositional state variables, and only two numerical state variables. Additionally, as can be verified from the proofs provided, all undecidability results still hold if the set of relational operators is limited to $\{=, \neq\}$ (rather than the full set $\{=, \neq, <, \leq, \geq, >\}$).

Our classification approach for planning formalisms is based on the idea of restricting the type of calculations that can be performed by the planner. There are other possible restrictions to facilitate planning. One of them is to assign lower and upper bounds to each state variable. Operators that try to set the value of a numerical state variable to some number which is beyond these bounds can either be disallowed, or increases and decreases can be "capped". If in addition to this, differences in the values of state variables cannot be arbitrarily small (e. g., if the values must be integers), this immediately leads to a finite state space and hence decidability.[9]

On a more practical note, what is the impact of undecidability? Even the most general decision problem in our framework has the property of *semi-decidability*, because it is possible to systematically enumerate all operator sequences, say in lexicographical order, and check for each of them whether it solves the task or not. Such an algorithm would succeed in generating a plan for all solvable tasks. For tasks that do not have a solution, however, it would not terminate, and the undecidability results we have proved show that there is no algorithm that can possibly recognize *all* unsolvable planning tasks.

However, even without numerical state variables, many planning systems do not make an effort to always detect unsatisfiable goals. This has practical reasons, one of them being that this kind of result is usually hard to find without completely exploring the state space. Systematic planning systems, like **Graphplan** (Blum & Furst 1997) or the symbolic breadth-first exploration mode of **Mips** (Edelkamp & Helmert 2001), can detect unsolvable tasks with simple termination criteria, but sacrifice efficiency in doing so, compared to local search approaches that are by design better suited for finding plans than for proving their non-existence.

We do not think that there is a general answer to the question how important it is to be able to reliably detect failure in planning, but it is evident that the decidability status of the underlying decision problem is of high relevance to that discussion: If undecidable formalisms are employed, not being able to reliably detect unsolvable tasks is a necessary property of *any* planning algorithm.

Related Work

We are not aware of any work in the AI planning literature that is directly concerned with the decidability issues that arise when numbers are introduced into a planning for-

[9] In our framework, bounds lead to decidability of PLANEX for all numerical effect function sets except $\mathcal{E}_p$ and $\mathcal{E}_{p+}$. For $\mathcal{E}_p$ and $\mathcal{E}_{p+}$, bounds do not make a difference. Unfortunately, we do not have the space to prove this here.

malism. Some interesting decidability and undecidability results for general cases of STRIPS-style planning can be found, e. g. in work by Erol et al. (1995), covering problems that arise when function symbols and/or an infinite number of constant symbols are introduced into the formalism (which would correspond to an infinite number of propositional variables in the terminology of this paper). Other related problems, such as reachability for hybrid automata, are investigated in the model checking literature.

Researchers such as Bylander (1994) or Bäckström and Nebel (1995) have studied hierarchies of planning formalisms similar to the ones in our paper with regard to complexity. In this context, work on compilation schemes for translating between different planning formalisms should also be mentioned (Nebel 1999).

As for algorithms for planning with numerical state variables, a multitude of approaches have been proposed. Only recently, Do and Kambhampati have presented a planning system using an action representation which they describe as "influenced by the PDDL+ language proposal", and which is at least as general as the most general planning formalism considered in this paper (2001).

Haslum and Geffner have recently presented an optimal planning system capable of dealing with some of the features we have investigated in this paper (2001). Apart from their model of time and concurrency, which has no counterpart in our formalism, their planning formalism most closely resembles $(\mathcal{C}_\emptyset, \mathcal{C}_c, \mathcal{E}_{\pm c})$, which is undecidable. However, they say that they have implemented the algorithm "with the restriction that consumable resources are monotonically decreasing", which relates to the decidable $(\mathcal{C}_\emptyset, \mathcal{C}_c, \mathcal{E}_{+c})$.

Outlook

As is usually the case in research, in the process of answering a question we also leave some open ends, issues that might or even should be addressed in the future. In this paper, we have only cared about decidability – can the decision problem be solved? We were able to say "yes" in a few non-trivial cases. For these, a natural next question would be "How efficiently can we solve it?", i. e. computational complexity should be analyzed.

We also provided some translations between different planning formalisms, or translations to some "normal form" (e. g., only using integers) within the same formalism. This is an area that could be expanded, proving domain compilation results like the ones obtained for propositional planning by Nebel (1999). Here, we only required our translations to be computable, but for practical applications in planning systems, a polynomial translation is much more useful.

When more PDDL2.1 planning domains involving numbers become available, it will be interesting to investigate to what extent – and how easily – they can be represented in the decidable planning formalisms we have studied.

Finally, recalling our introductory comments, the motivation for conducting this analysis was the advent of the PDDL2.1 planning formalism. As we mentioned in the beginning, PDDL2.1 is not just planning with numbers, and although we gave good reasons why numerical state variables were foremost on our agenda, there are some interesting questions around the new features of PDDL2.1 in the area of concurrent planning that have not been covered yet.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bixby, R.; Fenelon, M.; Gu, Z.; Rothberg, E.; and Wunderling, R. 2000. MIP: Theory and practice – Closing the gap. In Powell, M. J. D., and Scholtes, S., eds., *System Modelling and Optimization: Methods, Theory and Applications*, volume 174 of *International Federation for Information Processing*. Kluwer Academic Press. 19–49.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):281–300.

Boolos, G. S., and Jeffrey, R. C. 1989. *Computability and Logic*. Cambridge University Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Cesta, A., and Borrajo, D., eds. 2001. *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*.

Cutland, N. J. 1980. *Computability — An Introduction to Recursive Function Theory*. Cambridge University Press.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In Cesta and Borrajo (2001), 109–120.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

Edelkamp, S., and Helmert, M. 2001. The model checking integrated planning system (MIPS). *AI Magazine* 22(3):67–71.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* 76(1–2):65–88.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Available at http://www.dur.ac.uk/d.p.long/competition.html.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta and Borrajo (2001), 121–132.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Matiyasevich, Y., and Senizerguez, G. 1996. Decision problems for semi-Thue systems with a few rules. In *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science (LICS '96)*, 523–531.

Nebel, B. 1999. What is the expressive power of disjunctive preconditions? In Fox, M., and Biundo, S., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 294–307. New York: Springer-Verlag.

Local Search Topology in Planning Benchmarks: A Theoretical Analysis

Jörg Hoffmann
Institute for Computer Science
Albert Ludwigs University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany
hoffmann@informatik.uni-freiburg.de

Abstract

Many state-of-the-art heuristic planners derive their heuristic function by relaxing the planning task at hand, where the relaxation is to assume that all delete lists are empty. The success of such planners on many of the current benchmarks suggests that in those task's state spaces relaxed goal distances yield a heuristic function of high quality. Recent work has revealed empirical evidence confirming this intuition, stating several hypotheses about the local search topology of the current benchmarks, concerning the non-existence of dead ends and of local minima, as well as a limited maximal distance to exits on benches.

Investigating a large range of planning domains, we prove that the above hypotheses do in fact hold true for the majority of the current benchmarks. This explains the recent success of heuristic planners. Specifically, it follows that FF's search algorithm, using an idealized heuristic function, is polynomial in (at least) eight commonly used benchmark domains. Our proof methods shed light on what the structural reasons are behind the topological phenomena, giving hints on how these phenomena might be automatically recognizable.

Introduction

In the last three years, planning systems based on the idea of heuristic search have been very successful. At the AIPS-1998 planning systems competition, HSP1 compared well with the other systems (McDermott 2000), and at the AIPS-2000 competition, out of five prize-winning fully automatic planners, FF and HSP2 were based on heuristic search, while another two, Mips and STAN, were hybrids that incorporated, amongst other things, heuristic search (Bacchus 2001).

Interestingly, four of these five planners use the same base approach for deriving their heuristic functions: they relax the planning task description by ignoring all delete lists, and estimate, to each search state, the length of an optimal relaxed solution to that state. This idea has first been proposed by Bonet et al. (1997). The length of an optimal relaxed solution would yield an admissible heuristic. However, as was proven by Bylander (1994), computing the optimal relaxed solution length is still NP-hard. Therefore, Bonet et al. introduced a technique for approximating optimal relaxed solution length, which they use in both versions of HSP (Bonet & Geffner 2001). The heuristic engines in FF (Hoffmann & Nebel 2001) and Mips (Edelkamp & Helmert 2001) use different approximation techniques.

Three of the above planners, HSP1, FF, and Mips, use their heuristic estimates in variations of local search algorithms, where the search space to a task is the state space, i.e., the space of all states that are reachable from the initial state. Now, the behavior of local search depends crucially on the topology of the search space (as has been studied in the SAT community, for example by Frank et al. (1997)). Thus, the success of these heuristic planners on many planning tasks suggests that in those task's state spaces relaxed goal distances yield a heuristic function of high quality. Recent work has revealed empirical evidence confirming this intuition. By computing the optimal relaxed solution length to reachable states in small planning tasks from the competition domains, and measuring parameters of the resulting local search topology, the following was found (Hoffmann 2001b). In the majority of the domains, the investigated instances did not contain any dead ends (states from which the goal is unreachable), and neither did they contain any local minima (regions of the state space where all neighbours look worse). Sometimes all instances in a domain had the same constant maximal exit distance (roughly, the maximal distance to a state with better evaluation). It was hypothesized that these observations on the *small* example instances carry over directly to *all* instances in the respective domains (Hoffmann 2001b).

In the presented investigation, we consider 20 often used planning benchmark domains, including all competition examples. We prove that the majority of these domains do in fact have the aforementioned topological properties, confirming all of the above hypotheses except one. The results give a strong argument for interpreting the recent success of heuristic planners as utilizing the topology of the benchmarks, as was suggested by the previous empirical work (Hoffmann 2001b). Specifically, it follows that FF's search algorithm is a polynomial solving mechanism in eight of the domains, under the idealizing assumption that its heuristic identifies the optimal relaxed distances. What's more, our proof methods shed light on which structural properties—at the level of the planning task's definition—are responsible for the topological phenomena. As most of the structural properties we identify are of a syntactical nature, this knowl-

edge gives hints as to how the topological phenomena might be automatically recognizable.

The full details of the investigation form a long article that is available as a technical report (Hoffmann 2001a). Here, we summarize the definitions, and identify the key ideas in the form of proof sketches. The paper is organized as follows. The next section gives the background. We then include a section presenting the key lemmata underlying our proofs; the three sections after that prove the topological phenomena concerning dead ends, local minima, and maximal exit distance, respectively. Afterwards, one section gives the overall picture that our results determine, before we finish the paper by concluding and pointing to future research directions.

Background

Background is necessary on the planning framework, the investigated domains, local search topology, and the previous empirical work.

Planning Framework

To enable theoretical proofs to properties of planning *domains* rather than single tasks, we have developed a formal framework for STRIPS and ADL domains, formalizing in a straightforward manner the way how domains are usually handled in the community. The only other work we know of that uses a formal notion of planning domains is recent work by Malte Helmert (2001). There, the semantics of different transportation domains are formalized in order to prove their computational complexity. In difference to that, our definitions are strictly syntax-oriented—after all, the heuristic function we consider is extracted directly from the syntax of the planning task. We only summarize the rather lengthy definitions here, and refer the reader to our technical report (Hoffmann 2001a) for details.

A planning domain is defined in terms of a set of *predicate symbols*, a set of *operators*, and a set of *instances*. All logical constructs in the domain are based on the set of predicate symbols. The operators are (k-ary, where k is the number of operator parameters) functions from the set of all objects into the set of all STRIPS or ADL actions. A *STRIPS action* a is the usual triple $(pre(a), add(a), del(a))$ of fact sets; an *ADL action* consists of a first order logical formula without free variables as precondition, and a set of effects of the form (con, add, del) where con can again be a formula without free variables. An instance of a domain is defined in terms of a set of objects, an *initial state*, and a *goal condition*. The initial state is a set of facts, and the goal condition can be an arbitrary formula without free variables. An instance together with the respective operators constitutes a propositional planning task (A, I, G) where the action set A is the result of applying the operators to the objects, and the initial state I and goal condition G are those of the instance. We identify instances with the respective propositional tasks. The *result* $Result(s, a)$ of applying a STRIPS or ADL action a to a state s is defined in the usual manner: the result is defined only if the precondition is true in s; in that case, the action's add effects are made true and the delete effects are made false—for an ADL action, only those effects are applied that have their condition fulfilled in s. A *plan, or solution, for a task* (A, I, G) is a sequence of actions $P \in A^*$ that, when successively applied to I, yields a goal state, i.e., a state that fulfills G. P is *optimal* if there is no shorter plan for (A, I, G).

We investigate topological properties that arise when using the optimal relaxed solution length as a heuristic function. We name that function h^+. It is formally defined as follows. For any state s that is reachable in a propositional planning task (A, I, G), the *relaxed task to* s is (A^+, s, G): the task defined by the initial state s, the original goal condition, and the original action set except that all delete lists are empty. Then, $h^+(s)$ is the length of an optimal plan for (A^+, s, G) or $h^+(s) = \infty$ if there is no such plan. We will frequently make use of the following abbreviations: if a sequence of actions P^+ is a plan for (A^+, s, G), then we also say that P^+ is a *relaxed plan for* (A, s, G), or short a *relaxed plan for* s. [1] To give an example for a relaxed plan, consider the *Gripper* domain, as it was used in the AIPS-1998 competition. A real solution picks up two balls in room A, moves to room B, drops the balls, moves back, and does the same again until all balls have been transported. A relaxed plan simply picks up all balls with the same hand—the *free* predicate for the hand is not deleted—moves to room B, and drops all balls. While this might seem very simplistic, we will see, in fact prove, that it yields a high-quality heuristic function in a lot of benchmark domains.

Investigated Domains

We investigate the properties of 20 different STRIPS and ADL benchmark domains, including all 13 domains that have been used in the AIPS-1998 and AIPS-2000 planning system competitions. The competition domains are *Assembly*, *Blocksworld-arm*, *Freecell*, *Grid*, *Gripper*, *Logistics*, *Miconic-ADL*, *Miconic-SIMPLE*, *Miconic-STRIPS*, *Movie*, *Mprime*, *Mystery*, and *Schedule*. We assume that the reader is familiar with these domains, and do not describe them here. Descriptions can be looked up in the articles on the competitions (McDermott 2000; Bacchus 2001), and details are in our technical report (Hoffmann 2001a). Apart from the 13 competition domains, we investigate 7 more benchmark domains often used in the literature. These domains can be briefly described as follows.

1. *Blocksworld-no-arm*: unlike in the competition version, this encoding does not use an explicit robot arm; instead, blocks are moved around directly by operators moving them from one block to another block, or from the table to a block, or from a block to the table.
2. *Briefcaseworld*: transportation domain using conditional effects; objects can be put into or taken out of the (single)

[1] Ignoring the delete lists simplifies a task only if all formulae are negation free. In STRIPS, this is the case by definition. In general, for a fixed domain, any task can be polynomially transformed to have that property: compute the negation normal form to all formulae (negations only in front of atoms), then introduce for each negated atom $\neg B$ a new atom not-B and make sure it is true in a state iff B is false (Gazen & Knoblock 1997).

briefcase, and a move operator, which can be applied between any two locations, moves all objects along that are currently inside.

3. *Ferry*: also a transportation domain, with operators that board a car onto the (single) ferry, or debark a car from it; the ferry can only transport one car at a time, and a sail operator can be applied to move the ferry between any two locations.
4. *Fridge*: for a number of fridges, the broken compressors must be replaced. This involves un-fastening a number of screws that hold the compressors, removing the old compressors and attaching the new ones, and fastening all screws again.
5. *Hanoi*: encoding of the classical Towers of Hanoi problem, using a move(x, y, z) operator to move a disc x from a disc y to a disc z (the pegs are encoded as discs that can not be moved).
6. *Simple-Tsp*: a trivial version of the TSP problem, where the goal is that all locations have been visited, and a move operator can be applied between any two locations (all action costs being equal, as usual in STRIPS).
7. *Tyreworld*: a number of flat tyres must be replaced, which involves inflating the spare tyres, loosening the nuts, and in turn jacking up the hubs, undoing the nuts, removing the flat tyre and putting on the spare one, doing up the nuts, and jacking down the hub again; finally, all nuts must be tightened, and the tools must be put away into the boot.

For formally defining a domain, one must amongst other things decide what exactly the instances are. For almost none of the investigated domains is there such a definition in the literature. The obvious approach we have taken is to abstract from the known example suits. Full details for all domains are given in our technical report (Hoffmann 2001a).

Local Search Topology

We now define a number of topological phenomena that are relevant for local search. The definitions are summarizations of what was given in the previous empirical work (Hoffmann 2001b), slightly simplified to improve readability; details are in the technical report (Hoffmann 2001a). A propositional planning task is associated with its *state space* (S, E), which is a graph structure where S are all states that are reachable from the initial state, and E is the set of all pairs $(s, s') \in S \times S$ of states where there is an action that leads to s' when executed in s. The *goal distance* $gd(s)$ for a state $s \in S$ is the length of a shortest path in (S, E) from s to a goal state, or $gd(s) = \infty$ if there is no such path. In the latter case, s is a *dead end*.

When there are single-directed state transitions, there can be dead ends. A dead end s is *recognized* if $h^+(s) = \infty$, and *unrecognized* otherwise. To explain that terminology, note that $h^+(s) = \infty \Rightarrow gd(s) = \infty$: if a task can not be solved even when ignoring the delete lists, then the task is unsolvable. With respect to dead ends, any state space falls into one of the following four classes: the state space is

1. *undirected*, if $\forall (s, s') \in E : (s', s) \in E$,
2. *harmless*, if $\exists (s, s') \in E : (s', s) \notin E$, and $\forall s \in S : gd(s) < \infty$,
3. *recognized*, if $\exists s \in S : gd(s) = \infty$, and $\forall s \in S : gd(s) = \infty \Rightarrow h^+(s) = \infty$,
4. *unrecognized*, if $\exists s \in S : gd(s) = \infty \wedge h^+(s) < \infty$.

In the first class, there can be no dead ends because everything can be undone; in the second class, some things can not be undone, but those single-directed state transitions do not do any harm; in the third class, there are dead end states but all of them are recognized by the heuristic function. The only critical case for local search is class four, where a local search algorithm can run into an unrecognized dead end, and be trapped.

A different way of getting trapped is when local search ends up in a region of the state space where all neighbors look worse from the point of view of the heuristic function, i.e., when search encounters a local minimum: with all neighbors looking worse, it is not clear in which direction search should proceed. The formal definition of local minima, and of benches below, follows the definitions of Frank et al. (1997) for SAT problems; in difference to the *undirected* search spaces Frank et al. consider, we need to take care of single-directed state transitions. The adapted definitions are the following. A *flat path* is a path in (S, E) on which the heuristic value does not change. A *plateau of level* l is a set of states that have the same heuristic value l, and that form a strongly connected component in (S, E). An *exit of a plateau* is a state s that can be reached from the plateau on a flat path, and that has a better evaluated neighbor, i.e., $(s, s') \in E$ with $h^+(s') < h^+(s)$. A *local minimum* is a plateau of level $0 < l < \infty$ that has no exits. Note that we allow exits to not lie on the plateau itself, namely when the flat path leaves the plateau (which can happen if there is a single-directed state transition to a state with the same h^+ value); the main characteristic of local minima is that, starting from them, one must temporarily increase the heuristic value in order to improve it.

Finally, local search can get lost on large flat regions of the state space, usually referred to as benches (Frank, Cheeseman, & Stutz 1997). These are regions from which the heuristic value can be improved without temporarily increasing it, i.e., plateaus with exits. The hard bit for local search is to find the exits. The difficulty of doing this can be assessed by a variety of parameters like the size of the bench, or the exit percentage; in the previous empirical work (Hoffmann 2001b), the so-called maximal exit distance was measured. This parameter is especially relevant for FF's search algorithm, as will be explained in the next subsection. The definition is as follows. The *exit distance* of any state is the length of a shortest flat path connecting the state to an exit, or ∞ if there is no such path. The *maximal exit distance* in a state space is the maximum over the exit distances of all states s with $h^+(s) < \infty$, i.e., we ignore recognized dead ends—these can be skipped by search anyway. Note that states on local minima have infinite exit distance.

Previous Work: The Hypotheses

As described above, previous work has been done on empirically investigating topological properties in the competition domains, with respect to h^+ (Hoffmann 2001b); the approach being to compute h^+ for the states in example state spaces and measure parameters of the resulting local search topology. Because computing h^+ is NP-hard, the investigation was restricted to small instances. Amongst other things, the measured parameters were the dead-end class of the instances, the number of states on local minima, and the maximal exit distance. The observations are summarized in the table shown in Figure 1.

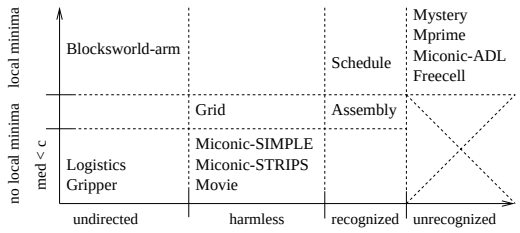

Figure 1: Overview of the empirical observations in small instances of the competition domains.

The x-axis in Figure 1 shows the different dead-end classes by increasing difficulty (for the domains in a dead-end class i, all investigated instances belong to a class $j \leq i$, and at least one instance belongs to class i). The y-axis combines local minima percentage with maximal exit distance: in the uppermost part of the table, the domains are shown were local minima were found; below that, domains are shown were there were no local minima at all in the small instances (unrecognized dead ends imply the existence of local minima, so that part of the table is crossed out); in the lower most part of the table domains are shown were all instances had the same constant maximal exit distance (remember that this parameter is infinity in the presence of local minima).

The table in Figure 1 was named the *planning domain taxonomy*, and it was hypothesized that these observations hold true for *all* instances in the respective domains: that all instances of a domain shown in dead-end class i are in a class $j \leq i$; that no instance of a domain shown in the lower parts of the table contains any local minima; that for a domain shown in the lowest part there is a constant c such that the maximal exit distance in all instances is at most c. The practical relevance of this is the following. The domains on the left bottom side of the taxonomy are "simple" for heuristic planners like HSP and FF because there the h^+ function, which those planners approximate, is a heuristic function of high quality. The majority of the competition domains seem to lie on the simple side. So, if these observations carry over to all instances in the respective domains, then the taxonomy can serve as an explanation for the good behavior of the aforementioned planners.

For domains in the lower most part of the taxonomy, FF's search algorithm is in fact polynomial in the sense that (assuming the heuristic function is given) it looks at polynomially many states before reaching the goal (Hoffmann 2001b). FF's search algorithm is the following.

$s := I$
while $h^+(s) \neq 0$ **do**
    do breadth first search for s', $h^+(s') < h^+(s)$
    $s := s'$
endwhile

Without local minima each iteration of this algorithm crosses a bench so breadth first search finds a better state at maximal depth $c + 1$; the branching factor is limited by the number of actions in the task; each iteration improves the heuristic value by at least one, so after at most $h^+(I)$ iterations a goal state is reached.

In the subsequent investigation, we verify how much truth there is in the hypotheses issued from observations on small examples. Going beyond that, we look at a larger number of domains, and determine exactly which part of the taxonomy they belong to. Obviously, we need only prove positive results (like the non-existence of local minima) for those domains were the empirical investigation did not reveal a negative example. As it turns out, all of the hypotheses are true except those about the *Assembly* domain. Like in the previous investigation, our main aim is to explain the good performance of local search planners, so we focus on *solvable* instances only—on unsolvable instances, local search is lost anyway.

A Theoretical Core

The following is the core of our theory, i.e., the key lemmata underlying our proofs in the single domains. This simplifies the subsequent proofs, and gives an insight into what the main structural reasons are behind the topological phenomena that we identify. The lemmata formulate sufficient criteria implying that (the state space of) a planning task has certain topological properties. Proofs for domains will proceed by applying the lemmata to arbitrary instances. For the sake of simplicity, we give our definitions only for STRIPS tasks. The proofs for ADL tasks are along the same lines of argumentation.

Dead Ends

We first identify sufficient criteria for a planning task containing no dead ends. Our starting point is a reformulated version of results published by Koehler and Hoffmann (2000). We need the notion of *inconsistency*, which is defined as follows. Two facts are inconsistent if there is no reachable state that contains both of them. Two sets of facts F and F' are inconsistent if each fact in F is inconsistent with at least one fact in F'.

If to each action there is an inverse action that undoes the action's effects, then the corresponding state space is undirected.

Definition 1 *Given a planning task (A, I, G). An action $a \in A$ is* invertible, *if*

1. *there is an action $\overline{a} \in A$ such that*
   (a) $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,
   (b) $add(\overline{a}) = del(a)$, *and*

(c) $del(\overline{a}) = add(a)$,

2. $add(a)$ *is inconsistent with* $pre(a)$, *and*
3. $del(a) \subseteq pre(a)$.

Lemma 1 *Given a planning task* (A, I, G). *If all actions* $a \in A$ *are invertible, then the state space to the task is undirected.*

Proof Sketch: For any state s and applicable action a, $\overline{a}$ is applicable in $Result(s, a)$ due to part 1(a) of Definition 1. Parts 2 and 3 of that definition make sure that a's effects do in fact appear, and parts 1(b) and (c) make sure that $\overline{a}$ undoes exactly those effects. ■

The next is a new criterion that is weaker and only implies the non-existence of dead ends. For an action not to lead into such a dead end, it is already sufficient if the inverse action re-achieves *at least* what has been deleted, and does not delete any facts that have been true before.

Definition 2 *Given a planning task* (A, I, G). *An action* $a \in A$ *is* at least invertible, *if there is an action* $\overline{a} \in A$ *such that*

1. $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,
2. $add(\overline{a}) \supseteq del(a)$, *and*
3. $del(\overline{a})$ *is inconsistent with* $pre(a)$.

Note that the previous Definition 1 is strictly stronger than Definition 2: if $del(\overline{a}) = add(a)$, and $add(a)$ is inconsistent with $pre(a)$, then, of course, $del(\overline{a})$ is inconsistent with $pre(a)$.

Another reason for an action not leading into a dead end is this. If the action must be applied at most once (because its add effects will remain true), and it deletes nothing but its own preconditions, then that action needs not be inverted.

Definition 3 *Given a planning task* (A, I, G). *An action* $a \in A$ *has* static add effects, *if*

$$add(a) \cap \bigcup_{a' \in A} del(a') = \emptyset$$

An action has irrelevant delete effects, *if*

$$del(a) \cap (G \cup \bigcup_{a \neq a' \in A} pre(a')) = \emptyset$$

If all actions in a task are either at least invertible or have static add- and irrelevant delete-effects, then the state space is at most harmless.

Lemma 2 *Given a solvable planning task* (A, I, G). *If it holds for all actions* $a \in A$ *that either*

1. a *is at least invertible, or*
2. a *has static add effects and irrelevant delete effects,*

then there are no dead ends in the state space to the task, i.e., $gd(s) < \infty$ *for all* $s \in S$.

Proof Sketch: For any state $s = Result(I, P)$ a plan can be constructed by inverting P (applying the respective inverse actions in the inverse order), and executing an arbitrary plan for (A, I, G) thereafter. In the first process, actions that are not (at least) invertible can be skipped: for those the second prerequisite holds true, so once they are applied their add effects remain true and their delete effects are no longer needed. In the second process, all actions are safely applicable except those not invertible ones that have been skipped in the first process. But for the same reasons as outlined above those actions need not be applied anyway. ■

The two properties handled so far, undirected and harmless state spaces, are properties of the planning tasks themselves, independent of the h^+ function. Different from that, the third dead end class, recognized dead ends, *does* depend on the heuristic function. We did, however, not find a general sufficient criterion for this case. Anyway, only two of our domains, *Schedule* and *Assembly*, belong to that class according to the hypotheses, and as it will turn out for *Assembly* the hypothesis is wrong.

Local Minima

We now identify a sufficient criterion for the non-existence of local minima under evaluation with h^+. As will be shown, the criterion can be directly applied to (the instances of) 6 of our 20 domains, and can be applied with slight modifications to 4 more domains. The criterion is based on actions that fulfill a weak notion of invertibility, and that are respected by the relaxation in the sense defined below.

When using a relaxed action to invert an action's effects, it is already enough if the inverse action re-achieves all facts that have been deleted—as the delete effects of the inverse action will be ignored anyway, there needs be no constraint about their form.

Definition 4 *Given a planning task* (A, I, G). *An action* $a \in A$ *is* at least relaxed invertible, *if there is an action* $\overline{a} \in A$ *such that*

1. $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,
2. *and* $add(\overline{a}) \supseteq del(a)$.

Note that the previous Definitions 1 and 2 are strictly stronger than Definition 4.

The following property is the key behind the non-existence of local minima in most of our domains: actions that are good for the real task are also good for the relaxed task.

Definition 5 *Given a solvable planning task* (A, I, G). *An action* $a \in A$ is respected by the relaxation *if, for any reachable state* s *such that* a *starts an optimal plan for* (A, s, G), *there is an optimal relaxed plan for* (A, s, G) *that also starts with* a.

As a simple example for an action that is respected by the relaxation, consider picking up a ball in *Gripper*: if that action starts an optimal plan, then the ball must be transported; any relaxed plan needs to transport the ball, and there is no other way of accomplishing this (except using the other hand, which does not yield a shorter solution). A similar argument applies to loading or unloading a truck in *Logistics*: the only way of transporting a package within its initial or destination city is by using the local truck, so *all* plans, real or relaxed, must use the respective action.

Lemma 3 *Given a solvable planning task (A, I, G), such that the state space does not contain unrecognized dead ends. If each action $a \in A$ either*

1. *is respected by the relaxation and at least relaxed invertible, or*
2. *has irrelevant delete effects,*

then there are no local minima in the state space under evaluation with h^+.

Proof Sketch: States s where $gd(s) = \infty$ have $h^+(s) = \infty$ by prerequisite and are therefore not on local minima. We show below that, from states s with $0 < gd(s) < \infty$, h^+ decreases monotonically on any optimal path to the goal. This finishes the argument: the goal state has a better h^+ value than s, and the path to it does not increase.

Say an action a starts an optimal plan in a state s. We identify, for the successor state $Result(s, a)$, a relaxed plan that has at most the same length as an optimal relaxed plan for s. If the first case of the prerequisite holds, then a starts an optimal relaxed plan P^+ for s. A relaxed plan for $Result(s, a)$ can be constructed by replacing a in P^+ with the inverse action $\overline{a}$: the inverse action is applicable in $Result(s, a)$, and it re-achieves all facts that a has deleted. In the second case of the prerequisite, if a has irrelevant delete effects, we distinguish two cases. Let P^+ be an optimal relaxed plan for s. First case, a is contained in P^+: then it can be removed from P^+ to form a relaxed plan for $Result(s, a)$, as a does not delete anything that is needed by other actions. Second case, a is not contained in P^+: then P^+ is still a relaxed plan for $Result(s, a)$ due to the same reason. ■

Planning tasks where there are unrecognized dead ends do contain local minima anyway (Hoffmann 2001b), so we must postulate that this is not the case; like when the task at hand is undirected or harmless.

Maximal Exit Distance

Concerning the maximal exit distance, we remark only the following simple property which underlies our proof technique.

Proposition 1 *Given a planning task (A, I, G), a reachable state s, and an action a that starts an optimal relaxed plan P^+ for (A, s, G). If removing a from P^+ yields a relaxed plan for $Result(s, a)$, then $h^+(Result(s, a)) < h^+(s)$, i.e., s is an exit state under h^+.*

The prerequisite holds, for example, if the action a is respected by the relaxation and has irrelevant delete effects.

In the following sections, we will focus on dead ends, local minima, and maximal exit distance in turn. For each of these three phenomena, we summarize our proofs for all domains in a single proof sketch. This improves readability, and makes it easier to see the common ideas behind the proofs.

Dead Ends

We first prove to which dead-end class our domains belong. Remember that we need only consider those domains where the empirical work did not reveal a negative example (like the unrecognized dead ends in *Freecell*, *Miconic-ADL*, *Mprime*, and *Mystery*). Most of the proofs are simple applications of the lemmata presented in the previous section.

Theorem 1 *The state space to any solvable planning task belonging to the*

1. *Blocksworld-arm, Blocksworld-no-arm, Briefcaseworld, Ferry, Fridge, Gripper, Hanoi, or Logistics domains is undirected,*
2. *Grid, Miconic-SIMPLE, Miconic-STRIPS, Movie, Simple-Tsp, or Tyreworld domains is harmless,*
3. *Schedule domain is recognized under evaluation with h^+.*

Proof Sketch: All actions in *Blocksworld-arm*, *Blocksworld-no-arm*, *Ferry*, *Gripper*, *Hanoi*, and *Logistics* instances are invertible in the sense of Definition 1, so we can apply Lemma 1 and are finished. In the *Briefcaseworld* and *Fridge* domains, while not strictly obeying the syntax of Definition 1, there is still always an action leading back to the state one started from.

In the *Movie*, *Simple-Tsp*, and *Tyreworld* domains, all actions are either at least invertible in the sense of Definition 2 or have irrelevant delete effects and static add effects in the sense of Definition 3, so Lemma 2 can be applied. In the *Grid*, *Miconic-SIMPLE*, and *Miconic-STRIPS* domains, while not strictly adhering to these definitions, similar arguments prove the non-existence of dead ends: in *Grid*, to all actions there is an inverse action, except opening a lock; the latter action excludes only other actions opening the same lock (similar to irrelevant deletes), and each lock needs to be opened at most once, as locks can not be closed (static add effects). In the *Miconic* domains, moving the lift can be inverted; letting passengers in- or out of the lift can not be inverted (as the passengers will only get in or out at their respective origin or destination floors), but those actions need to be applied at most once (similar to static add effects) and they do not interfere with anything else (similar to irrelevant deletes).

In *Schedule*, any state s with $gd(s) < \infty$ can be solved by applying a certain sequence of working steps to each part in turn. If that sequence can not be applied for some part p—which must be the case in any dead end state—then it follows that this part is hot in s. No operator adds the fact that a part is cold. But from the dead end state s at least one needed working step requires p being cold as a precondition. It follows that there can be no relaxed solution to s either, as the relaxation does not improve on the add effects. ■

In *Assembly*, one *can* construct an unrecognized dead end state, falsifying the hypothesis that all dead ends are recognized there. We have proven that the construction of an unrecognized dead end involves complex interactions between the ordering constraints that can be present in *Assembly*. These complex interactions are not likely to appear when

ordering constraints are sparse like in the AIPS-1998 benchmark suit, and the interactions are particularly unlikely to appear in small instances as were used in the previous investigation. We refer the interested reader to the technical report (Hoffmann 2001a) for details. As the existence of unrecognized dead ends implies the existence of local minima, in consequence the hypothesis that there are no local minima in *Assembly* is also falsified.

Local Minima

Like before, there is no need to prove anything where the empirical work already revealed a negative example. Most of our positive results concerning local minima are proven by application, or along the lines of, Lemma 3. A few results make use of rather individual properties of the respective domains.

Theorem 2 *The state space of any solvable planning task belonging to the Blocksworld-no-arm, Briefcaseworld, Ferry, Fridge, Grid, Gripper, Hanoi, Logistics, Miconic-SIMPLE, Miconic-STRIPS, Movie, Simple-Tsp, or Tyreworld domains does not contain any local minima under evaluation with h^+.*

Proof Sketch: With Theorem 1, none of those domains contains unrecognized dead ends. As follows from the theorem's proof sketch, all actions in the *Ferry*, *Gripper*, *Logistics*, *Movie*, *Simple-Tsp*, and *Tyreworld* domains are either at least relaxed invertible, or have irrelevant delete effects. With Lemma 3 it suffices to show that all actions are respected by the relaxation. In *Movie*, if a snack is bought in an optimal plan then the snack must also be bought in any relaxed plan, likewise for rewinding the movie or resetting the counter; in *Simple-Tsp*, the optimal plan visits a location that is not yet visited, and any relaxed plan must also visit that location; in *Tyreworld*, if some working step has not yet been done then the relaxed plan must also do it; the *Ferry*, *Gripper*, and *Logistics* domains are all variations of the transportation theme, with actions that load objects onto vehicles, actions that move the vehicles, and actions that unload objects. As an example proof, consider *Logistics*: if an optimal plan loads or unloads some package then that package must still be transported and the relaxed plan has no better option of doing so; if an optimal plan moves a vehicle then that vehicle must either deliver or collect some package, and again the relaxed plan has no better choice.

In the *Fridge*, *Miconic-SIMPLE*, and *Miconic-STRIPS* domains, the actions do not adhere strictly to invertibility according to Definitions 3 and 4; but we have seen that they have similar semantics, i.e., they can either be inverted, or delete only facts that are no longer needed once they are applied. Furthermore, all actions in these domains are respected by the relaxation: in *Fridge*, similar to *Tyreworld*, missing working steps must also be done in the relaxed plan; in *Miconic-SIMPLE* and *Miconic-STRIPS* similar arguments like above for the transportation domains apply.

In *Briefcaseworld*, all actions can be inverted. Actions that move the briefcase or put in objects are respected by the relaxation due to the transportation arguments. Taking out objects is not respected because the objects already have their *at*-relation added (as a conditional effect) by moving the briefcase. However, taking out an object does not delete important facts if that object is already at its goal location. Thus, in a state s where an optimal plan starts with a take out action, an optimal relaxed plan for s can also be used for the successor state. It follows that h^+ does not increase on optimal solution paths.

For the remaining three domains, the proofs are more sophisticated. In all cases it can be proven that there is a path to the goal on which h^+ does not increase. In *Blocksworld-no-arm*, if an optimal starting action a stacks a block into its goal position, then a also starts an optimal relaxed plan. If there is no such action a in a state s, then one optimal plan starts by putting some block b—that must be moved—from some block c onto the table, yielding the state s'. Any relaxed plan P^+ for s also moves b. To obtain a relaxed plan for s', that moving action can be replaced in P^+ by moving b from the table instead of from c. So in all states there is an optimal starting action leading to a state with equal or less h^+ value.

In *Grid*, a rather complex procedure can be applied to identify a flat path to a state with better h^+ value. In a state s, let P^+ be an optimal relaxed plan for s, and a the first unlock action in P^+ or a putdown if there is no such unlock action (the last action in P^+ is a putdown without loss of generality, as the only goals are to have some keys at certain locations). Identifying a flat path to a state s' where a can be applied suffices with Proposition 1: unlocking deletes only facts that are irrelevant once the lock is open, and the deletes of putting down a key are irrelevant if there are no more locks that must be opened. The selected action a uses some key k at a position x. P^+ must contain a sequence of actions moving to x. Moving along the path defined by those actions does not increase h^+: those actions are contained in an optimal relaxed plan, and they can be inverted. If k is already held in s, then we can now apply a. If the hand is empty in s, or some other key is held, then one can use P^+ to identify, in a similar fashion, a flat path to a state where one *does* hold the appropriate key k.

In *Hanoi*, it can be proven that the optimal relaxed solution length for any state is equal to the number of discs that are not yet in their goal position. As no optimal plan moves a disc away from its goal position, h^+ does thus not increase on optimal solution paths. ■

Maximal Exit Distance

We finally present our results concerning the maximal exit distance. The proof technique is to walk along optimal solution paths until an action is reached whose delete effects are no longer needed once it is applied.

Theorem 3 *To any of the Ferry, Gripper, Logistics, Miconic-SIMPLE, Miconic-STRIPS, Movie, Simple-Tsp, or Tyreworld domains, there is a constant c such that, for all solvable tasks belonging to that domain, the maximal exit distance in the task's state space is at most c under evaluation with h^+.*

Proof Sketch: We have seen that in all these domains the actions are respected by the relaxation, and can either be in-

verted or have irrelevant deletes. If s is a state and a an action starting an optimal plan for s, then a starts an optimal relaxed plan P^+ for s, and a relaxed plan for $Result(s, a)$ can be constructed by either: replacing a in P^+ by the respective inverse action, which re-achieves a's delete effects; or by removing a entirely, which can be done if the delete effects of a are not needed by P^+. In the latter case, $h^+(Result(s, a)) < h^+(s)$ follows (as is stated by Proposition 1). So it suffices to derive a constant number c of steps on any optimal solution path such that after c steps there is an optimal starting action for which the second case holds.

In *Movie*, all actions have no, and therefore irrelevant, delete effects, with the single exception of rewinding the movie (which deletes the counter being at zero). Obviously, no optimal plan rewinds the movie twice in a row. Thus, $c = 1$ is the desired upper limit.

In *Simple-Tsp*, $c = 0$ suffices. With the terminology two paragraphs above, say we are at location l in s. Any optimal plan starts by visiting a yet unvisited location l'. A relaxed plan for $Result(s, a)$ can be constructed by removing a from P^+, and replacing all moves from l to some l'' with moves from l' to l''.

In the transportation domains *Ferry*, *Gripper*, *Logistics*, *Miconic-SIMPLE*, and *Miconic-STRIPS*, the argument is the following. If the optimal plan starts with an unload- or load-type of action, then that action can be removed from P^+ to form a relaxed plan for $Result(s, a)$: for unloads, once an object is where you wanted it to be, you don't need to have it in the vehicle anymore; similarly for loads, once the object is inside the appropriate vehicle, you do not need it anymore at its origin location (in *Gripper*, "loading" a ball also deletes the respective hand being free; however, the hand is made free again anyway by the relaxed plan, when it puts the ball into its goal location; a similar argument applies in *Ferry*). Concerning moves, in all these domains all locations are immediately accessible from all other locations (for the appropriate type of vehicle), so no optimal plan moves a vehicle twice in a row, which gives us $c = 1$ as the constant upper limit.

In *Tyreworld*, the lowest constant upper limit is $c = 6$. If the optimal plan carries out some working step a that needs to be undone later on (like jacking up the hub with the flat wheel on), then the relaxed plan for $Result(s, a)$ must include the inverse action to a (like jacking down the hub). If the optimal plan carries out some final working step that does *not* need to be undone (like putting away a tool no longer needed, or jacking down the hub), then that action can be removed from P^+. As it turns out, $c = 6$ is the maximal number of non-final working steps that any optimal plan does in a row. ■

For the *Blocksworld-no-arm*, *Briefcaseworld*, *Fridge*, *Grid*, and *Hanoi* domains, Theorem 2 proves that there are no local minima. Thus, those domains stand a chance of having a constant upper limit to the maximal exit distance. However, in all of these domains one can easily construct instances where the maximal exit distance takes on arbitrarily high (finite) values. In *Grid*, for example, consider the instances where the robot is located on a $n \times 1$ grid (a line) without locked locations, the robot starts at the leftmost location, and shall transport a key from the rightmost location to the left end. The initial value of h^+ is $n + 2$ (walk over to the key, pick it up, and put it down—the *at* relation is not deleted), and the value does not get better until the robot has actually picked up the key. In *Hanoi*, the maximal exit distance grows in fact exponentially with the number of discs. For details the reader is referred to our technical report (Hoffmann 2001a).

The Taxonomy

Our results together with the negative examples found in the previous empirical investigation (Hoffmann 2001b) prove the picture specified in Figure 2.

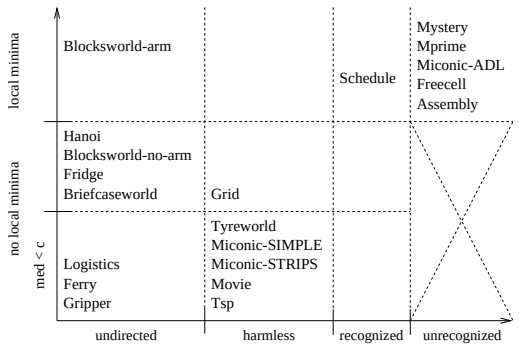

Figure 2: The extended and revised planning domain taxonomy, overviewing our results.

All of the competition domains belong to exactly those parts of the taxonomy, shown in Figure 2, where the empirical observations suggested to put them; except the *Assembly* domain. Obviously, most of the domains are to be found near the left bottom side of the taxonomy.

As discussed in the previous work, from the point of view of heuristic planners like HSP or FF which derive their heuristic function by approximating h^+, the left bottom side of the taxonomy is intuitively the "simple" corner: there are no or only recognized dead ends, and h^+ is a heuristic of high quality. In contrast, the top right corner of the taxonomy contains the "demanding" domains: there can be dead ends that are not recognized by h^+, and local minima. Consistently with this, the *Freecell* and *Miconic-ADL* domains constituted much more of a problem to the heuristic planners in the AIPS-2000 competition than, for example, the *Logistics* domain did. More intriguingly, we have seen that FF's search algorithm is polynomial when the heuristic has the quality corresponding to the lower most part of the taxonomy, i.e., a limited maximal exit distance. For 8 of our 20 domains, h^+ does in fact have this quality.

Conclusion and Outlook

Looking at a large collection of commonly used benchmark domains, we have proven that in the majority of these domains the h^+ heuristic function has the qualities hypothesized by previous work (Hoffmann 2001b). As many current state-of-the-art heuristic planners work by approximat-

ing that same h^+ function, this suggests to interpret those planner's success as utilizing the quality of h^+.

In the process of proving our results, we have also determined the reasons behind the quality of h^+, from a more structural point of view: the reasons are mainly that in many domains all actions are (at least) invertible or need not be inverted (like when their adds are static and their deletes are irrelevant); and that the actions are respected by the relaxation, i.e., often there is simply no other way to achieve the goal than by applying them. The knowledge about these implications opens up the possibility of recognizing the relevant structural properties automatically, and thereby predicting the performance of planners like FF. This can be useful for designing hybrid systems; in particular, it might be possible to identify sub-parts of a task (some more on that below) that can efficiently be solved by heuristic planners. In fact, most of the definitions given in our theoretical core are purely syntactical. Lemma 3 gives a new sufficient criterion for recognizing planning tasks where there are no dead ends in the state space (a problem which Hoffmann and Nebel (2001) have proven to be PSPACE-hard). The only non-syntactical prerequisite of Lemma 3 is inconsistency, for which there are several good approximation techniques in the literature (Fox & Long 1998; Gerevini & Schubert 2000; Rintanen 2000). The challenge is how to determine that an action is respected by the relaxation, and thereby identify situations where h^+ does not yield local minima.

The main piece of work left to do is to corroborate the argumentation that planners like FF and HSP are in essence utilizing the quality of h^+—that is, it must be verified to which extent those planner's approximative heuristic functions really have the same quality as h^+. On the collection of small examples looked at in the previous empirical investigation, FF's heuristic function is similar to h^+ (Hoffmann 2001b). To verify this observation in general, one can look at fragments of the state spaces of larger planning tasks, and compute statistics about the distribution of local minima etc.

Another interesting future direction is to try and prove properties of h^+ for domain *classes* rather than for single domains in turn. Recent work by Malte Helmert (2001) has defined a hierarchy of transportation domains in the context of investigating their complexity. Judging from our results, it seems to be the case that, with Helmert's terminology, any transportation domain where there is unlimited fuel does not contain local minima under h^+. The main difficulty in such an investigation is the definition of the domain class: in difference to Helmert who focuses on the *semantics* of the domains, for our purposes we need a strictly *syntactical* definition: after all, h^+ depends directly on the syntax of the operators.

Let us finally address one of the most pressing questions opened up by this work: is this a *good* or a *bad* result for AI planning? Does it mean that we have identified widely spread structural properties that can make planning feasible, or does it mean that our benchmarks are superficial? Probably, both. The author's personal opinion is this. As for the benchmarks, it is certainly no new perception that they are more simplistic than realistic. On the other hand, it will not serve the goals of the field to construct more complex examples with the sole intention of outwitting heuristic planners. The best would be to use real-world applications, or at least as close as possible models thereof. As for the structural properties we have identified, it might be feasible to exploit these even if they occur only in sub-parts of a task, and it might well turn out that (sub-parts of) real-world tasks (like transportation issues with sufficient fuel available) exhibit that structure quite naturally.

References

Bacchus, F. 2001. The AIPS'00 planning competition. *AI Magazine* 22(3):47–56.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1–2):5–33.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, 714–719. MIT Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *AIJ* 69(1–2):165–204.

Edelkamp, S., and Helmert, M. 2001. The model checking integrated planning system (MIPS). *AI Magazine* 22(3):67–71.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in tim. *JAIR* 9:367–421.

Frank, J.; Cheeseman, P.; and Stutz, J. 1997. When gravity fails: Local search topology. *JAIR* 7:249–281.

Gazen, B. C., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. ECP-97*, 221–233. Toulouse, France: Springer-Verlag.

Gerevini, A., and Schubert, L. 2000. Inferring state constraints in DISCOPLAN: Some new results. In *Proc. AAAI-00*, 761–767. Austin, TX: MIT Press.

Helmert, M. 2001. On the complexity of planning in transportation domains. In *Proc. ECP-01*, 349–360. Toledo, Spain: Springer-Verlag.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2001a. Local search topology in planning benchmarks: A theoretical analysis. Technical Report 165, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany. ftp://ftp.informatik.uni-freiburg.de/documents/reports/report165/report00165.ps.gz

Hoffmann, J. 2001b. Local search topology in planning benchmarks: An empirical analysis. In *Proc. IJCAI-01*, 453–458. Seattle, Washington, USA: Morgan Kaufmann.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR* 12:338–386.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *Proc. AAAI-00*, 806–811. Austin, TX: MIT Press.

Knowledge-based Configuration for Configuring Combined Hardware/Software Systems

Thorsten Krebs[1] and **Lothar Hotz**[2] and **Andreas Günter**[3]

Abstract. In this paper, we present a survey on research topics we consider with in the EU project *ConIPF* (*Configuration of Industrial Product Families*). The application domain which is considered in this project contains combined hardware/software systems. Those systems share a lot of commonalities and are configured in a rich variety of outcoming products. Besides others, we discuss issues concerning features and evolution. Feature trees describing the functionality in terms of customer requirements are used to efficiently select, combine and configure components from the product catalogue. Mappings between these feature models and the product descriptions have to be developed and modeled sufficiently for the configuration task. A further problem area arises in this application domain with evolution in both the products themselves and the models describing the product catalogue.

1 Introduction

The project *ConIPF* (*Configuration of Industrial Product Families*) is a three year project that is carried out in the European Union (EU) with four partners (two industrial and two university partners). These partners are the University of Hamburg, the University of Groningen, Robert Bosch GmbH and Thales Naval Netherlands.

Software is an important basis of most technical systems. Growing complexity and variability of technical systems replace the development of single software products with the development of product families. Furthermore, the increasing use of embedded systems combining hardware and software make the process of software development to a difficult task. To get started, we will analyse software product development processes of industrial companies, restricting us to industrial product lines with hardware and software components. As industrial products, we will focus on the car periphery supervision domain, which includes services like pre-crash detection, and parking assistance. Besides considering software and hardware components, also requirement templates, feature models, and intermediate representations will be examined.

Currently one approach for handling software systems is the product line approach [11]. In this research field distinct kinds of variability of software and their realization with appropriate architectures and implementations are examined. In the project, we will consider how configuration methodologies known in Artificial Intelligence can be applied to the product line approach. Already discovered problem areas in the current situation are pointed out in the following. Possible solutions to these problems are given as far as the project status allows to consider this.

The paper is structured as follows. First we give an overview about some application domains. Section 3 introduces aspects about Configuration known in Artificial Intelligence (CAI). In Section 4 related work is discussed. In Section 5 we describe the problem domain and aspects of the possible solution domain. Section 6 presents future work that will be done in ConIPF.

2 Application Domain

As a relatively new domain for applying knowledge-based configuration methods in the EU project ConIPF combined hardware/software systems consisting of hardware components like sensors, electronic control units, displays and software components like assembler programs, DLLs, functions are explored.

2.1 Application Areas

The following are examples for application areas in the above mentioned domain:

- The **Car Periphery Supervision** pursues detection of the car environment, recognition of hazardous situations and handling of difficult traffic situations. Main goals are avoiding stress, accidents and thereby damage to the driver and the car itself. Typical applications are parking assistance, pre-crash detection, automatic cruise control, side obstacle detection, changes in geometry (e.g. trailer operation) and diagnosis.
- **Gasoline** and **Diesel Systems** are complex combinations of electronic control units (ECUs) like engine control systems, fuel support systems and on-board diagnosis systems. This includes hard- and software. These systems are developed for low-cost, midrange and high-end market segments and for different country-specific requirements like emission rules in Europe and the US.

2.2 Variability

In domains like the above mentioned, combined hard- and software systems are developed. The combination of these components is determined by customer requirements that are mapped to features which affect the possible selection of hardware components and software modules. Thus, variability in the development process are considered by the following aspects:

- **Customer requirements** can be of very different nature. Thus, not all possible combinations of a system are described in feature models. Sometimes the customer is not interested in all the details

[1] LKI, Fachbereich Informatik, Universität Hamburg, email: krebs@informatik.uni-hamburg.de
[2] HITeC, c/o Fachbereich Informatik, Universität Hamburg, email: hotz@informatik.uni-hamburg.de
[3] dito, email: guenter@informatik.uni-hamburg.de

of possible functionality. This makes the system not fully configurable with traditional knowledge-based configuration methods. Instead, components that are needed for a specific customer might have to be newly developed.

- **Features** are hierarchically modeled in a tree structure describing the system functionality. There can be hundreds of nodes in such feature trees. Features themselves can be related to each other by means of alternatives, existence or exclusion. Furthermore, features taking part in such relations can be mandatory or optional. In [6, 13] feature models that are under development in the area of combined hard- and software systems are described.
- **Software and hardware components** exist in distinct versions and variations, with sometimes explicitly documented or implicitly known dependencies between each others.

2.3 Current Configuration Approach

Currently there is no automatic configuration of standardized products in the companies that we considered. The focus is placed on functions for establishing a bottom-up composition of multifunctional systems. Variability usually evolves over long time periods (often over years). In order to establish a suitable product line approach, the need of computerized configuration support arises.

The development takes place in an evolutionary cycle such that future products benefit from the development experience. Development of products is an incremental procedure, which is customer specific. First a concept is elaborated which then is used for describing the product functionality. Subsequently, software components have to be coded, linked, packaged, tested and calibrated - in that order. When a new product is created the development starts often from similar products that has been developed before. Then existing models are copied and modified to meet the desired functionality. When the end of line is reached the product is delivered to the customer (e.g. a car manufacturer).

To handle different versions of software modules and changings Software Configuration Management (SCM) tools are used. SCM systems focus on controlling the whole evolutionary process of software system development. This process is seen as a continuous process which does not stop while the software is in use. To support this constantly changing process, SCM systems have been developed. There are a number of SCM systems which support the main concepts of SCM systems more or less (for surveys in SCM see e.g. [3, 4, 7, 21]). These concepts are: The *representation* of software objects as *atoms* by giving them a rudimentary name and a not further specified content ”description” (e.g. ”program code”, ”documentation”, ”test result” etc.), or as *configurations* by enumerating independently changing atoms or other configurations. *Version control* examines how interim products are produced in the course of product development and how development of different aspects of the product can proceed in parallel. *Change control* examines how changes to the software objects are more or less formally described by giving information about the change like the objective of the change, the state of the change (e.g. open, rejected) etc. *Process control* supports the whole software development process, by describing the process in terms of ”completion, acceptance, integration & test, take over”. *Distribution* is related to distributed development of software by locally distributed developers. Further issues on the relation between SCM and CAI are discussed in [14].

2.4 Some Requirements for Future Systems

There are a few goals for future development of products. All product-specific documentation should be collected in one place to allow usage in different stages (and departments) during the development process. Customer offers then could be made based on feature models. Furthermore system features could be used for a better understanding with the customer. For example opposing features could be detected. Also evaluation of customer requirements could be supported. Further issues are:

- For the development an aim is to distinguish between changes of existing products, which are already included in a **product line** and new product specific functionality, which is not part of a product line. By integrating new products into a product line approach a larger group of developers can be supported.
- Also **feature trees are evolving** over time. Features are integrated into already existing trees depending on whether they are relevant for the product line. New features can be used in future product developments.
- A further **goal of a configuration methodology** is the possibility of calculating time necessary for adopting the software and cost for the necessary hardware components for a customer specification. Further, a technical judgment through the sales department leading to some kind of filtering is aspired.

3 Knowledge-based Configuration known in Artificial Intelligence (CAI)

The configuration of technical systems is one of the most successful application areas of knowledge-based systems. [10] made a general analysis of configuration problems in which four central aspects (or knowledge types) concerned with configuration tasks were identified:

- A set of domain objects (*concepts*) in the application domain and their *properties* (parameters). By instantiating a domain object *concept instances* are created. Thus, a domain object describes its instances.
- A set of relations between domain objects. Taxonomical and compositional relations with alternatives, number restrictions, and optionality are of particular importance for configuration. But further relations can be expressed between arbitrary domain objects.
- A *task specification* (configuration objective) that specifies the demands, which a created configuration must fulfill.
- *Control knowledge* for specifying the configuration process.

For all those kinds of knowledge not only a model can be declaratively defined but also an inference machinery is given, which interprets the knowledge. Thus, CAI provides not only a modeling facility like STEP or UML but operational, processable models. In the so-called structure-based approach a compositional, hierarchical structure of the domain objects serves as a guideline for controlling the solution process, and as a logical basis for configuring. That means, that the constructs used for modeling can be mapped to a logical language where the semantics are well-defined [19]. The constraint-based approach consists of representing restrictions between objects or their properties by means of constraints, and evaluating these by constraint propagation. This approach is not in conflict with the structure-based approach but is frequently combined with it. Other approaches are resource-based and case-based configuration.

Concepts used in the area of configuration have well-defined, system independent semantics which are manifested in implementations

termed *configuration systems* (CS) like *EngCon* [1]. Such systems provide a more formal notion of consistency and completion than Software Configuration Management systems [18]. For instance, in CS the consistency of the hierarchy is well-defined (namely a strict specialization hierarchy) and will be checked by a specific module of a CS. Constraint solving uses methods that have been proven correct, and property values of components can be inferred. The control mechanism determines that all open issues (e.g. properties, parts) of the configuration are handled by the configuration process [9]. All these modules of a CS are general and thus, domain independent. A domain specific configuration system can be obtained by implementing a domain specific application user interface over the domain specific configuration model. Such a user interface maps those kinds of knowledge and inference processes to interface components that are suitable for the domain specific configuration process being supported. For example, in the case of software development a user interface may consist of presentation tools for software objects using the concepts mentioned, presenting the evolutionary process etc. CS map the configuration process to an operational computer supported process. As such, CAI gives a kernel technology for distinct application domains, but which always needs an appropriate surrounding.

In CAI, this has been done traditionally for domains where hardware components are configured. For software, the main challenge is to understand the specific concepts of the software development process in terms of the general concepts of the logic-based configuration terminology.

Note that most configuration tasks from the principle-, variant- and adaptive construction areas known in the field of mechanical engineering can be equated with configuration.

For software product lines which are the objects of our attention, the non-static aspect of running software is important [18]. The main challenge is to understand the specific concepts of the software development process in terms of the general concepts of the logic-based configuration terminology.

4 Related Work

The Finnish research group around T. Männistö explores the usage of tool support for the configuration of software product families. In [22] a Linux Familiar operating system distribution is modeled with a configuration modeling language aimed at representing the structure of physical products. The used language is largely suitable for software product configuration. But it is also stated that functions, features or resources and optimality criteria, as well as versioning and reconfiguration would be useful.

In [2] the possibility of applying techniques developed for configuring mechanical and electronic products for configuring software is studied. A way of representing much of the architectural knowledge using the configuration modeling concepts is defined. This indicates that it is relatively easy to provide software configuration support using existing techniques. However, it is required to extend the current conceptualization of configuration knowledge to capture software products adequately.

5 Problem Areas and Possible Solutions

In this section we present problems mentioned in the Software Configuration Management (SCM) community and their potential solutions coming from Configuration Methodologies known in AI (CAI). We focus on *software product representation*, *versioning*, and *representing distinct kinds of knowledge*. Further on, we present issues on feature modeling and evolution (for the last topic see also [17] and [5]).

5.1 Software Configuration Management

In the following some issues concerning Software Configuration Management are presented.

- In current commercial SCM systems *files* and file handling are the basic components, thus, operations like merging, revision etc. are based on files not on objects [4]. *Models* and software products (*instances*) are roughly seen as the same kind of entities namely software programs [5]. These facts demonstrate a main problem: there is no abstract, declarative model of the source code being configured [17]. Each task is done directly on source code and files. In CAI a configuration language is given which provides the means for defining the central aspects of configuration tasks (see Section 3). With such a model, software products could be defined on a concept level, and the instances (e.g. files, source code), which realize a specific application, can be computed by the configuration system.
- In [21] it is mentioned that versioning is mainly based on a directed acyclic graph by using the `is-version-of` relation. In CAI one can represent this aspect in the framework of specialization hierarchies. Here, versions can be described in terms of subconcepts with specializing properties or relations. Thus, also version management of structures, relationships and interfaces can be modeled, which is a further requirement mentioned in [7]. As described in Section 3, not only a model but also an inference machinery is given by CAI, hence, consistent configurations selected from multiple versions can be computed. Thus, the selection of appropriate components is dynamically supervised by the configuration system.
- The CAI methodology is general and generic. Hence, distinct aspects like features, requirements, designs, test cases, task agenda, standard code components and their relations, etc. could be represented (see [21]). These diverse aspects of a domain can be modeled in distinct knowledge bases and can be combined, e.g. by using *strategies* [9], in an integrated system. As an example, in [13] a feature model is described, which includes common and variable aspects of the capabilities of software products. Such a feature model could be used by a configuration system for identifying suitable software and hardware components.

Further open issues which do not have an obvious solution and are still research aspects in both fields, CAI and SCM are:

Combination of CAI methods and SCM techniques The features of SCM, that are well understood are e.g. buildability, versioning on file level, and (partly) workspaces. Their integration with the kernel methods of CAI must be taken into account. Further integration issues for SCM are discussed in [7].

Representing functionality of software modules When assembling software components the problem is to identify suitable components, specify their interfaces, and infer the overall state description of the aggregate. For this task the *functionality of software modules* in terms of states and their dependencies are important. Methods like state charts and automaton can possibly be used for modeling [16]. Further on, techniques like *simulating* totally or partly configured components should be considered.

Product variability In [7] distinctions between permanent variants: variability at construction time and product variability: variability

that must be handled by the product are mentioned. The first can be handled by domain modeling. For considering the variability at run time, variable components and their dependencies should be part of the resulting configuration. In this case methods from CAI like *reconfiguration*, and knowledge base evolution should be examined.

Distributed, concurrent work In [7] further on, problems with distributed work on same products are discussed, e.g. connectivity between group members, multi-site, multi-organization. Approaches in CAI which work in this area are named *distributed configuration*. Decisions about distributing artefacts, parallel development, merging of configured components, and propagating of dependencies are studies in this field.

Aspects, which are more related to SCM, are deeply discussed in [4, 7, 21]: Automated change integration and merging of changes; interoperability among configuration management systems; relationship between software architecture and configuration management systems; dealing with building of executable object modules, representing data flow among modules, representing control flow among modules; deployment issues and post deployment phase: distributing software into the field and the maintaining if once there. How these problems can be handled with CAI as a kernel technology is still open.

5.2 Feature Models

Features in terms of customer requirements and system functionality are used to specify products to be developed. Features are hierarchically modeled in tree structures to allow specializations and decompositions. These feature trees are then used to support the configuration task.

A set of hard- and software components has to be combined to fulfill specific customer requirements. The desired functionality within a product line can differ so much that a non-expert might not be able of building a valid composition of given components. Further, the components each have a specific functionality which together build the functionality of the whole system. Speaking of functionality in terms of customer requirements the major difficulty is to choose the components of a system so that they work together and provide the desired system functionality.

A main problem is to map system requirements as a set of features to the hard- and software components actually combined in the developed system. Different components may work together well or may be not. In the latter case they should not be connected to construct a stable system architecture. Further, the union of the components functionality has to provide the desired functionality corresponding to the customer requirements.

The wishes of the end user are evolving with new improvements in technical research areas. Thus, the models describing customer requirements have be able to deal with this. In rather short evaluation cycles new products are introduced into the product catalogue and thus technical requirements and dependencies are getting more complex. Because multiple software components are already present and used for new product development, the integration of a case-based approach seems promising. A *case* would describe an already developed software components, which should be integrated into the configuration process.

The configuration tools *KONWERK* [8] and *EngCon* [1] are examples for structure-oriented configuration tools. They are able to guide a user through the configuration process in order to create a solution tailored to suffice the user specific configuration task. A flow of configuration steps can be modeled as process knowledge and used to enhance the quality of possible solutions. A methodology for using ontologies as domain knowledge input is also provided. This methodology is able of dealing with taxonomic relations, decompositions, artifact specific parameters and constraints between components and their parameters.

Both tools currently do not support feature trees in the form described above. It is in the scope of the *ConIPF* project to evaluate if feature trees can be incorporated into them. Further, a mapping between features and components has to be defined to allow an efficient composition of the desired system.

5.3 Evolution

Reuse of existing components can be enhanced when models of components are present and are used for configuring new products (configurations). But, mostly not only existing components have to be incorporated in new products but some components have to be modified to supply customer requirements. The inclusion of those new components in the model would be an evolution of the model and give the possibility of future reuse of those new components.

Some examples of how evolution can influence the system development task are:

- A new kind of short range radar is developed. This one can observe more of the surrounding environment. Other software modules (drivers) may be needed to make this work properly.
- A CPS system may consist of a set of applications. Usually the significant differences in these systems are the application versions and the composition of sensors and software modules.
- One customer would like to have a combination of 4 yellow and 2 red bar graphs for displaying the range detected by a park pilot system and a short interval tone for making the critical distance audible. Another car manufacturer may wants to have a different display but the same speaker settings. And both are not yet included in the configuration model.

Following entities can be subject of evolution inside the model:

- Inserting new components and/or new features
- Handling distinct dependencies between different versions of components
- Evolving relations between features and components
- Expanding decomposition and taxonomical relations

A further example where evolution can occur is given by changing user requirements during development time. Thus, those user requirements are not fixed in the beginning of system development. For requirements that are foreseen in a configuration model the knowledge-based configuration approach is suitable, because changing requirements could be handled by subsequent configuration sessions. But, if there are requirements which are not part of a model (e.g. intuitively discovered during testing a prototype of the product), this situation should be handled with further methods. One notion is to develop some kind of open, innovative, expandable model that covers most (probably unknown) requirements in a generic way and supports the refinement of the model during the configuration session (compare [15]).

A more economical aspect of evolution is given by a small amount of personal and financial resources which are used for evolution. Evolution of a model is mostly not planned and included in the product development process for a specific customer. For a customer more

specific needs are important, like quick specific samples, short development time, etc.

Currently a copy-and-modify-approach exists often for dealing with evolution in system development. New versions of components are created by copying code of similar systems and modifying it. For this, knowing of the newest and / or all versions is necessary. The selection of suitable versions for given requirements is done manually. An individual development of customer specific products is realized. While developing the devices for specific customer requirements is fast at first sight but it becomes slow and expansive in contrast to reuse of such devices.

Requirements for the methodology in respect to evolution are:

Combining of modeling, and configuration: Moving from a more separating approach, where first a configuration model is specified and then configurations are inferred by use of that model, to a more integrated or intrinsic approach, where modeling and configuring are interchanged.

Combining of modeling, configuration and even implementation: To support a product developer one has to combine all three tasks: modeling, configuring, and implementing, because all those tasks have to be executed for realizing a specific product. First, for new customer requirements a suitable product is configured by using the already existing knowledge (model). Then specific software modules, which are not yet incorporated in the model have to be implemented. After that, those new modules have to be included in the model - by evolving it. How this can be supported during the implementation phase is an open issue.

Ease of modeling: Because of the combination modeling, configuring, and implementation the modeling task is not done by a specific kind of knowledge engineer, who knows all about the modeling facilities, but the modeling will be done by software developers. Thus, the modeling has to be included in their daily work, and known tooling.

Distributed development: Software development and modeling is done by multiple developers in a distributed fashion. Thus, distributed configuration, distributed modeling and distributed implementation has to be taken into account.

Version management for models: To handle the evolving model some kind of version management for the model (not only for some software modules) has to be realized.

Evolution of existing products: A further question is how new versions are included in existing products which are already in use. In this case methods from CAI like *reconfiguration*, and knowledge base evolution are to be examined.

6 Future Work

Future work will be evaluating the suitability of existing configuration methods for usage in combined hardware/software systems. Therefore a possible integration of feature trees into structure-oriented configuration has to be examined. A mapping between feature trees and product catalogues has to be modeled. Here existing representations e.g. of the *KONWERK* tool [8] can possibly be used since they already provide the hierarchical concepts of specializations and decompositions. Non-hierarchical relations (i.e. requires, excludes etc.) as needed by feature models should be integrated.

Thus, the scope of the *ConIPF* project is to examine the following issues:

- In order to support realistic industrial applications, guidelines will be described for facilitating domain modeling (see [18, 20] for similar approaches).
- A further focus is set to products that can be developed quickly and with little effort. On the one hand an early result can be produced and on the other hand, for those components libraries can be developed for reusing generic software components.
- For modeling, known configuration description languages will be used and possibly further developed for describing specific aspects of software modules, like functionality and state descriptions.
- A technology used for modeling will be Description Logics (DL), which provides a well-defined semantics for basic concept and role (=relation) definitions (for a description of a DL system see [12]). The combination of a DL with CAI will be examined to support inferencing on the concept level, e.g. for automatically classifying new component models in a specialization hierarchy.

REFERENCES

[1] V. Arlt, A. Günter, O. Hollmann, T. Wagner, and L. Hotz, 'Engcon - engineering & configuration', in *Proc. of AAAI-99 Workshop on Configuration*, Orlando, Florida, (1999).

[2] T. Asikainen, T. Soinen, and T. Männistö, 'Representing software product family architectures using a configuration ontology', in *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, Lyon, France, (July 21-26 2002).

[3] S. Dart, 'Concepts in configuration management systems', in *Proc. of the 3rd. Intl. Workshop on Software Configuration Management*, Trondheim, Norway, (1991).

[4] J. Estublier, 'Software configuration management: a roadmap', in *ICSE - Future of SE Track*, pp. 279–289, (2000).

[5] J. Estublier, J.M. Favre, and Morat P., 'Toward PDM / SCM: integration?', in *Proc. of the 8. Intl. Workshop on Software Configuration Management*, LNCS 1439, pp. 75–95, Bruxelles, Belgium, (July 1998). Springer Verlag.

[6] A. Ferber, J. Haag, and J. Savolainen, 'Feature interaction and dependencies: Modeling features for re-engineering a legascy product line', in *Proc. of 2nd Software Product Line Conference (SPLC-2)*, Lecture Notes in Computer Science, San Diego, CA, USA, (August 19-23 2002). Springer Verlag.

[7] K. Frühauf and A. Zeller, 'Software configuration management: State of the art, state of the practice', in *9th International Symposium on System Configuration Management (SCM-9)*, Toulouse, France, (1999).

[8] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995. in german.

[9] A. Günter and R. Cunis, 'Flexible control in expert systems for construction tasks', *Journal Applied Intelligence*, **2(4)**, 369–385, (1992).

[10] A. Günter and C. Kühn, 'Knowledge-based configuration - survey and future directions', in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, (1999).

[11] J. van Gurp, J. Bosch, and M. Svahnberg, 'On the notion of variability in software product lines', in *Proceedings of the 2001 Working IFIP/IEEE Conference on Software Architecture*, (2001).

[12] V. Haarslev and R. Möller, 'Consistency testing: The race experience', in *Proceedings TABLEAUX'2000*. Springer-Verlag, (2000).

[13] A. Hein, M. Schlick, and R. Vinga-Marting, 'Applying feature models in industrial settings', in *Software product lines - Experience and research directions*, ed., Donohoe P., pp. 47–70. Kluwer Academic Publishers, (2000).

[14] L. Hotz and A. Günter, 'Using knowledge-based configuration for configuring software?', in *Proc. of the Configuration Workshop on ECAI 2002*, Lyon, France, (2002).

[15] L. Hotz and T. Vietze, 'Innovatives Konfigurieren in technischen Domänen', in *S. Biundo (Hrsg.), 9. Workshop Planen und Konfigurieren*, Kaiserslautern, Germany, (1995). DFKI Saarbrücken.

[16] C. Kühn, 'Modeling structure and behaviour for knowledge based software configuration', in *14th Workshop, New Results in Planning, Scheduling and Design (PuK2000)*, ed., http://www-is.informatik.uni oldenburg.de/sauer/puk2000/paper.html, (2000).

[17] T. Männistö, *Towards Management of Evolution in Product Configuration Data Models*, Ph.D. dissertation, University Helsinki, 1998.

[18] T. Männistö, T. Soininen, and R. Sulonen, 'Product configuration view to software product families', in *Software Configuration Workshop (SCM-19)*, Toronto, Canada, (2001).
[19] R. Möller, C. Schröder, and C. Lutz, 'Analyzing configuration systems with description logics: A case study', in *http://kogs-www.informatik.uni-hamburg.de/~moeller/publications*, (1997).
[20] J. Tiihonen, T. Lehtonen, T. Soininen, A. Pulkkinen, R. Sulonen, and A. Riitahuhta, 'Modelling configurable product families', in *Proc. of the 4th WDK Workshop on Product Structuring*, Delft, The Netherlands, (October 1998).
[21] A. van der Hoek, D. Heimbigner, and L.W. Wolf, 'Does configuration management research have a future?', in *Proceedings of the 5th Inter. Conf. on Software Configuration Management, LNCS 1005*, Berlin, (1995). Springer-Verlag.
[22] K. Ylinen, T. Männistö, and T. Soinen, 'Configuring software products with traditional methods - case linux familiar', in *Proc. of 15th European Conference on Artificial Intelligence (Configuration Workshop)*, Lyon, France, (July 21-26 2002).

Lösen von Scheduling-Konflikten durch Verhandlungen zwischen Agenten

Karl-Heinz Krempels
krempels@i4.informatik.rwth-aachen.de

RWTH Aachen
Lehrstuhl für Informatik IV
Ahornstraße 55
D-52056 Aachen

Zusammenfassung Die Terminplanung im medizinischen Akutbereich ist gekennzeichnet durch eine hohe Komplexität sowohl der Arbeitsabläufe als auch der beteiligten Personalstrukturen. Notfälle und unvorhersehbare Behandlungsabläufe verlangen zusätzlich eine hohe Flexibilität der Planung. Diese Kombination flexibler Planungsprozesse mit komplexen Organisationsstrukturen führt zum Scheitern mathematischer Optimierungsansätze. Die vorliegende Arbeit beschreibt einen Ansatz zur Lösung von Scheduling-Konflikten und stellt einen Teil eines Terminplanungssystems im Krankenhausbereich dar, das auf intelligenten, präferenzbasierten Agenten (Policy-Agenten) als Vertreter reeller Personen basiert. Im Mittelpunkt des verfolgten Ansatzes steht die Integration der Präferenzstrukturen aller beteiligten Personen und der gegebenen hierarchischen Strukturen der Organisationseinheiten in den Schedulingprozeß sowie die Lösung von Scheduling-Konflikten mittels Verhandlungen zwischen Agenten.

1 Einleitung

Die Terminplanung im Akutkrankenhaus ist ein wesentliches Element des Klinikmanagements. Aufgrund der Ungewissheit hinsichtlich des Kapazitätsangebotes und der tatsächlichen Nachfrage ist die Terminplanung im Akutbereich durch ein hohes Maß an planerischer Flexibilität gekennzeichnet, welche von der Terminplanungssoftware gehandhabt werden muss [3]. Aus Akzeptanzgründen sollte eine weitgehend automatisierte Terminplanung auch die Interessen und Präferenzen der individuellen Akteure berücksichtigen. Dies bedingt, dass ein Terminplanungssystem diesen Faktoren in ähnlicher Weise Rechnung tragen muss, wie es bei der derzeitigen Praxis manueller Planung möglich ist und begünstigt den Einsatz von Agentensystemen.

2 Beschreibung des Lösungsansatzes

Die Integration der Präferenzstrukturen aller beteiligten Personen und der gegebenen hierarchischen Strukturen der Organisationseinheiten eines Krankenhauses in den Schedulingprozess bedingt deren Modellierung und formale Spezifikation. Die Idee, einen Scheduling-Konflikt durch eine Verhandlung zwischen Agenten zu lösen, begünstigt die Einführung einer Verhandlungsmasse, aufgrund welcher eine Verhandlung oder ein Tausch ermöglicht wird. Dies muss bei der Modellierung der Präferenz- und Hierarchiestrukturen berücksichtigt werden. Die Modellierung von hierarchischen Strukturen kann einfach vorgenommen werden, indem die Akteure mit einer Verhandlungsmasse ausgestattet werden, die ihrer Rolle in der Hierarchiestruktur entspricht. So erhält ein Chefarzt zum Beispiel mehr Verhandlungsmasse als ein Stationsarzt. Als aufwendiger erweist sich die Modellierung der Präferenzen und die Verteilung der Verhandlungsmasse auf diese. In den folgenden Abschnitten wird hierdrauf näher eingegangen.
Das Scheduling von Aktionen und Ressourcen wird in zwei Phasen vorgenommen:

- in der ersten Phase erstellt der Scheduler einen vorläufigen Plan ohne die Präferenzen der Akteure zu berücksichtigen. Hierfür können bekannte Ansätze [9] verwendet werden.
- In der zweiten Phase versucht der Scheduler den vorläufigen Plan durch die Berücksichtigung der Präferenzen der Akteure zu verbessern.

Die von dem Scheduler erkannten konfligierenden Präferenzen bezüglich einer Ressource werden an das Agentensystem zwecks Lösung durch Verhandlung weitergegeben, wobei hier zu Gunsten genau eines Agenten entschieden wird.

2.1 Die Präferenzen

Eine Präferenz [12] ist die Angabe einer Reihenfolge für zwei beliebige Alternativen a, b in der Form, dass $a \prec b, a \succ b \Leftrightarrow b \prec a$ oder $a \sim b$. So kann ein Akteur in einem Krankenhausszenario zum

Beispiel angeben, dass er lieber mit Schwester A in OP 1 als mit Schwester B in OP 2 operiert. Ein Akteur muss seine Präferenzen der Einfachheit halber in eine totale Ordnung bringen, die die Wichtigkeit der Präferenzen in aufsteigender Reihenfolge darstellt: $p_1 \prec p_2 \prec \cdots \prec p_k$.

2.2 Die Gewichtsfunktion

Um die gesamte Gewichtsmasse M so auf die k Präferenzen eines Agenten zu verteilen, dass die gegebene Präferenzenordnung seines Prinzipals erhalten bleibt, ergeben sich die folgenden Anforderungen an die Gewichtsfunktion W, wobei die erste Präferenz p_i als die unwichtigste angenommen wird und dementsprechend mit dem kleinsten Gewicht belegt wird:

- das Gewicht der Präferenz p_i soll grösser sein als die Gewichte der einzelnen Präferenzen p_j für alle $i > j$ und $1 \leq i, j \leq k$:
$$w_i > w_j, \forall\ i > j\ \ \wedge\ \ 1 \leq i, j \leq k. \tag{1}$$
Dies bedeutet, dass die Gewichte der ersten beiden Präferenzen vorgegeben werden müssen
$$w_1 = m,\ w_2 = qm, \tag{2}$$
wobei entweder das Gewicht der ersten Präferenz m oder das Verhältnis zwischen der zweiten und der ersten Präferenz q angegeben werden kann. Hier werden die Wertebereiche für m und q der Einfachheit halber eingeschränkt: $m \geq 1$ und $q > 1$.

- das Gewicht der Präferenz p_i an der Position i der gegebenen Präferenzenordnung muß gleich der Summe der Gewichte der Präferenzen p_j mit $0 < j < i$ und $i > 2$ sein (also derjenigen Präferenzen die weniger wichtig sind als die Präferenz p_i). Im Folgenden werden die Gewichte der Präferenzen $W(p_i)$ der Einfachheit halber mit w_i bezeichnet:
$$w_i = \sum_{j=1}^{i-1} w_j. \tag{3}$$

- die gesamte Gewichtungsmasse M soll auf die Präferenzen verteilt werden:
$$M = \sum_{i=1}^{k} w_i. \tag{4}$$

Die Gewichte der restlichen Präferenzen w_k lassen sich unter Berücksichtigung der in den Gleichungen 1 bis 4 spezifizierten Anforderungen bestimmen: $w_k = 2^{k-3}m(q+1)$, wobei die maximale Anzahl der gewichtbaren Präferenzen auf $k_{max} = \lfloor 1 + \log_2 M \rfloor$ beschränkt ist.

2.3 Die Verhandlung

Die an der Verhandlung teilnehmenden Agenten werden von einem ServiceAgent mit den Präferenzen ihrer Prinzipale und deren Gewichtungsmasse initialisiert und verteilen anschließend die Gewichtungsmasse entsprechend der vorgestellten Gewichtungsfunktion auf ihre Präferenzen. Der ServiceAgent fragt alle Agenten nach dem Gewicht der konfligierenden Präferenz und wählt den Agenten mit der größten Gewichtung als Gewinner der Verhandlung aus. Dieser Agent verteilt die Gewichtungsmasse der konfligierenden Präferenz auf die restlichen beteiligten Agenten nach einer vorgegebenen Strategie, wobei diese auf ihre Präferenz verzichten. Die Verhandlung wird mit der Übergabe der aktuellen Gewichtungsmasse und der Präferenzenliste eines jeden beteiligten Agenten an den ServiceAgenten beendet.

2.4 Die Nutzenfunktion

Ein Agent muss bestimmen können, wann er während einer Verhandlung besser oder schlechter gestellt ist. Hierfür wird als Maß die gesamte jeweils aktuelle Gewichtungsmasse des Agenten verwendet. Die Nutzenfunktion $N(t)$ ist damit gleich der Summe über alle Gewichte der Präferenzen w_i zum Zeitpunkt t: $N(t) = \sum_{i=1}^{k} w_i$. Eine Besserstellung des Agenten liegt vor, wenn $N(t) < N(t{+}1)$ ist.

3 Umsetzung des Lösungsansatzes

Auf die Anwendung des beschriebenen Lösungsansatzes auf das geplante System wird anhand einer

Beschreibung des in Abbildung 1 dargestellten Ablaufes eingegangen. Die relevanten Eigenschaften der existierenden Ressourcen sowie der aktuellen Anforderungen werden in einer Wissensbasis abgelegt (1) und von einem Expertensystem für das Scheduling (2) verwendet. Wenn von dem Scheduler ein Konflikt gefunden wird, dann werden die davon betroffenen Akteure (personelle Ressourcen) ermittelt und für jeden als Vetreter ein Agent auf einem Agentensystem gestartet (3). Anschließend initialisiert der Scheduler die Agenten mit den Informationen ihrer Prinzipale und der

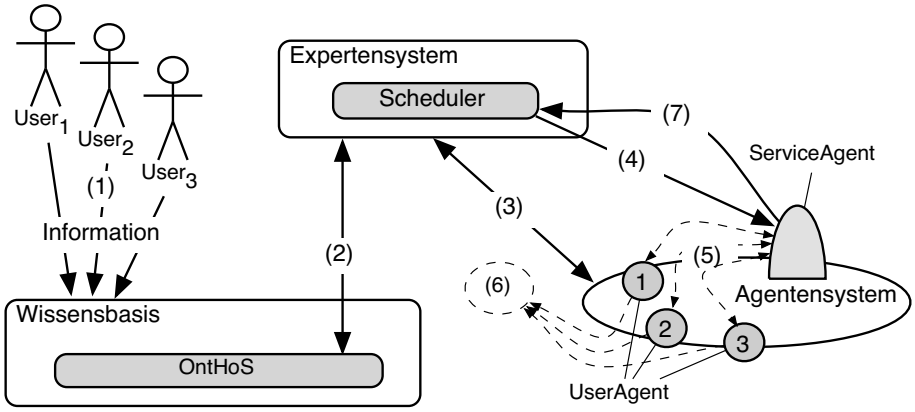

Abbildung 1: Ablauf einer Konfliktlösung durch Verhandlungen von Agenten

konfligierenden Präferenz über einen Dienst des Agentensystems (4,5) und startet eine Verhandlung zwischen den Agenten (6). Das Ergebnis dieser Verhandlung, das genau ein gewinnender Agent ist, wird dem Scheduler über den Dienst des Agentensystems mitgeteilt (7), so dass dieser nun eine konfliktfreie Zuweisung der Aktion zu einer Ressource vornehmen kann. Im Folgenden werden die verwendeten Systemkomponenten näher beschrieben, wobei insbesondere auf ihre Funktionalität und Ihren Aufbau eingegangen wird.

Die Wissensbasis Um eine Lösung des Problems losgelöst von dem realen Umfeld eines Krankenhauses zu ermöglichen, wurde das Domänenwissen mit Hilfe der Domänenontologie OntHoS[1] modelliert und in einer Wissensbasis abgelegt. Hierdrauf können verschiedene Ansätze getestet und ihre Ergebnisse miteinander verglichen werden. Für jeden Ansatz kann eine eigene Task-Ontologie verwendet werden, die allerdings einen Teil der Domänenontologie beinhalten muss, um das benötigte Wissen aus der Wissensbasis zu verwenden. Innerhalb der Domänenontologie OntHoS sind die Anforderungen an eine Aktion spezifiziert. Als Beispiel soll hier die Blinddarmoperation betrachtet werden. Die Klasse *appendectomy* ist eine Spezialisierung der Klasse *operation* und erbt von dieser sämtliche Slots[2]. Die Slots werden weiter spezialisiert, so dass die Instanzierung der Klasse *appendectomy* für den Slot *required ressources* konkrete Instanzen der Klassen *operating room, nurse, nurse, microscope* zugewiesen werden müssen. Durch die Spezialisierung der Klasse *operation* konnte zusätzliches Wissen in die Domänenontologie OntHoS eingebracht werden, welches bei der Instanzierung eines Terms der spezialisierten Klasse *appendectomy* als *hard constraint* berücksichtigt werden muss. Die Pflege der Ontologie wird mit dem Werkzeug Protégé [11] vorgenommen, wobei verschiedene Erweiterungen (BeanGenerator[3] [2], JessTab[4] [6]) hierfür zum Einsatz kommen.

Das Expertensystem Die Zuweisung der konkreten Instanzen unter Berücksichtigung der an die Slots gehefteten Facets erfolgt durch einen Scheduler, der in JESS[5] [5] implementiert wird. Die JESS Komponente spielt eine zentrale Rolle in dem System, da hier die Zuweisungen der Aktionen zu den Ressourcen vorgenommen werden und eventuelle Konflikte erkannt werden, die dann anschließend zur Lösung an das Agentensystem weitergereicht werden. JESS verfügt über eine Schnittstelle zu Protégé, über die die gesamte Ontologie und die existierenden Instanzen der ontologischen Terme importiert werden können.

[1] Die Domänenontologie OntHoS wurde innerhalb des DFG Schwerpunktprogramms 1083 "Intelligente Softwareagenten und betriebswirtschaftliche Anwendungsszenarienëntwickelt [1, 10]"

[2] Hier werden nur die Klassentypen als Facets der Slots betrachtet.

[3] Das BeanGenerator Plugin erweitert Protégé um die Fähigkeit, eine Ontologie in Java Quellcode zu exportieren.

[4] Das JessTab Plugin stellt die Schnittstelle zwischen Protégé und dem Expertensystem JESS bereit

[5] Java Expert System Shell.

Das Agentensystem Als Agentensystem wird das FIPA[6]-konforme System JADE[7] [4, 7] verwendet, das über Schnittstellen zu JESS und Protégé verfügt. Über die Schnittstelle zu Protégé können die von den Agenten benötigten Task-Ontologien importiert werden, während die Schnittstelle zu JESS für die gesamte Steuerung des Agentensystems genutzt werden kann. Hierdurch können Agenten und Agentendienste auf einer Agentenplattform gestartet und gesteuert werden, wobei die Kommunikation zwischen JESS und einem Agenten immer über einen Agentendienst vorgenommen werden muß. Das von den Agenten beigetragene Wissen zur Lösung einer Aufgabe wird an die Wissensbasis übertragen.

4 Bewertung und Ausblick

Messungen des Datenaufkommmens und der Dauer einer Verhandlung an dem Prototypen des umgesetzten Lösungsansatzes haben ergeben, dass dieser im Krankenhausbereich zur Lösung von Schedulingkonflikten eingesetzt werden kann. Der Schwerpunkt der Weiterentwicklung des Prototypen wird auf der Umsetzung von mehreren Verhandlungsstrategien für die Agenten sowie auf der Implementation des vollständigen Schedulers liegen, so daß dieser in einem realen Umfeld eingesetzt werden kann.

Literatur

[1] Becker, M., Heine, C., Herrler, R., Krempels, K.-H.: OntHoS - an Ontology for Hospital Scenarios. 2002

[2] BeanGenerator Home Page, http://www.swi.psy.uva.nl/usr/aart/beangenerator/

[3] Fitzpatrick, K.E., Baker, J.E., Dave, D.S.: An application of computer simulation to improve scheduling of hospital operating room facilities in the United States. In: International Journal of Computer Applications in Technology, Vol. 6, No. 4, p. 215-224, 1993.

[4] Jade Home Page, http://jade.cselt.it

[5] Java Expert System Shell, http://herzberg.ca.sandia.gov/jess/

[6] JessTab: Integrating Protégé and Jess, http://www.ida.liu.se/ her/JessTab/

[7] Foundation for Intelligent Physical Agents, http://www.fipa.org

[8] Fritsch, M., Wein, T., Ewers, H.-J.: Marktversagen und Wirtschaftspolitik - Mikrokonomische Grundlagen staatlichen Handelns, München 1999. S.30-32.

[9] Nareyek, A.: Constraint-Based Agents. Lecture Notes in Artificial Intelligence, Vol. 2062. Springer-Verlag, Berlin Heidelberg New York (2001)

[10] OntHoS Home Page, http://l6-spike-dos.informatik.uni-wuerzburg.de/Ontologie

[11] Protégé Home Page, http://protege.stanford.edu

[12] Schneeweiß, C.: Planung 1, Systemanalytische und entscheidungstheoretische Grundlagen, Springer-Verlag, Berlin Heidelberg New York (1991)

[6] Foundation for Intelligent Physical Agents.
[7] Java Agent Development Environment.

Integrating Transportation in a Multi-Site Scheduling Environment

Jürgen Sauer, Hans-Jürgen Appelrath
University of Oldenburg
Dept. of Computer Science
Escherweg 2, D-26121 Oldenburg
Germany
{sauer|appelrath}@informatik.uni-oldenburg.de
http://www-is.informatik.uni-oldenburg.de/~sauer

Abstract

Multi-site scheduling deals with the scheduling problems of an enterprise with several distributed production sites, where sites are using the intermediate products of other sites to manufacture the products of the enterprise. Therefore the transportation of the raw materials and the intermediate products to the plants is an important task within the whole process of manufacturing. Scheduling problems are treated on two levels. On the global level a global schedule is generated including the requirements for the local level schedulers which then have to transform the global schedule into a local schedule for manufacturing. Since transportation is a vital task in the multi-site scenario, we will view it as a scheduling task on the local level as well. Besides the "classical" objectives of transportation tasks such as finding shortest paths or minimizing costs the temporal restrictions of meeting the delivery dates here are the most important goals. In this paper we describe how transportation tasks can be modeled as a scheduling problem and which kind of solution strategies are appropriate.

1. Introduction

The main task of scheduling is the temporal assignment of activities to resources where a number of goals and constraints have to be considered. Scheduling problems can be found in several different application areas, e.g., the scheduling of production operations in manufacturing industry, computer processes in operating systems, aircraft crews, etc. Scheduling covers the creation of a schedule of the activities over a longer period (predictive scheduling) and the adaptation of an existing schedule due to actual events in the scheduling environment (reactive scheduling) [1, 2]. However, scheduling also has a very important interactive dimension because we always find humans involved in the scheduling process who have to decide, interact or control. Among the decisions to be taken by the human scheduler (the user of the scheduling system) are, e.g., introducing new orders, canceling orders, changing priorities, setting operations on specific schedule positions. These decisions have to be regarded within the scheduling process [3].

The complexity of real-world scheduling scenarios is mainly determined by

- the requirements imposed by numerous details of the particular application domain, e.g. alternative machines, cleaning times, set-up costs, etc.,
- the dynamic and uncertain nature of the manufacturing environment, e.g. unpredictable set-up times, machine breakdowns, etc.,
- conflicting organizational goals, e.g. minimize work-in-process time, maximize resource utilization, and
- the need of interaction with a human scheduler.

Additional tasks and problems arise if one looks at a multi-site production environment where hierarchical coordination and distributed scheduling is necessary. This will be described in more detail in sections 2 and 3.

Because most of the scheduling problems to be optimized have been proven to be NP-hard and due to the dynamic character of the scheduling environment the solutions proposed for real world scheduling problems rather look for feasible than optimal solutions.

Transportation problems as the other area of interest classically deal with finding cost optimal routes to deliver goods from depots to customers. Therefore the problems are formulated as combinatorial optimization tasks with a depot and a set of demands of delivery points with known distances and capacities of a fleet of vehicles. This problem is known as vehicle routing problem (VRP) and stresses the geographical aspects of transport. The vehicle routing problem with time windows (VRPTW) is an extension of the problem introducing time windows to define intervals in which the demand has to be satisfied.

Here the temporal aspect and its constraints are emphasized but the capacity restrictions of the vehicles are neglected or simplified.

If we look at transportation tasks within supply chains or in a multi-site scheduling environment with just in time demands we have transportation orders like: "load amount A1 of product P1 at location L1 between time t1 and t2 and deliver it at location L2 between t3 and t4". Now the temporal constraints are the most important constraints, but the capacity and cost constraints are valid as well in order to create economically feasible solutions. Additionally, like the scheduling tasks mentioned above, the transportation tasks have to cope with a highly dynamic environment and uncertain information.

In this paper we will look at the transportation problem from a scheduling perspective. First the transportation problem within a multi-site scheduling scenario is described. Afterwards our solution approach to transport scheduling and related approaches are presented.

2. Transportation in a multi-site scheduling environment

Usually, scheduling problems are treated in a single plant environment where a set of orders for products has to be scheduled to a set of machines [1, 4-6]. In other systems single resources, e.g., the Hubble telescope, or a set of specific transportation orders, e.g., the DITOPS system [7, 8], are tackled. However, within many industrial enterprises the production processes are distributed over several manufacturing sites, which sometimes are spread over several countries. The sites themselves are responsible for the production of various parts of a set of final products. Therefore the in time transportation of intermediates from one location to another becomes a key issue in the whole manufacturing process and thus also in the scheduling process.

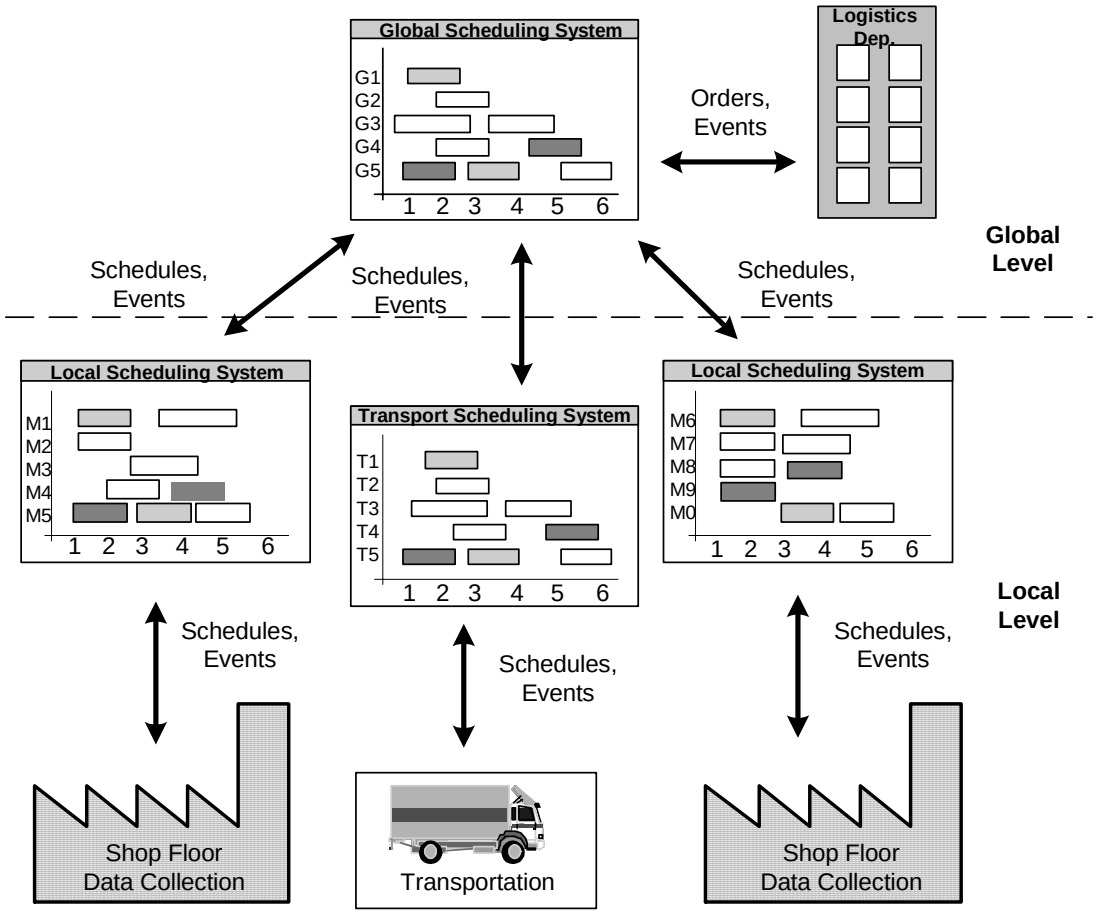

Figure 1. Multi-site scheduling with transportation scheduling

Due to the distribution of production processes to different plants and the need for coordinated scheduling some specific problems arise which have to be regarded in the scheduling algorithms. Among them are:

- production processes that are performed in different plants are related, e.g., by temporal or precedence relationships. The production process may have different costs at different sites.
- on the global level cumulative and imprecise (estimated) data are used.
- coordination and communication between the participating systems is necessary.
- different, often contrasting goals have to be regarded on the different levels, e.g., meeting due dates on the global level and minimizing work in process on the local level.
- the transportation of the intermediate products from one plant to another respectively to a depot used as intermediate stock becomes one of the important tasks in order to guarantee short lead times of production.

Additionally, in the scheduling procedures used for multi-site scheduling today there is no immediate feedback from the local plants to the logistics department and communication between the local schedulers takes place without any computer-based support.

Within our multi-site scheduling project [9] we cope with these problems and introduce a hierarchically layered system with coordinated scheduling systems for the specific scheduling tasks on the different levels. Figure 1 illustrates a hierarchical two-level structure of multi-site scheduling reflecting an organizational structure often found in business.

On the global level requirements are generated for intermediate products manufactured in individual locations. On this level the generation of a robust global schedule is very important. That means, that a schedule should be generated that gives enough flexibility for a local scheduler to react to local disturbances without affecting the other sites. This can be achieved among others by heuristics using buffer times in the time windows for local production and trying to optimize the load balancing on the machine groups or by using fuzzy techniques. Additionally, it is important to detect capacity problems as early as possible and in case of reactive scheduling, to preserve as much as possible of the existing global schedule in order to minimize the subsequent effort on the local level.

Local scheduling (at individual locations) deals with the transformation of the global schedule into concrete local production schedules which represent the assignment of operations to machines. On both levels predictive, reactive as well as interactive problems are addressed, not only to generate schedules but also to adapt them to the actual situation in the production process.

Additionally, the coordination between these tasks has to be supported in order to provide all components with actual and consistent information.

An important point within the production process is that the intermediates have to be transported between the sites. Normally, e.g., in ERP systems, only a time buffer is used to denote that the intermediate product has to be transported from one site to the other. But what if the transport fails or is delayed for any reason? The succeeding site has to know about the delay in order to reschedule its activities. And what if the product to be transported will not be ready for transport? The transport facility, too, needs information about such changes in order to reschedule the transportation tasks. Therefore it makes sense to look at transportation tasks as if they were activities to be scheduled and use the representation and problem solving techniques from scheduling to solve this problem. This also means that transportation is an integrated feature in multi-site scheduling systems and should be interpreted as a local (scheduling) site. Figure 1 shows the extended scenario with a transportation facility as a local scheduling site.

3. Solving the transportation scheduling tasks

If we look at transportation as an integrated local scheduling task in multi-site scheduling systems we can formulate the problem similar to those of the other local production schedulers. First, some of the tasks and problems will be described using a simple example.

The transportation scheduler receives a set of transportation orders with information about the intermediate product to be transported, the amount of the product, the earliest pick up date, and the latest delivery date (due date).

Table 1. Transportation orders

Order/ Product	from	to	amount	pick-up	deliver
1	A	B	100	3	5
2a	B	C	100	7	10
2b	B	C	100	7	10
3a	A	C	100	6	9
3b	A	C	100	6	9
4	B	A	100	2	4

Table 1 shows a set of four orders for the transportation of 4 products between three locations A, B, C. The time window within which the orders should be scheduled is given by "pick-up" and "deliver". The resources are the transportation vehicles which can transport a specific amount of products of a specific type, e.g., liquids or palettes. In the example we have two trucks with a capacity of 100 each. As the maximum capacity of the trucks is 100, orders 2 and 3 have been splitted to 2a, 2b, 3a, 3b. As transportation costs we use the duration of the transport:

- from A to B (B to A): 2
- from B to C (C to B): 2
- from A to C (C to A): 3.

Some additional information is not used in the example, e.g., the products to be transported can vary in size, type and weight, orders may be merged, products may be stored, there may be more technical requirements, e.g., a specific kind of transport vehicle is needed, or a specific sequence of orders is necessary.

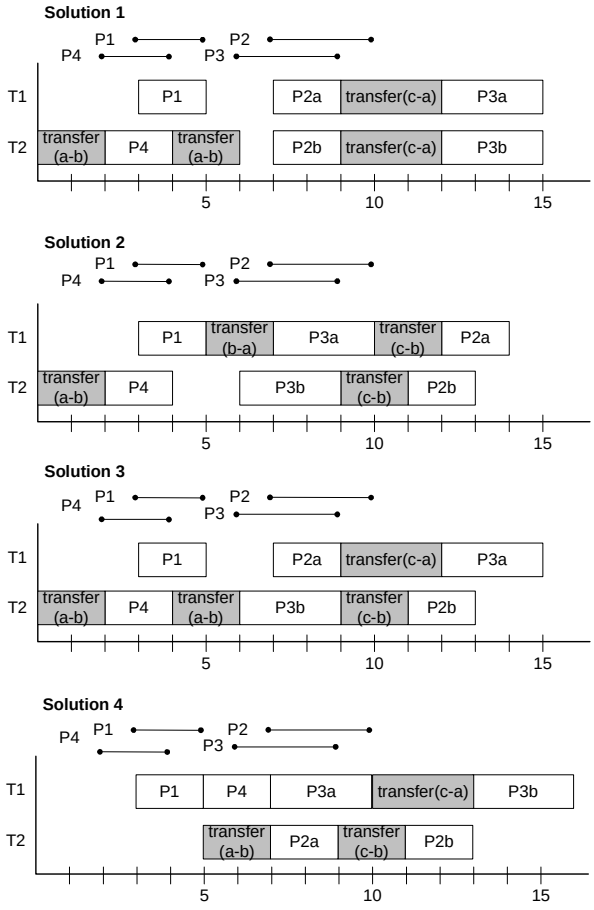

Figure 2. Example solutions

Figure 2 shows four possible solutions. We assume that the trucks are located in A at time 0. The first two solutions are calculated using an order-based approach (see below), solution 3 tries to deliver as much as possible of the products in time, solution 4 tries to minimize the transportation costs. The transfer is necessary to bring the trucks to the next location for pick up. There is no solution which fulfils all the temporal constraints. Table 2 shows some results of objective functions for the solutions, the number of late orders (products), the sum of the latenesses of the orders and the sum of the transportation costs for the schedule. The table illustrates one important problem not only of this kind of scheduling, i.e. what is the best solution and how can one find it? Solution 2 is the best one if we look at lateness, solution 3 is slightly worse but part of the products will be delivered in time. Is it therefore better than solution 2? Solution 4 is the best when considering transportation costs, but 4 of the six products will be late. Experience shows that it mainly depends on the problem situation and the user of the system what will be accepted as the best solution. Additionally, if we have to cope with reactive scheduling then feasible solutions and user interaction will become more important.

Table 2. Evaluation of the results

solution	late orders	lateness	transport costs
1	2 (1 product)	12	24
2	3 (2 products)	8	22
3	2 (2 products)	9	23
4	4 (3 products)	14	21

For a solution approach it is necessary to describe what information is needed to model the problem and how it is used. Therefore the following information should be available at the local level for a transportation order to determine a detailed transportation schedule:

- a set of transport activities for products. Each activity is accompanied with information, e.g., about the type of product, appropriate vehicles, the estimated duration and route of the transport.
- a set of vehicles as resources. For each vehicle we know, e.g., the capacity, the type and the range. Coupled with the vehicles are the drivers who can also be seen as resources.

Of importance is also information about stocks at the production sites or elsewhere because this allows or forbids delays. This and other necessary information may be represented by hard and soft constraints, e.g.,

- as hard constraints
  - meet user requirements, e.g., orders already scheduled by the user,

  - meet the technical requirements, e.g. the capacity of the vehicles, the need for a specific kind of transport vehicle, or existing capacity restrictions for some routes.
- as soft constraints
  - realize a just in time delivery,
  - meet the due dates respectively time windows of the transportation orders,
  - use preferred routes,
  - optimize the vehicle (capacity) usage,
  - optimize transportation costs.

We should keep in mind that the last two goals are even goals of the classical vehicle routing problems. If we look at the predictive scheduling task it seems to be possible to include features of the problem solving approaches from transportation research, e.g., from vehicle routing problems with time windows [10]. But if we look at the dynamic environment in which the transportation scheduling is integrated, it seems likely that reactive scheduling will be often the case in transportation scheduling, too. The environment consists of imprecise global schedules where new orders are introduced dynamically and several local scheduling systems that have to react to all kinds of events. The events one has to deal with in the transportation domain are amongst others:

- changes of local resources, e.g., breakdowns, maintenance,
- delays in transport due to traffic conditions such as deviations, traffic jams,
- changes in orders for transportation activities,
- user interactions.

Additional characteristics of the transportation tasks that have to be handled when solving the problem include the division of orders into suborders, a dependence of the number of necessary vehicles on the order amount and several routing decisions, e.g., if and when a vehicle should be rerouted or which orders can be handled by one vehicle.

Thus, we will have to look for good heuristics that will produce feasible solutions or proposals for schedules in a short amount of time. The solution does not need to be optimal in terms, e.g., of production costs, but feasible especially regarding the temporal constraints. Additionally, user interaction should be possible, e.g., for fixing some activities to specific vehicles or time intervals, and the system has to deal with the events of the dynamic environment. This functionality has to be incorporated in the scheduling system (not only for the transportation problem). Therefore we favor an approach that produces feasible solutions and checks constraint violations by the user. The approach bases on AI techniques for modeling and problem solving and will be presented in the remainder of this section starting with the modeling of the problem, then presenting a heuristic for the creation of a schedule and some remarks on other features of the system like rescheduling and user interaction.

Table 3. Modeling global, local and transportation scheduling

	Local Scheduling	Transportation Scheduling	Global Scheduling
R	machines	transportation vehicles with capacity and other restrictions	groups of machines
P	intermediate products consisting of several production steps (operations)	transport of intermediate products using specific transportation vehicles	final products consisting of several intermediate products
O	internal orders for intermediates	internal orders for intermediates	external orders for final products
HC	schedule all orders, regard production requirements (one variant, precedence constraints)	schedule all orders, regard technical requirements (type of vehicle, transport capacity)	schedule all external orders, regard production requirements (one variant, precedence constraints, capacity)
SC	"optimal" machine utilization, meet due dates, minimize work-in-process costs.	meet due dates, "optimal" vehicle utilization, minimize costs.	meet due date, minimize transportation times/ costs, use production equally, reduce inventory costs.

Global, local and transportation scheduling problems can be modeled similarly by the five-tuple (R, P, O, HC, SC) [11], where R denotes the set of required resources, P the set of producible products, O the set of actual orders, and HC and SC stand for the sets of hard and soft constraints, respectively. Table 3 shows this model applied to global, local and transportation scheduling with examples for the items.

The actual data of the R, P and O sets are typically stored in a database, the constraints have to be handled by the problem solving component. In the rule-based approach presented below the constraints are incorporated in the selection rules and the control constructs of the algorithm.

Similar to the representation it is also possible to transfer algorithmic approaches from knowledge-based production scheduling to the domain of transportation scheduling. Because it will not be possible to find the one and only algorithmic solution for all the features of the transportation scheduling problem described above, several strategies have to be checked and an appropriate one should be choseable by the user together with manual scheduling. To describe (and implement) several strategies we adopt an approach from [11] with which scheduling algorithms can be built dynamically by combining strategies represented as skeletons with selection rules, e.g., heuristics for orders, resources, intervals etc. A simple order-based heuristic strategy similar to those used in scheduling production could be as shown in figure 3. Resource-based or time-based strategies are possible as well.

```
BEGIN
WHILE transportation orders to
      schedule
       select order
       select possible transportation
            vehicle
       select time interval
       IF possible THEN schedule it
       ELSE solve_conflict.
END WHILE
optimize schedule
END
```

Figure 3: Order-based heuristic for transportation scheduling

Heuristic knowledge of the domain as well as approaches from knowledge-based scheduling [12] and operations research like the savings-heuristic [13] can be used within the *select* statements and the *solve_conflict* statement. Possible rules for selection and conflict resolution are:

- **select orders** by earliest due date or slack rule or importance (user given priority)
- **select vehicle** by first fit or best fit regarding capacities or by using bottleneck resources first or by looking at the route already scheduled (looking for the nearest route passing the start/ destination)
- **select time interval** in a just in time manner (backward from due date) or forward from earliest pick up
- **conflict resolution** may incorporate looking for alternative intervals, alternative transport vehicles or user driven decisions like outsourcing of transportation orders to external carriers.

In the optimization phase (*optimize schedule*) it should be possible to use one or more strategies to improve the schedule in order to optimize other goals like minimal costs. This could be done, e.g., like in DITOPS by splitting and merging routes [8]. Iterative improvement techniques may also be a promising approach [14].

In the case of a necessary reaction due to events like vehicle breakdowns, parts of the strategy presented above can be used to find alternatives, e.g., as in *conflict resolution*. Important is the possibility of user interaction, e.g., for proposing alternatives or changing orders or capacities.

The system architecture is based on the common architecture of all the local and global scheduling systems within our MUST project (see figure 4). The user interface is adopted from the global scheduling area presenting two views of the transportation schedule, an order based perspective showing the orders and the resources used for transportation, and a resource based view showing the resources and the products they have to transport. Because communication is an important feature of the multi-site scheduling system it is provided also for the transportation scheduling system, which is located on the local level.

The following events can be communicated between the levels:

- the global schedule showing the transportation orders,
- changes in the global orders,
- the local realization of the global orders,
- events of the local level important to the global scheduler because they need rescheduling effort, e.g., breakdowns of resources.

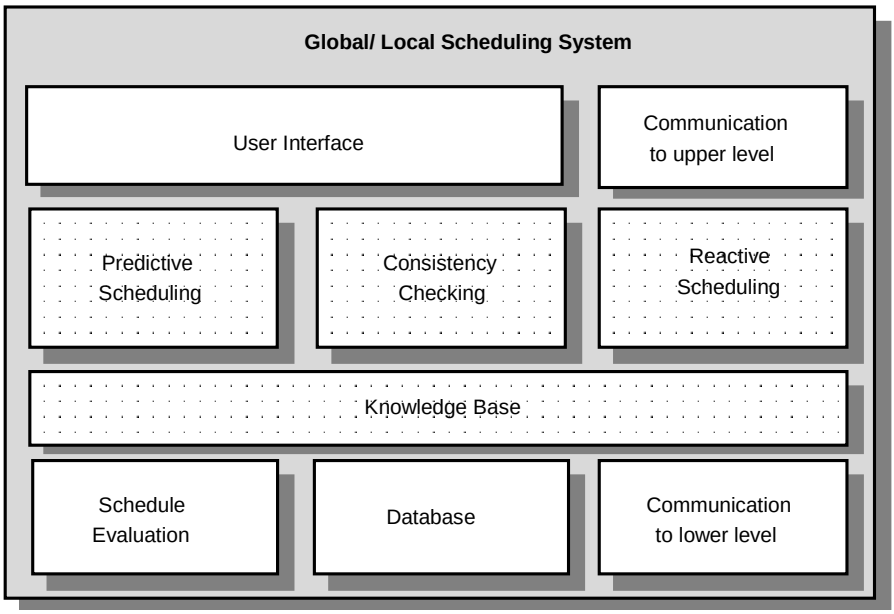

Figure 4: Common architecture of scheduling system

4. Related work

In Operations Research most of the approaches presented deal with vehicle routing problems, where optimal solutions regarding some cost function have to be found. Thus the techniques used only look at the geographical aspects of the problems. Extensions using time windows for delivering and uphauling products are called vehicle routing with time windows or vehicle scheduling problems. Here the approaches concentrate on the temporal constraints often neglecting the capacity restrictions of the vehicles and the distances between the locations. As most of the problems belong to the NP-hard problems newer solution approaches are using constraint programming [15], heuristics [16], and genetic algorithms [10] to find near optimal solutions.

Using the approaches from above the dynamic nature of the transportation problems as well as the changing environment can not be taken into consideration. They seem to be better suited for the predictive case of finding routes or schedules for specific or standard problems.

Solutions inspired by the work in AI can be divided into two categories. One consists of systems that use an agent based paradigm where a set of cooperating agents is used to model the transportation problem and to solve the transportation scheduling tasks. Among these are the MARS system [17], a system using partial intelligent agents [18] and the TELETRUCK system [19]. Important differences are in what the agents are representing and what kind of responsibilities they have, e.g., in the MARS system two groups of agents - truck agents and transportation company agents - are cooperating via an extension of the contract net protocol to solve the transportation tasks. The companies do not have scheduling facilities, the actual schedule for the whole company is spread over the truck agents and maintained by them. Thus the agent based paradigm provides a more control oriented, reactive view of the problem and its solution.

The second group of systems are intended as decision support systems and use heuristics to find initial and reactive solutions for specific transportation tasks, e.g. the DITOPS system [20]. In the DITOPS system a mixed initiative approach is used including a constraint-based scheduling system and user interaction for proposing alternatives whenever the problem is overconstrained and no valid solution can be found. The approach presented here belongs to this second category.

5. Conclusion and future work

It has been shown that, especially within a multi-site scheduling environment, it is possible to treat transportation problems as scheduling problems. A representation formalism and a heuristic strategy for solving transportation scheduling problems have been presented.

The work is included in the distributed knowledge-based scheduling system MUST [9] which has been designed to support the human experts in the management of the dynamic distributed manufacturing environment, in

particular in scheduling the appropriate distribution of the orders to the different manufacturing plants as well as in coordinating the decentralized scheduling activities for all plants within one enterprise. The objective of this approach is the reduction of complexity of distributed scheduling and improving the quality of the solution at the same time. The MUST system consists of one global scheduling subsystem and several local subsystems, one for each individual production site. Common features of all subsystems of the multi-site approach are:

- All components are based on knowledge-based techniques, i.e. problem-specific knowledge is identified, represented, and applied for the solution of the addressed problem.
- Several problem solving techniques have been investigated for use in the scheduling components.
- The reactive scheduling components on both scheduling levels are realized as a leitstand with a sophisticated graphical user interface allowing interactive scheduling.
- The user interfaces are window-oriented and most functions are mouse-sensitive.
- Each subsystem contains two communication interfaces for the information exchange within MUST and the integration of the MUST system into an existing organizational environment.

An early prototype was implemented in PROLOG. We now work on a redesign using object oriented features and JAVA as implementation language.

Within the transportation scheduling system several strategies shall now be implemented and evaluated using example or benchmark problems if available. Another approach investigated in our working group uses agent-based technology to solve the multi-site scheduling problem with a multi agent system including an multi agent approach for the transportation problems. This approach will also be compared with the "simple" heuristic one presented here.

The results of the work on multi-site scheduling can be adapted to the whole supply chain or to other application areas, e.g., distributed software development or project management. This will be one of our future research areas.

References

[1] S. F. Smith, “Knowledge-based production management: approaches, results and prospects,” *Production Planning & Control*, vol. 3, 1992.

[2] R. M. Kerr and E. Szelke, *Artificial Intelligence in Reactive Scheduling*: Chapman & Hall, 1995.

[3] W. L. Hsu, M. Prietula, G. Thompson, and P. S. Ow, “A mixed-initiative scheduling workbench: Integrating AI, OR, and HCI,” *Journal of Decision Support Systems*, vol. 9, pp. 245-247, 1993.

[4] J. Dorn and K. A. Froeschl, *Scheduling of Production Processes*: Ellis Horwood, 1993.

[5] J. Sauer and R. Bruns, “Knowledge-Based Scheduling Systems in Industry and Medicine,” *IEEE-Expert*, 1997.

[6] M. Zweben and M. S. Fox, *Intelligent Scheduling*: Morgan Kaufman, 1994.

[7] S. F. Smith, O. Lassila, and M. Becker, “Configurable, Mixed-Initiative Systems for Planning and Scheduling,” in *Advanced Planning Technology*, A. Tate, Ed.: AAAI Press, 1996.

[8] M. A. Becker, “Reconfigurable Architectures for Mixed-Initiative Planning and Scheduling,” Carnegie Mellon University, Pittsburgh, USA, Dissertation 1998.

[9] J. Sauer, “A Multi-Site Scheduling System,” presented at AAAI's Special Interest Group in Manufacturing Workshop on Artificial Intelligence and Manufacturing: State of the Art and State of Practice, Albuquerque, 1998.

[10] J. Homberger, “Zwei Evolutionsstrategien für das Standardproblem der Tourenplanung mit Zeitfensterrestriktionen,” in *Betriebswirtschaftliche Anwendungen des Soft Computing*, J. Biethahn and e. al., Eds. Braunschweig: Vieweg, 1998.

[11] J. Sauer, “Meta-Scheduling using Dynamic Scheduling Knowledge,” in *Scheduling of Production Processes*, J. Dorn and K. Froeschl, Eds.: Ellis Horwood, 1993.

[12] J. Sauer, “Knowledge-Based Scheduling Techniques in Industry,” in *Knowledge-Based Intelligent Techniques in Industry*, L. C. Jain, R. P. Johnson, Y. Takefuji, and L. A. Zadeh, Eds.: CRC Press, 1999, pp. 53 - 84.

[13] K. Neumann and M. Morlock, *Operations Research*. München: Hanser, 1993.

[14] J. Dorn, "Iterative Improvement Methods for Knowledge-Based Scheduling," *AICOM*, vol. 8, pp. 20-34, 1995.

[15] N. Christodoulo, E. Stefanitsis, E. Kaltsas, and V. Assimakopoulos, "A Constraint Logic Programming Approach to the Vehicle Fleet Scheduling Problem," presented at Second International Conference on the Practical Application of Prolog, London, 1994.

[16] S. R. Thangiah, J. Y. Potvin, and T. Sun, "Heuristic Approaches to Vehicle Routing with Backhauls and Time Windows," *International Journal of Computers and Operations Research*, pp. 1043 - 1057, 1996.

[17] K. Fischer, J. P. Müller, and M. Pischel, "Cooperative Transportation Scheduling: An Application Domain for DAI," *Journal of Applied Artificial Intelligence*, vol. 10, 1996.

[18] K. Fischer and N. Kuhn, "A DAI Approach to Modeling the Transportation Domain," DFKI, Saarbrücken, Research Report RR-93-25, 1993.

[19] H.-J. Bürckert, K. Fischer, and G. Vierke, "Transportation Scheduling with Holonic MAS - The TELETRUCK Approach," presented at Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM 98), London, 1998.

[20] S.-F. Smith and O. Lassila, "Toward the Development of Flexible Mixed-Initiative Scheduling Tools," presented at ARPA ROME Laboratory Planning Initiative Workshop, Tuscon, AZ, USA, 1994.

SCHEDULING ZUR AUTOMATISIERUNG VARIABLER LABORABLÄUFE

Schulz A., Sack M., Hortig J.
Fraunhofer Institut Fabrikbetrieb und -automatisierung,
Sandtorstraße 22, D-39106 Magdeburg
e-mail: aschulz@iff.fhg.de
Tel.: +49 (0) 391/ 4090-219
Fax: +49 (0) 391/ 4090-250

1. Einleitung

In den vergangenen Jahren ist die Bedeutung der Biotechnologie ständig gewachsen. Forschungsschwerpunkte der Biotechnologie sind unter anderem die Entschlüsselung der Wirkungsweise komplexer biologischer Prozesse auf molekularer und zellulärer Ebene und darauf aufbauend die Entwicklung von Wirkstoffen und Bioprodukten. Krankheiten bei Menschen, Tieren und Pflanzen sollen gezielt bekämpft werden. Der Weg zu neuen Wirkstoffen und Produkten auf diesen Gebieten ist teuer und sehr zeitaufwendig. Bereits im Forschungsstadium stellt sich daher die Frage nach der Automatisierbarkeit umfangreicher, teurer, monotoner und fehleranfälliger manueller Tätigkeiten.
In einige Bereichen der biotechnologischen Forschung ist die Automatisierung bereits stark vorgedrungen. Die wesentlichste Rolle spielt hierbei das High Throughput Screening (HTS). HTS-Anlagen übernehmen die automatisierte Durchführung von immer wiederkehrenden Versuchsabläufen im Bereich des Liquid Handling.
Die Möglichkeiten des Einsatzes von moderner Automatisierungstechnik und automatisierten Anlagen im Bereich von Forschung und Entwicklung der Biotechnologie gehen jedoch weit über die Beschränkungen des auf dem Liquid-Handling basierenden HTS hinaus. Sie reichen bis zur intelligenten und dynamischen Verteilung und Durchführung von wechselnden oder variierenden Testverfahren. Viele Arbeitsabläufe im Laborbereich werden immer noch manuell durchgeführt, gerade dann, wenn die durchgeführten Testverfahren durch hohe Varianz in der Probenbehandlung gekennzeichnet sind.

Am Fraunhofer IFF wird derzeit eine automatisierte Wirkstoff-Screening-Anlage entwickelt und realisiert. Die Anlage stellt die Automatisierung eines neuartigen Verfahrens zur Erprobung von innovativen Wirkstoffen an Zellgewebekulturen dar. Eine Biotechnologie-firma entwickelte dieses Verfahren und etablierte es bereits in ausschließlich manueller Ausführung.
Der geringe Probendurchsatz läßt momentan jedoch nur eine geringe Anzahl an Tests von neuen Wirkstoffen und Wirkstoffkombinationen zu. Für eine effizientere Arbeitsweise und die Weiterentwicklung des neuen Testverfahrens bedurfte es einer automatisierten Versuchsdurchführung.
Trotz des hohen Automatisierungsgrades der Anlage müssen einige Tätigkeiten des Verfahrens manuell ausgeführt werden, die den nachfolgenden Versuchsablauf maßgeblich beeinflussen. Dies erfordert eine ständige Veränderung und Anpassung der Testabläufe einzelner Versuchsobjekte. Die manuelle Anforderung neuer Abläufe bzw. Änderung der bereits eingeplanten Abläufe soll zu jeder Zeit möglich sein und in Echtzeit eingeplant werden. Für erfolgreiche Test von Wirkstoffen gelten zudem hohe Anforderungen an die Einhaltung der Termine einzelner Prozessschritte. Aus diesen Gründen entschieden wir, einen eigenen Algorithmus (Scheduler) für eine Ablaufplansteuerung zu entwickeln und zu implementieren.

Anlagendesign und Arbeitsablauf

Für die Tests der Wirkstoffe werden 6-Lochplatten als Probenträger benutzt. Eine 6-Lochplatte (Multischale) enthält 6 Vertiefungen (Wells). In jedes Well kann ein Membraneinsatz eingelegt werden. Die Membraneinsätze werden mit Zellkulturen bestückt. Die Anlage ist aus mehreren Stationen aufgebaut. Der Transport und das Handling der Multischalen zwischen den Stationen erfolgt durch einen Transportroboter. Die Stationen können folgende Aktionen durchführen:

Inkubator #1 und #2.....	Kultivierung der Testobjekte
Inkubator #3.....	Einbringen der Testobjekte in eine definierte Testatmosphäre
Pipetierstation.....	Wechseln der Nährstofflösungen Zugabe von Testsubstanzen Umsetzen von Membraneinsätzen in eine andere Multischale
Mikroskopstation.....	automatische Bildaufnahmen
Vorratsmagazine.....	Lager für neue Multischalen
Trash.....	Auswurf von Multischalen
Arbeitsstation.....	Arbeitsplatz, an dem manuelle Tätigkeiten an den Zellkulturen ausgeführt werden können
Transporter.....	transportiert die Multischalen zwischen den Stationen

Struktur der Abläufe

Für das Planen der automatisierten **Versuchsabläufe** werden diese in **Struktureinheiten** zerlegt. Die oberste Einheit ist eine **Sequenz**. Eine Sequenz bezieht sich auf eine oder mehrere Multischalen. Eine Multischale ist jedoch immer genau einer Sequenz zugeordnet. Die nächst tieferen Struktureinheit bezeichnen wir als **Vorgang**, **Job** und **Operationen**. In Abbildung 1 ist die allgemeine Struktur gezeigt.

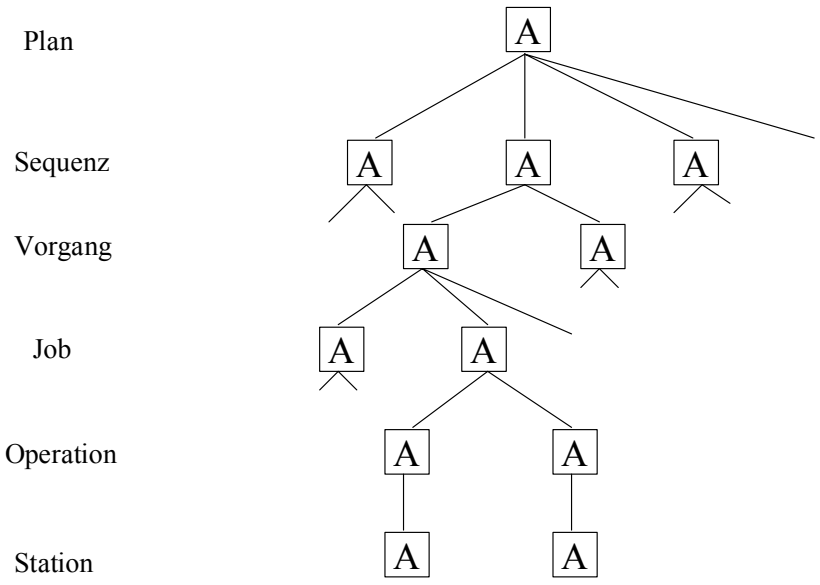

Abbildung 1: UND/ODER Baum der Ablaufplanstruktur

Ein Vorgang ist eine logische Einheit aus Sicht des Bedieners. Zum Beispiel ist die Zugabe einer Testsubstanz innerhalb der Pipetierstation ein Vorgang. Dieser Vorgang besteht aus 3 Jobs:

Job#1: Transport der Multischale aus einem Inkubator zur Pipetierstation
Job#2: Zugabe der Testsubstanz
Job#3: Transport der Multischale von der Pipetierstation in einen Inkubator

Der Bediener kennt die einzelnen Jobs im Hintergrund nicht. Die Jobs sind in einer Bibliothek hinterlegt, werden automatisch zusammengefügt und können für verschiedene Vorgänge verwendet werden. Der Bediener kann durch die Verkettung mehrerer Vorgänge einen Versuchsablauf zusammenstellen.
Jeder Job besteht aus mindestens einer **Operation**. So setzt sich der obige Transportjob z.B. aus den folgenden drei Operationen zusammen:

OP#1: Ausschleusen der Multischale aus dem Inkubator#1
OP#2: Transport der Multischale zur Pipetierstation
OP#3: Einschleusen der Multischale in die Pipetierstation

Jede Operation wird dabei an genau einer Station ausgeführt.

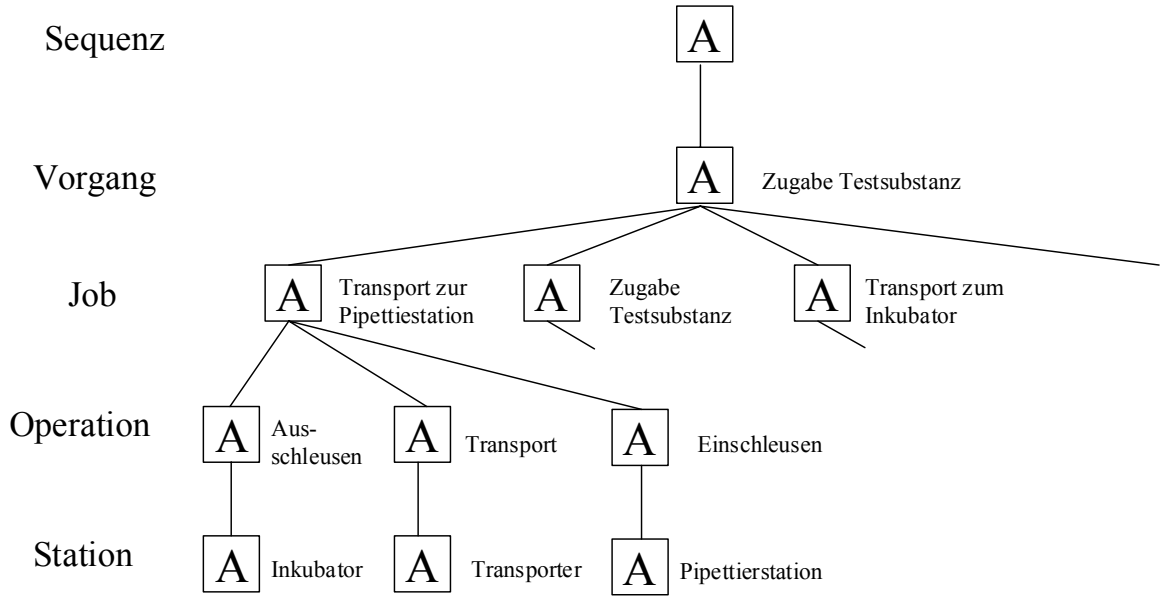

Abbildung 2: UND/ODER Baum des Vorgangs Zugabe von Testsubstanz

Der Scheduler arbeitet auf Ebene der Operationen. Damit wird der zeitlich parallele Ablauf von Jobs verschiedener Vorgänge möglich. Die Operationen werden im voraus in einen bestehenden Zeitplan geplant. Die Software liest ständig den Zeitplan und übergibt den Stationen die aktuell fälligen Operationen zur Ausführung.

Parameter und Datenstrukturen des Schedulers

Ein Ablauf stellt sich für den Scheduler als Abfolge von Operationen dar. Die Operationen haben folgende Parameter:

Startzeit	t_s
Dauer	t_d
Abstand	t_a
minimaler Abstand	t_{min}
maximaler Abstand	t_{max}
Deadline	d
Änderung Kapazität	δ_{Kap}

Die Startzeit bestimmt den geplanten Startzeitpunkt der Operation. Der Startzeitpunkt der ersten Operation ist zugleich der Startzeitpunkt der Sequenz.
Der zeitliche Abstand gilt jeweils zwischen zwei aufeinanderfolgenden Operationen einer Sequenz. Abstände zwischen nicht aufeinanderfolgenden Operationen der gleichen Sequenz werden bislang nicht berücksichtigt. Zwischen Operationen verschiedener Sequenzen werden bislang ebenfalls keine Abhängigkeiten definiert.
Der Abstand zwischen den Operationen bewegt sich innerhalb eines vorgegebenen Toleranzbereiches. Der Toleranzbereich wird durch das Zeitfenster t_{min} und t_{max} festgelegt.
Die Deadline einer Operation benennt den spätest erlaubten Startzeitpunkt der Operation. Die Änderung der Kapazität legt fest, ob die Operation die Kapazität der Station um jeweils eine Multischale erhöht bzw. verringert oder die Kapazität der Station nicht beeinflusst. Die Stationen besitzen jeweils eine maximale Kapazität an Multischalen.

Die Operationen einer Sequenz sind in ihrer Reihenfolge festgelegt. Es gilt

$$t_{s(i+1)}=t_{si}+t_{di}+t_{ai} \qquad \text{i Nummer der Operation in der Sequenz}$$

Es gilt immer $t_{s(i+1)}>t_{si}$. Der Abstand t_s ist zumeist positiv, kann aber auch negative Werte annehmen. Dies wird genutzt, um eine Operation zu starten, während eine andere Operation der gleichen Sequenz in Bearbeitung ist. So kann der Transporter schon an die Station fahren , an der er als nächstes eine Multischale aufnehmen soll. Dennoch darf die Bedingung $t_{s(i+1)}>t_{si}$ nicht verletzt werden.

Algorithmen des Schedulers

Eine wichtige Bedingung für die Entwicklung des Schedulers war die unbedingte Minimierung der benötigten Rechenzeit des Algorithmus. Somit ergab sich eine möglichst geringe Komplexität. Es wurde daher nicht angestrebt eine Zielfunktion zu optimieren, der Scheduler sollte vielmehr sehr schnell eine gültige, suboptimale Lösung finden.

Die Planung einer dem Scheduler übergebenen Sequenz erfolgt an einem bestehenden Zeitplan und kann zu jedem Zeitpunkt erfolgen. Dabei werden sowohl Sequenzen geplant, die erst Stunden bzw. Tage nach der Einplanung starten, als auch Sequenzen, welche unmittelbar nach der Planung ausgeführt werden sollen, z.B. um kurzfristig manuelle Tätigkeiten an einer Multischale durchzuführen. Daher ist die Rechenzeit für den Scheduler begrenzt. Die Planung ähnelt den Plankorrekturverfahren. Die bereits bestehenden Sequenzen im Zeitplan werden durch die derzeitigen Version des Schedulers während eines Planungsvorganges nicht mehr verändert.

Derzeit wird am IFF weiterführend an einem Verfahren gearbeitet, welches es erlaubt, kleine Änderungen an bestehenden Sequenzen im Zeitplan vorzunehmen, ohne dass dadurch der gesamte Zeitplan neu geplant werden muß.
Es ist auch möglich, mehrere einzuplanende Sequenzen an den Scheduler zu übergeben. Dieser verwendet das einfache FIFO Prinzip, um die Reihenfolge der Planung festzulegen. Dabei wird jedoch vorab geprüft, ob die betreffende Multischale nicht bereits von einer anderen Sequenz im gleichem Zeitbereich belegt ist.

Das Einordnen der Operationskette einer Sequenz in den Zeitplan ähnelt dem Tiefensuchverfahren mit Backtracking. Zusätzlich existieren harte Randbedingungen, die erfüllt sein müssen. Diese sind:

- der Abstand zur vorherigen Operation ist kleiner als t_{max}
- der Abstand zur vorherigen Operation ist größer als t_{min}
- die Kapazität einer Station ist kleiner als die maximale Kapazität
- die Deadline der Operation ist nicht überschritten

Für die erste Operation wird das nächste freie Zeitfenster ab ihrem Startzeitpunkt ts auf der betreffenden Station gesucht. Anschließend werden die Randbedingungen geprüft. Sind die Bedingungen erfüllt, wird die nächste Operation betrachtet. Ist dagegen die Deadline überschritten, wird das Einplanen abgebrochen. Ist die Kapazität zu klein, dann wird das nächste freie Zeitfenster gesucht und die Bedingungen für diese Operation erneut geprüft. Wenn der Abstand zur vorherigen Operation zu groß ist, dann wird der Wert der unerlaubten Abweichung ermittelt und die vorherige Operation wieder betrachtet. Für diese Operation wird der Startzeitpunkt um die Abweichung verschoben. Anschließend wird mit der neuen Startzeit für diese Operation erneut das nächste freie Zeitfenster bestimmt und die Randbedingungen geprüft. Dieses rekursive Verfahren wird so lange fortgesetzt, bis eine gültige Lösung gefunden bzw. eine Deadline, wenn vorhanden, überschritten wurde.

6. Ausblick

Mit dem realisierten Scheduler konnten im realen Praxisbetrieb bisher noch keine langfristigen Erfahrungen gesammelt werden, da sich die Labor-Anlage derzeit noch in der Inbetriebnahmephase befindet. Somit kann noch keine Aussage dahingehend getroffen werden, ob der von uns realisierte Optimierungsalgorithmus die an die Anlage gestellten Forderungen vollends erfüllen kann.

Für zukünftige Projekte, bzw. zur weiteren Optimierung der bestehenden Automatisierungsanlage soll der Planungsalgorithmus in seiner Leistungsfähigkeit erweitert werden. Durch die Nutzung weiterer Parameter wie z.B.

- Beziehungen zwischen allen Operationen innerhalb der gleichen Sequenz
- Beziehungen zwischen Operation verschiedener Sequenzen

sollen komplexe Abläufe einfacher und zeitlich besser optimiert planbar werden. Durch Einführung von verschiedenen Varianten für Jobs oder Vorgänge kann die Modularität und Erweiterbarkeit einer Anlage erhöht werden. Weiterhin sollen verbesserte Verfahren für die Plankorrektur zum Einsatz kommen. Dies ist auch für die Änderung des Zeitplans im Fehlerfall wichtig.
Darüber hinaus sollen andere Verfahren aus der KI eingesetzt und getestet werden. Dazu gehören Bottleneck- Verfahren oder der Einsatz von Suchverfahren mit Heuristiken. Aber auch genetische Verfahren, Constrainbasierte Verfahren oder wissensbasierte Methoden.

Design Problem Solving by Functional Abstraction

Benno Stein

Paderborn University
Dept. of Computer Science
D-33095 Paderborn, stein@upb.de

Abstract The paper introduces the Functional Abstraction paradigm as a concept to solve complex design problems.

Design problem solving can be considered as selecting an appropriate model from a space of possible models . By Functional Abstraction a simplified view , , onto the design space is constructed. I. e., Functional Abstraction follows a model simplification strategy in order to make a synthesis problem tractable.

Aside from a description of the paradigm the paper outlines two design problems where the paradigm of Functional Abstraction has been applied.

1 Introduction

In order to make this paper self-contained to a certain degree, the introductory section presents the definitions that are used later on.

1.1 Systems and Models

A system, , can be considered as a clipping of the world and has, as its salient property, a boundary. On account of this boundary, it can be stated for each object of the world whether or not it is part of .[1] Models are the essential element within the human ability to reason about systems. Numerous definitions, explanations, and introductions have been formulated about the term model—my favorite definition stems from Minsky:

> *"To an observer B, an object is a model of an object to the extent that can use to answer questions that interest him about ."*
>
> Minsky, 1965, pg. 45

Here,

- the interesting objects, , are technical systems,
- the observer, , is a domain expert who works on a problem solving task from the field synthesis, such as a configuration or design task,
- the questions are embedded in a ternary way; they relate to the technical system, to the problem solving task, and to the domain expert,
- the models, , are exactly that what Minsky has pointed out above.

[1] This characterization of the term system is derived from Gaines's definition [5], which, like other definitions, can be found in the well written introduction of Cellier's book on continuous system modeling [2].

1.2 Synthesis Tasks and Model Spaces

Our starting point when solving a synthesis task is characterized as follows. We are given a set of systems, , called the system space, along with an open question . The question may ask whether a system with a desired functionality (= demands) exists, say, is member of , or how much a system with a desired functionality at least costs. Answers to such questions can be found by constructing the desired system and analyzing its functionality.

Clearly, constructing a system solely for answering an open question cannot be accepted in the very most cases. A way out is the creation of a bijective function, , that maps each system onto a model M in a *model space* (see Figure 1). Usually the model space is described intensionally, by means of a finite number of combinable objects along with operations that prescribe how objects can be connected to each other.

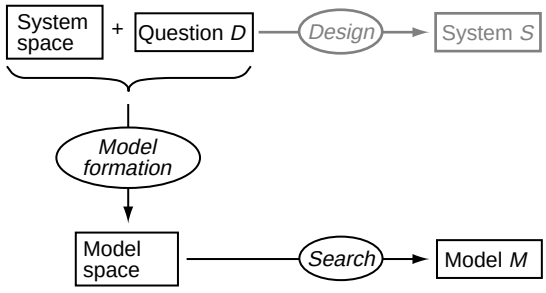

Figure 1: Synthesis problems are solved by means of a systematic search in the model space—provided that a bijective mapping from the system space onto the model space can be stated.

A computer program for automated design or configuration realizes a function : that maps a set of demands onto a model M, ( ) M. Put in a nutshell, synthesis problems are solved by developing a systematic search strategy that turns a model space into a search space [12].

1.3 Structure Models and Behavior Models

Our view to models and model construction is bivalent: it is oriented at structure and behavior (see Figure 2).

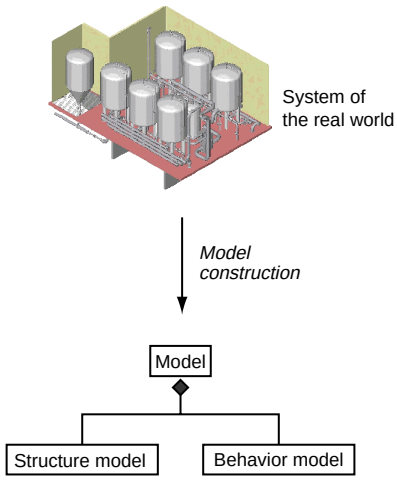

Figure 2: Basically, model construction divides into structural and behavioral considerations.

A structure model renders the structural or topological setup of a system. A behavior model reproduces, in extracts, a system's behavior. In this connection, model formation relates to both structural composition and behavior specification.

Structure models are a powerful means to define a model's composition space—without committing the level of abstraction at which a technical system is described. Note that a structure model says nothing about the models type or purpose, whether it establishes a qualitative model, a dynamic behavior model, or some other model. Typically, an infinite number of behavior models can be associated with the same structure model.[2]

In the following, let M define the structure model of some model M in question.

2 Functional Abstraction

The term model simplification speaks for itself: By reducing a model's complexity a problem solving task in question shall become tractable. Functional Abstraction is a model simplification paradigm that aims at a substantial reduction of the number of models that have to be synthesized and analyzed in order to solve a configuration or design problem.

The two case studies sketched out in Section 3 present approaches to solve complex design tasks. Both tasks comprise creative aspects, and for neither a design recipe is at hand: The acquisition effort for the design knowledge exceeds by far the expected pay back [10], and, moreover, the synthesis search spaces are extremely large and scarcely to control—despite the use of knowledge-based techniques.

[2]In this place a precise and formal definition of structure models and behavior models should be given [16]. However, I continue without doing so since (1) most readers may have a more or less clear understanding of these terms, and (2) a formal definition would go beyond the scope of this paper.

Two possibilities to counter this situations are "competence partitioning" and "expert critiquing". The idea of competence partitioning is to separate the creative parts of a design process from the routine jobs, and to provide a high level of automation regarding the latter (see [15, pg. 93]). Expert critiquing, on the other hand, employs expert system technology to assist the human expert rather than to automate a design problem in its entirety [6, 4].

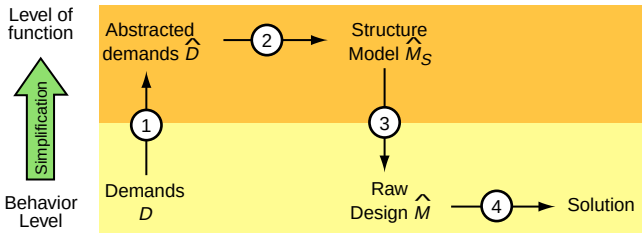

Figure 3: The paradigm of Functional Abstraction in design problem solving.

The paradigm of "Design by Functional Abstraction", illustrated in Figure 3, can be regarded as a special expert critiquing representative. We have chosen this name for the problem solving method to reveal its similarity with the problem solving method HEURISTIC DIAGNOSIS, which became popular as the diagnosis approach underlying MYCIN [3].

> The key idea of Design by Functional Abstraction is a systematic construction of candidate solutions within a very simplified design space , , which typically is some structure model space: The structure model of a solution candidate, M , is transformed into a preliminary raw design, M, by *locally* attaching behavior model parts to M . The hope is that M can be repaired with reasonable effort, yielding an acceptable solution for .

Design by Functional Abstraction makes heuristic simplifications at least at two places: The original demand specification, , is simplified towards a functional specification (Step 1 in Figure 4), and, M is transformed locally into M (Step 3 in Figure 4). Both, the synthesis step and the adaptation step may be operationalized with complete algorithms (Step 2 and 4 in the figure).

Putting it overstated the paradigm says: At first, we construct a poor solution of a design problem, which then must be repaired.

The solutions in the following case studies were developed after this paradigm.

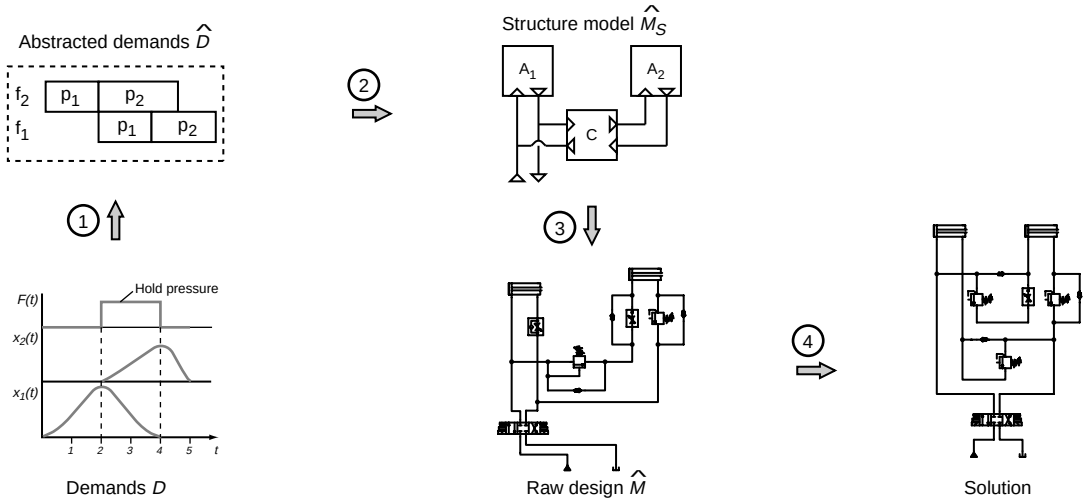

Figure 4: The Functional Abstraction paradigm applied to fluidic circuit design.

3 Case Studies

This section introduces two design problems where the paradigm of Functional Abstraction has been applied. The background of the respective domains, the related constraints, and an in-depth description of the solutions are given in [16]. Here we content ourselves with a short introduction to the ideas.

3.1 Case-Based Design in Fluidics

Fluidic drives are used to realize a variety of production and manipulation tasks. Even for an experienced engineer, the design of a fluidic system is a complex and time-consuming task, that, at the moment, cannot be automated completely. Designing a system means to transform demands, , towards an explicit system description, which is a behavior model of the desired system in most cases.

Taken the view of configuration, the designer of a fluidic system selects, parameterizes, and connects components like pumps, valves, and cylinders such that is fulfilled by the emerging circuit.[3] Solving a fluidic design problem at the component level is pretty hopeless, and model simplification must be applied to reach tractability. Key idea is to perform a configuration process at the level of functions (instead of components), which in turn requires that fluidic functions possess constructional equivalents that can be treated in a building-block-manner. This requirement is fairly good fulfilled in the fluidic domain—the respective building blocks are called "fluidic axes".

The overall design approach outlined here follows the paradigm depicted in Figure 3 and is illustrated in Figure 4:

(1) The original demand specification, , is abstracted towards a functional specification .

(2) At this functional level a structure model M according to the coupling of the fluidic functions in is generated.

(3) M is completed towards a tentative behavior model M by plugging together locally optimized fluidic axes; in [7] this step is realized by a tailored case-based reasoning approach.

(4) The tentative behavior model M is repaired, adapted, and optimized globally. In this connection scaling rules and heuristic repair rules come into operation.

Remarks. A human designer is capable of working at the component level, *implicitly* creating and combining fluidic axes towards an entire system. His ability to automatically derive function from structure—and vice versa: structure *for* function—allows him to construct a fluidic system without the idea of high-level building blocks in the form of fluidic axes.

Operationalization The concepts have been embedded within a design assistant[4] and linked to FLUIDSIM, a drawing and simulation environment for fluidic systems [17]. The design assistant enables a user to formulate his design requirements by specifying both a set of fluidic functions and a coupling hierarchy. For each fluidic functions a sequence of phases can be defined, where for each phase a set of characteristic parameters, such as duration, precision, or maximum values can be stated.

Clearly, a direct evaluation of generated design solutions must be limited within several respects since

(1) an absolute measure that captures the quality of a design does not exist, and

(2) the number of properties that characterizes a design is large and their quantification often requires a high effort.

[3] The ideas presented in section have been verified in the hydraulic domain in first place; however, they can be applied in the pneumatic domain in a similar way, suggesting us to use preferably the more generic word "fluidic".

[4] The design assistant has been realized and evaluated as a part of the doctoral thesis of Hoffmann [7].

# Axes	Reuse, repair	Correct (at $\sigma > 0.$)	Quality		
1	0.10s	80%	60% (+)	35% (o)	5% (–)
2	0.63s	75%	50% (+)	45% (o)	5% (–)
3	0.91s	70%	40% (+)	50% (o)	10% (–)
4	1.43s	60%	20% (+)	65% (o)	15% (–)
5	2.00s	20%	5% (+)	80% (o)	15% (–)

Table 1: Runtime results and modification effort for automatically generated designs. Test environment was a Pentium II system at 450 MHz with 128 MB main memory.

However, the quality of a generated design can also be rated *indirectly*, by quantifying its "distance" to a design solution created by a human expert. In this connection, the term "distance" stands for the real modification effort that is necessary to transform the computer solution into the human solution. The experimental results presented in Table 1 describe such a competition; a more detailed discussion of evaluation concepts as well as related problems can be found in [7]. Description of the table columns:

- *Axes number.* Number of axes of each test set; a test set contains 20 queries.
- *Reuse, Repair.* Average time of the reuse and repair effort in seconds.
- *Correct.* Number of generated designs with a similarity ≥ 0.9.
- *Quality.* Evaluation of a human expert. In this connection a (+), an (o), and a (–) designate a small, an acceptable, and a large modification effort necessary to transform the machine solution into a solution accepted by the human expert.

3.2 Design in Chemical Engineering[5]

A chemical plant can be viewed as a graph where the nodes represent the devices, or unit-operations, while the edges correspond to the pipes responsible for the material flow. Typical unit-operations are mixing (homogenization, emulsification, suspension, aeration, etc.), heat transfer, and flow transport. The task of designing a chemical plant is defined by the given demands in the form of properties of various input substances, along with the desired output substance. The goal is to mix or to transform the input substances in such a way that the resulting output substance meets the imposed requirements.

The design happens by passing through (and possibly repeating) the following five steps: Preliminary examination of the demands, choice of unit-operations, structure definition, component configuration, and optimization. An automation of the steps at a behavioral level would be very expensive—if possible at all. Present systems limit design support to isolated subjobs; such systems relieve the human designer from special simulation or configuration tasks, and the effort involved there is high enough [1, 8].

However, the Functional Abstraction paradigm can be applied to develop an overall design approach (cf. Figure 5):

(1) The properties of the input and output substances, , are abstracted towards linguistic variables, .

(2) At the functional level a structure model M is synthesized that fulfills and that is used as a solution candidate for ; this step is realized by so-called design graph grammars [14].

(3) M is completed towards a behavior model M.

(4) M is repaired, adapted, or improved.

Both Step 2 and Step 4 rely on a knowledge base with domain-specifc and task-specific design rules. These rules have been formulated as graph grammar rules by a domain expert.

Operationalization The concepts described above are implemented within a prototypical tool, called DIMOD (domain independent modeler). Figure 6 shows a screen snapshot. The core of the system consists of a generic graph grammar engine, used for modeling and application of knowledge, and a domain-specific module used to guide the search process.

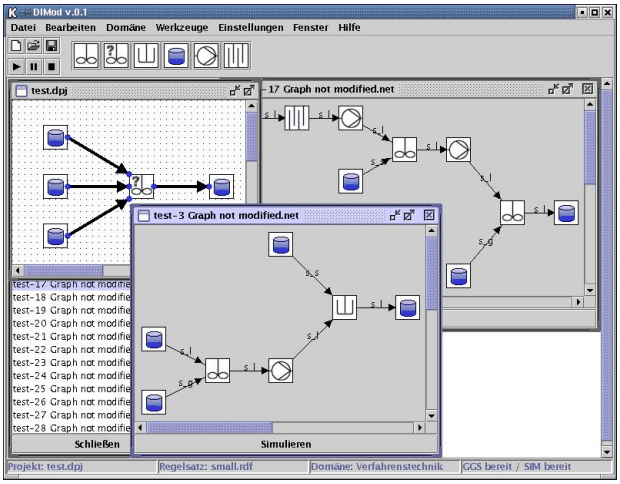

Figure 6: The DIMOD system. Upper left window represents the abstracted demands, the windows to the right and center show two generated structure models.

Within DIMOD the simulation is realized as follows. A generated structure model is completed towards a tentative behavior model by attaching behavior model descriptions to the components of the structure model (Step 3 in Figure 5). The behavior model is then validated by a simulation. For this purpose, the ASCEND IV simulator is used [13]; the attached model descriptions stem from the ASCEND IV model library and from custom models.

[5]This work developed from a cooperative project with the Chemical Engineering Group at Paderborn University that focused on the design of plants for the food processing industry [9].

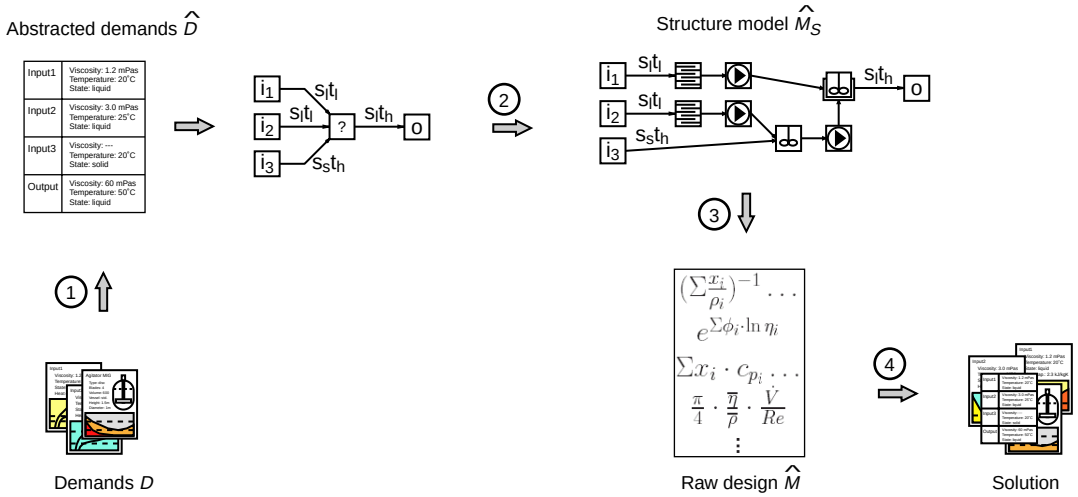

Figure 5: The Functional Abstraction paradigm applied to the design of food processing plants.

References

[1] Axel Brinkop, Norbert Laudwein, and Rüdiger Maassen. Routine Design for Mechanical Engineering. In *Proceedings of the Sixth Annual Conference on Innovative Applications of AI (IAAI 94)*, Seattle, August 1994.

[2] François E. Cellier. *Continuous System Simulation*. Springer, Berlin Heidelberg New York, 1991.

[3] William J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[4] Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, and Gerry Stahl. Embedding Critics in Design Environments. *The Knowledge Engineering Review*, 8(4):285–307, December 1993.

[5] Brian Gaines. General Systems Research: Quo Vadis. In *General Systems Yearbook*, volume 24, pages 1–9, 1994.

[6] Sture Hägglund. Introducing Expert Critiquing Systems. *The Knowledge Engineering Review*, 8(4):281–284, December 1993.

[7] Marcus Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1999.

[8] Achim Knoch and Michael Bottlinger. Expertensysteme in der Verfahrenstechnik – Konfiguration von Rührapparaten. *Chem.-Ing.-Tech.*, 65(7):802–809, 1993.

[9] Annett Kurzok, Manfred H. Pahl, and André Schulz. Software zur wissensbasierten Prozeßmodellierung – WIP. *Chemie Ingenieur Technik*, 73(9), 2001.

[10] Mary Lou Maher and Andres Gomez de Silva Garza. The Adaptation of Structural System Designs Using Genetic Algorithms. In *Proceedings of the International Conference on Information Technology in Civil and Structural Engineering Design: Taking Stock and Future Directions*, Glasgow, Scotland, August 1996.

[11] Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.

[12] Judea Pearl. *Heuristics*. Addison-Wesley, Massachusetts, 1984.

[13] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language. *Computers Chemical Engineering*, 15(1):53–72, 1991.

[14] André Schulz and Benno Stein. On Automated Design of Technical Systems. Notes in Computer Science tr-ri-00-218, Department of Mathematics and Computer Science, University of Paderborn, Germany, December 2000.

[15] Benno Stein. *Functional Models in Configuration Systems*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, June 1995.

[16] Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation thesis, University of Paderborn, Department of Mathematics and Computer Science, 2001.

[17] Benno Stein, Daniel Curatolo, and Marcus Hoffmann. Simulation in FLUIDSIM. In Helena Szczerbicka, editor, *Workshop on Simulation in Knowledge-Based Systems (SIWIS 98)*, number 61 in ASIM Notes, Bremen, Germany, April 1998. Technical committee 4.5 ASIM of the GI.