In: Beiträge zum 13. Workshop *Planen und Konfigurieren (PuK-99)*

Extracting Goal Orderings to Improve Partial-Order Planning

Jochem Hüllen and Frank Weberskirch
University of Kaiserslautern, Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {huellen,weberski}@informatik.uni-kl.de

Abstract

The common wisdom that *goal orderings* can be used to improve planning performance is nearly as old as planning itself. During the last decades of research several approaches emerged that computed goal orderings for different planning paradigms, mostly in the area of state-space planning. For partial-order, plan-space planners goal orderings have not been investigated in much detail. Mechanisms developed for state-space planning are not directly applicable because partial-order planners do not have a current (world) state. Further, it is not completely clear how plan-space planners should make use of goal orderings. This paper describes an approach to extract goal orderings to be used by the plan-space planner CAPLAN. The extraction of goal orderings is based on the analysis of an extended version of *operator graphs* which previously have been found useful for the analysis of interactions and recursion of plan-space planners.

1 Introduction

Research in classical AI planning produced a number of approaches with significant conceptual differences in terms of search space and representation of actions and plans. Because of the fact that planning in general is intractable, research always tried to improve algorithms and representations in order to be able to solve larger problems. The common wisdom says that knowledge about the order in which planning goals should be processed by a planner and/or achieved in a plan can greatly affect the performance (e.g., [DC89, CI89]). It led to so-called *goal orderings* which are given to a planner in order to improve performance. This idea remained during all of these different planning algorithms and approaches.

For some of the planning approaches the question of how such goal orderings can be computed automatically has been subject of research and can be found in literature. On one hand, *learning approaches* have been studied [Min88, RI92], that need different amounts of a priori knowledge about the domain to learn goal orderings, for example, from failures. On the other hand, *preprocessing* of a domain and/or a problem specification in order to get information about the orders of planning goals has been proposed. For state-space planners the most comprehensive investigations about the use of goal orderings have been made [Min88, RI92, Etz93].

How these goal orderings can be interpreted by a planner depends on the type of planner. The interpretation of goal orderings for state-space planners typically is that, if a goal g_1 is ordered before a goal g_2 this means that the planner will work on g_1 before g_2, thus creating a subplan achieving g_1 before creating one achieving g_2. A wrong order for goal selection in worst case can cause backtracking over goal selection or non-minimal solution plans.[1] Likewise, for the Graphplan algorithm [BF97] there is an approach to compute so-called goal agendas [Koe98] which are up to a certain degree comparable with goal orderings of state-space planners. A goal agenda partitions the goals in different, increasing sets of goals which are ordered with respect to each other and processed by the planner in that order.

For partial-order, plan-space planners like SNLP [MR91], UCPOP [PW92] and, descendants goal selection is not a backtracking point but it is an important aspect for a control strategy. In that context, Barrett and Weld described a study focusing on how the goal structure of a problem affects the efficiency of a planner [BW94]. Their distinction between trivially serializable and laboriously serializable subgoal collections characterizes problems for which the exponential costs of backtracking on a small but significant number of subgoal orderings dominate the average planning time. However, research did not investigate in detail whether for plan-space planners something like goal orderings is also appropriate. A direct application of the methods for state-space planning (e.g., STATIC [Etz93]) does not work because of the fact that plan-space planners do not have a current world state.

This paper describes a new approach to precompute goal orderings to be used by partial-order planners with a domain representation that is based on the STRIPS formalism. All mechanisms are fully implemented and part of the planner CAPLAN [Web95]. We will clarify how goal orderings can be used by a partial-order, plan-space planner and what are the assumptions. Our approach to extract goal orderings uses an extended version of *operator graphs* [SP93] as the basic structure on which analysis is performed. It does not need any additional knowledge about the domain for that.

The rest of the paper is organized as follows: In Section 2 we summarize important basic concepts, in particular, the structure called operator graph and its extension. Section 3 presents mechanisms to extract goal orderings from operator graphs for the plan-space planner CAPLAN. Section 4 compares and discusses the approach with respect to related approaches and makes some concluding remarks.

2 Preliminaries

For the purposes of this paper we consider a SNLP style planner and partially ordered plans [MR91, BW94]. Our intention is to present an approach for the computation of goal orderings to be used by such a planner. So, we first have to make clear how a partial-order planner can take profit from goal orderings. Then we explain the basic structure that is used to extract such goal orderings.

Goal orderings for partial-order planning. Problem specifications (I, G) traditionally consist of an initial situation I and the planning goals G. The planner CAPLAN [Web95] allows to make use of additional knowledge about a planning problem expressed as goal orderings. The input consists of so-called *extended problem specifications*, $(I, G, <_G)$, which allow to specify a partial order $<_G$ on the given planning goals. The goal ordering $<_G$ can be interpreted and used by the planner in two ways: (1) if two goals are ordered by $<_G$ then the plan steps achieving those goals in the plan are ordered accordingly, (2)

[1] State-space planning should not be mixed up with total-order planning although most state-space planners use a totally ordered plan representation. For example, the planner TO [MBD94] does not have to backtrack over goal selection but it is a total-order planner.

motivated by the idea of serializability [Kor87, BW94] goals are processed by the planner in an order that is consistent with the goal orderings.[2] Both interpretations can be found in the planning literature. For example, [CI89] gives a definition of goal orderings that is comparable with (1), while others define goal orderings as the problem of deciding on which goal to work next [DC89]. However, until now both mechanisms have not been explicitly distinguished as two completely different ideas. This is mainly a result of the underlying planning paradigm for which such investigations have been made.

In approaches based on state-space planners [Min88, RI92, Etz93] a certain goal ordering typically is used for both purposes at the same time as goal selection also defines in which order operators are added to the plan. The same is true for a new preprocessing approach which computes so-called goal agendas [Koe98] that are used in the context of the Graph-plan algorithm to decide in which order the algorithm should achieve certain goal sets. Here we also have the assumption that goals that are processed first are achieved first in the overall solution plan. Otherwise, the planner does not find minimal plans.
For plan-space planners we have that there is not necessary a relation between the order in which goals are processed and the order in which they are achieved in the plan.[3] In particular, if we have a mechanism that computes goal orderings based on an analysis of operators and their relation in possible solution plans, these orders can be used as specified by interpretation (1) to order plan steps, but they cannot be guaranteed to be useful for the control problem of goal selection (see [WMA98]). With respect to the two interpretations of goal orderings mentioned above, we cannot expect that a goal ordering that is optimal with respect to interpretation (1) is also optimal or at least good for goal selection.

In the context of this paper we always compute goal orderings that are primary intended to be used for ordering the establishers of goals rather than for defining a control strategy. Whether the computed orderings are also useful for goal selection depends on the domain. In addition, we only consider so-called *necessary goal orderings*, which have the property that they are true in all minimal[4] solutions that exist for a certain problem.

Extending operator graphs. Our analysis makes use of *operator graphs* [SP93] which are computed for a problem (I, G) based on the action definitions of the domain. An operator graph captures information about relevant operators for goals and preconditions of operators. It can be understood as a kind of generalized view on the solution space of a problem. This structure was first used to analyze possible interactions (threats) that occur during planning [SP93] and later to recognize recursion for causal-link planners [SP96].

More precisely, an operator graph is a directed bipartite graph consisting of two kinds of nodes: it contains an *operator node* for each operator relevant to the problem (and the dummy operators from the null plan); for each precondition of each operator node there is a *precondition node*. Operator graphs have an implicit AND/OR structure. An operator node is connected to all its precondition nodes (AND connection) and the precondition nodes are called predecessors of that operator node. A precondition node is connected to all operator nodes representing an operator with an effect that unifies with the precondition (OR connection), i.e., operator nodes representing applicable operators are predecessors of precondition nodes. In general, operator graphs can contain cycles because all ground

[2] Interpretation (1) and (2) are also discussed in more detail in [WMA98].

[3] The same is true for the total-order planner TO [MBD94] as goal selection is no backtracking point and TO is able to insert a step at an arbitrary place in its totally ordered plan.

[4] A solution plan is minimal if no subplan is also a solution plan. SNLP finds minimal solutions if simple establishment is preferred to step addition (see also [Kam95]).

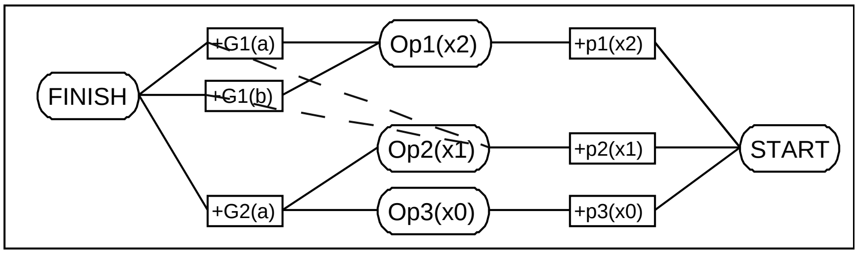

Figure 1: Operator graph for a problem of an artificial domain

instances of a certain operator are represented by exactly one operator node. Figure 1 shows an example of an operator graph. The operator node for FINISH is displayed on the left side, the START node on the right. Predecessors of a node are on the right side of the node and connected with a solid line. In the example, FINISH has three preconditions (the planning goals), *G1(a), G1(b), G2(a)* and *Op2(x1)* and *Op3(x0)* are operators applicable to *G2(a)*. *Op2(x1)* has a precondition *p2(x1)*. We will write *p2(x1)@Op2(x1)* to indicate the precondition node *p2(x1)* that is a direct predecessor of *Op2(x1)*.
Operator graphs also contain information about possible interactions [SP93] that might occur during planning (indicated by dashed lines). Figure 1 shows interactions between *Op2(x1)* and the precondition nodes *G1(a)* and *G1(b)* because *Op2(x1)* deletes *G1(x1)*.

There are two extensions of the original operator graphs that are implemented in CAPLAN and that are important for the process of extracting goal orderings. Both are connected with the fact that CAPLAN does not use ground instances of operators but operators with variables: (1) CAPLAN allows to use different kinds of constraints in its operator specifications, codesignation constraints and type constraints [Web95]. Types can be organized in hierarchies with single or multiple inheritance. The expansion of operator graphs, in particular, the test for applicability of an operator has to take into account that such type constraints should also be considered. This results in operator graphs with less nodes and/or edges. (2) Operator graphs are very compact since all ground instances of an operator are represented by one node. The disadvantage (in comparison with representing all ground instances separately) is that there is no information about possible bindings of variables. To overcome this, constant propagation in the operator graph precomputes additional knowledge in the operator nodes (for more details see [Hül98]).
An operator node for the operator O contains for each goal g of the problem a list with the constants of the problem which can be bound to the individual variables of O when O is used to achieve g. Constant propagation enables a further advantage: an empty set of possible bindings for at least one variable of an operator O w.r.t. a goal g denotes that O can't achieve (directly or indirectly) g in this problem. In this situation we say O is *blocked* for g. This information can make the analysis more efficient and also more powerful.

Figure 1 shows the importance of the extensions of the operator graphs for analyzing goal orderings. The definition of the domain (the operators) of this example and the goals can be derived from the operator graph. The initial state is {*+p1(a),+p1(b),+p2(a),+p3(b)*}. Since there is no *p3(a)* the operator *Op3(x0)* is blocked for the goal *G2(a)* (this is detected during constant propagation since there is no possible binding for *x0* in *Op3* for solving *G2(a)*. So, the only possibility for solving *G2(a)* is *Op2(x1)*. Further, there exist two potential threats to *G1(a)* and to *G1(b)* and there are no structural differences with respect to these two goals. But with the knowledge of variable bindings the important difference can be discovered. Operator node *Op2(x1)* contains the possible bindings for *x1* when

Op2(x1) is used to achieve *G2(a)*. The only possibility is $(x1 \rightarrow a)$ whereas the threat to *G1(b)* needs the binding $(x1 \rightarrow b)$. So the necessary ordering *G2(a)* < *G1(a)* can be detected with the goal clobbering criterion (Section 3) whereas *G1(b)* and *G2(a)* remain unordered. Without the information of *blocked* operators no ordering would be detected and without consideration of variable bindings the ordering *G2(a)* < *G1(b)* (which is no necessary ordering) would be detected. The consideration of type constrains can have analogous consequences: suppose, there is *+p3(a)* in the initial state but *Op3(x0)* has a type constraint which prohibits the binding $(x0 \rightarrow a)$.

To take advantage of the information of possible bindings we use the relation *imply* for variable bindings. Variable bindings v imply variable bindings w if all bindings in w also occurs in v, i.e., $\{(x \rightarrow a), (y \rightarrow b), (z \rightarrow c)\}$ *implies* $\{(x \rightarrow a), (y \rightarrow b)\}$ but not vice versa.

Operator graphs with cycles. We call domains which have operator graphs with cycles *recursive domains*. Whereas the effort for constant propagation in non recursive domains is not critical (the time for constant propagation is less than one percent of the total solving time in all tested non recursive domains) it could get problematic in recursive domains. Therefore, we formulate an efficient propagation algorithm for recursive domains which approximates the possible bindings. The criteria in Section 3 work correctly with this approximation if the relation *imply* is adapted (for more details see [Hül98]). However it can happen that with this approximation less orderings are found.

3 Extracting Goal Orderings From Operator Graphs

In our approach the structure of an extended operator graph for the considered problem is analyzed to extract goal orderings without using any additional knowledge about the problem or domain. In general, different reasons for goal orderings exist in domains. We formulate three criteria (*goal subsumption, goal clobbering, precondition violation*) which cover these different reasons. For a better understanding we present these criteria separately although in the implementation they are tested simultaneously since the combination forms a more powerful composite criterion.

All criteria compare two goals g_1, g_2 and decide if $g_1 <_G g_2$ holds, i.e., g_1 is always achieved in solution before g_2 is achieved. To analyze the whole problem the composite criterion has to be tested for all ordered pairs of goals. One can use transitivity and irreflexivity (if no inconsistent problems are considered) of $<_G$ to speed up this procedure. For all three criteria we give a colloquial description and explain how an efficient implementation using operator graphs looks like. All criteria are correct but not necessarily complete. The formal definitions and proofs can be found in [Hül98].

Goal subsumption. For a problem (I, G) the goal ordering $g_1 <_G g_2$ holds because of goal subsumption if every solution plan achieving g_2 also achieves g_1 previously and there is no goal $g \in G \backslash \{g_1\}$ whose solution could negate[5] g_1.

In the artificial domain *LinkRepeat* [VB94] we can find goal subsumption. Figure 2 shows the operator graph for a problem where *G1* $<_G$ *G2* holds because of goal subsumption. This ordering can be detected by analyzing the operator graph as follows: We have to examine if every possibility for solving *G2* also solves *G1*. Therefore we look at all operators which

[5]We say *negate* instead of *delete* since CAPLAN and our operator graph implementation allow negative as well as positive goals and preconditions.

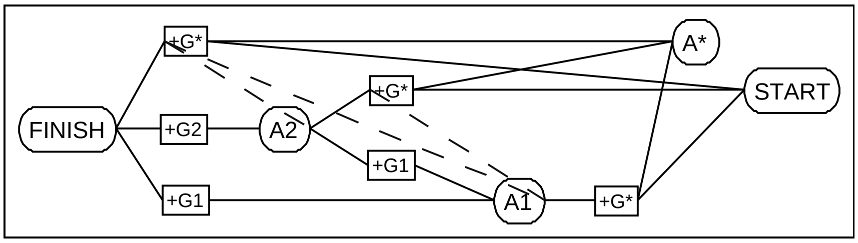

Figure 2: Operator graph for a problem of the *LinkRepeat* domain

can solve *G2*. This are exactly the predecessors of *G2* (here *A2*). *A2* itself doesn't solve *G1* (*A2* is not a predecessor of *G1*) but we have to check the preconditions of *A2* (here *G*@A2* and *G1@A2*). For both we have to make the same test as for *G2*. *G*@A2* could be solved by *A** or by *START* which both do not solve *G1*; *G1@A2* can only be solved by *A1* which also solves *G1@FINISH*. So, *G1* has to be solved before *G2* and, furthermore, there is no threat to *G1*, i.e., there exists no goal whose solution could negate *G1*.

The goal subsumption criterion can be formulated recursively by testing if the predecessors of g_2 solve g_1. To generalize the example the following aspects have to be considered:

- The implicit AND/OR structure of operator graphs has to be considered: *All* predecessors of a precondition node which are not blocked for g_2 (i.e., the applicable operators) must lead to an establisher of g_1 (since one arbitrary not blocked operator could be chosen) but it is sufficient if *one* predecessor of an operator node always leads to an establisher of g_1 (since all preconditions have to be solved to execute the operator).
- If an operator in the first level (here *A2*) solves g_1 then $g_1 <_G g_2$ doesn't hold since then the goals could be achieved simultaneously.
- If operators have variables then the possible variable bindings have to be considered. The bindings for solving a subgoal of g_2 have to imply the bindings for solving g_1. A detailed explanation of this problem can be found in [Hül98].
- We must test if there are threats to g_1 from operators which could be executed to solve other goals. So, in the problem from above $G^* <_G G2$ doesn't hold although G^* is a precondition of *A2* since there exist threats from *A2* and *A1* to G^*.

Now we define a recursive predicate *subsumption* for a precondition node p and two goals g_1, g_2 of an extended operator graph. This predicate is true if solving p also guarantees that g_1 is solved. The condition $p \neq g_2$ ensures that g_1 is achieved before g_2 (not simultaneously). The formulation of *subsumption* shows that this predicate can be tested very efficient using operator graphs.

Definition 1 *Given a problem with goals* g_1, g_2 *and the corresponding operator graph* OG *then* $subsumption_{g_2}(p, g_1)$ *holds for a precondition node* $p \in OG$ *if and only if for all predecessors* O *of* p *the following statement holds:*

$$
\begin{array}{lll}
 & \text{O } \textit{is blocked for } g_2 & \vee \\
(& \text{O } \textit{is predecessor of } g_1 \quad \wedge \quad p \neq g_2 \quad \wedge & \\
 & \textit{the variable bindings of } \text{p } \textit{in } \text{O } \textit{imply the variable bindings of } g_1 \quad) & \vee \\
 & \textit{there exists a predecessor } \text{q } \textit{of } \text{O } \textit{such that } subsumption_{g_2}(q, g_1) \textit{ holds} &
\end{array}
$$

Theorem 1 *For a problem (I, G) with $g_1, g_2 \in G$ we have that $g_1 <_G g_2$ is a necessary goal ordering if $subsumption_{g_2}(g_2, g_1)$ holds in the corresponding operator graph and there is no threat to g_1 from an operator which could be executed with the right variable bindings to solve another goal.*

This is a formal description of the idea of goal subsumption which can be proved to be correct [Hül98]. If the property *subsumption* is stored locally for each node in the OG this criterion can be tested with time cost $O(n)$ (for n being the number of nodes in OG).

Goal clobbering. For a problem (I, G) the goal ordering $g_1 <_G g_2$ holds because of goal clobbering if every solution plan for g_1 negates g_2. So it is of no use to achieve g_2 first since after solving g_1 it has to be solved again.

In *LinkRepeat* also goal clobbering occurs. $G1 <_G G^*$ and $G2 <_G G^*$ hold because of goal clobbering (see Figure 2). $G1 <_G G^*$ can be found by analyzing the operator graph as follows ($G2 <_G G^*$ works similar): All possibilities for solving *G1* have to be examined. The only predecessor of *G1* is *A1* and there exist a threat from *A1* to *G**.

In this example the test is very simple but it gets much more complicated if indirect predecessors of g_1 negate g_2 (the implemented composite criterion covers such cases). The AND/OR structure has to be considered as in the subsumption criterion. If variables occur the bindings for solving g_1 have to imply the bindings of the threat to g_2.

Precondition violation. For a problem (I, G) the ordering $g_1 <_G g_2$ holds because of precondition violation if every solution plan for g_2 results in a state in which g_1 can't be solved afterwards.

Figure 3 shows an operator graph for a problem of a variant of the D^1S^2 domain [BW94].[6] For this problem $G1 <_G G2$ holds because of precondition violation and is detected by this criterion as follows:[7] Again the possibilities for solving *G2* have to be examined. The only candidate is *A22* which threatens *P1*. So, after achieving *G2* with *A22* goal *G1* can't be solved only using *A12* because of the missing precondition *P1*. But *P1* could be achieved with *A11* since *I1* wasn't threatened by *A22*. The preconditions of *A22* have to be examined in the same way as for *G2*. The only operator which can solve *P2* is *A21* which threatens *P1* and *I1*. Consequently, there is no possibility to solve *G1* after *G2*.

The precondition violation criterion can be formulated analogously to the subsumption criterion with the same properties regarding AND/OR structure and variable bindings.

Composite criterion. The three criteria are implemented together in one composite criterion. When they are tested simultaneously they can complement each other to find more goal orderings. A very important aspect is that using operator graphs results in an efficient implementation since they represent all potential solutions in an abstract way such that questions like *'what operators can solve or threaten a certain predicate'* can be answered immediately.

---

[6] The modification consists of the following variants of the operators in D^1S^2 (notation: $\{preconditions\}$ *operator* $\{effects\}$): $\{I_i\}$ A_{i1} $\{+P_i, -P_{i-1}, -I_{i-1}\}$ and $\{P_i\}$ A_{i2} $\{+G_i, -P_{i-1}\}$. In D^1S^2 the action A_{i2} has the additional effects $-I_{j|\forall j}$ instead of the effect $-P_{i-1}$ in A_{i1}.

[7] This ordering also holds in the original version of D^1S^2 (although there are different solution plans) and is also detected by the precondition violation criterion but the variant we use here points out more distinctly the properties and difficulties of this criterion.

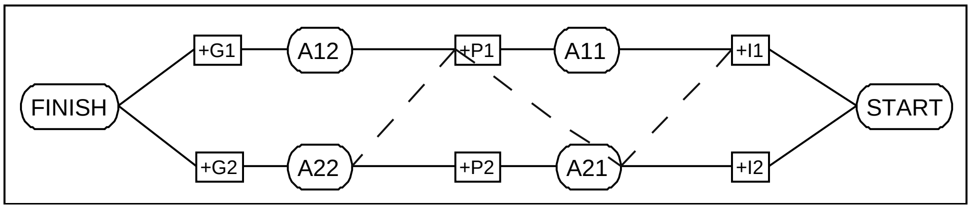

Figure 3: Operator graph for a problem from a D^1S^2 variant

Analysis in recursive domains. Recursive domains are in general difficult to analyze. Nevertheless, the criteria presented above are also correct for recursive domains. And since each node has to be examined at most once there are no problems with termination or efficiency. But the results for many recursive domains are very poor.[8] This is a consequence of using operators with variables instead of ground instances of operators. The approach with possible variable bindings is not meaningful enough in this situation.

Empirical Results

The general efficiency improvement of planning with goal orderings in CAPLAN has been investigated in [WMA98]. Here we summarize which orderings can be found with the presented criteria and how costly is the preprocessing. We tested the analysis in the domains D^mS^n, $\theta_kD^mS^n$ [BW94], *LinkRepeat* [VB94], *process planning* [MAW96] and *Blocksworld*.

Effort of preprocessing: Since even for complex problems the operator graphs are relatively compact the time for preprocessing is negligible. The only critical part is the constant propagation in recursive domains. Using the approximation (see Section 2) the whole analysis (construction of the operator graph, constant propagation and testing the presented criteria) needs less than 3% of the overall problem solving time in all tested domains.

Results of preprocessing: In the problems of the artificial domains D^mS^n, $\theta_kD^mS^n$ and *LinkRepeat* all existing necessary orderings were found. In the *process planning* a subset of the necessary orderings were found (depending on the problems). In *Blocksworld* no orderings were found. This is a result of the fact that Blocksworld is highly recursive and operator graphs for nearly all problems have the same structure (only different variable bindings).

4 Related Work and Conclusion

As said in the introduction there are a number of approaches dealing with the computation of goal orderings in planning. Early approaches like [CI89] or [DC89] pointed out the different interpretations of goal orderings without noticing in detail that two completely different ideas are behind these interpretations.

Our operator graph analysis is comparable with some other analytic approaches. STATIC [Etz93] is an example that computes goal orderings to be used by the state-space planner PRODIGY. It evaluates so-called problem space graphs (PSGs) which are similar to operator graphs, but which are computed for each predicate of the domain instead of a

[8] Blocksworld is an example. It consists of only a few operator (most of them in cycles) and the difficulty is to dispose them in various instantiations.

problem-specific computation. Goal orderings are obtained by the analysis of the PSGs for the goals that are compared. A major problem is that PSGs need a priori knowledge about the domain (a specific form of domain axioms) which is easy to define for Blocksworld but not for larger and more realistic domains. A lack of such axioms makes PSGs and their analysis become very complicated and inefficient. Further, PSGs do not focus on a certain problem but on the hole domain. This makes it difficult to find useful problem-specific orderings.

Operator graphs are also in some way comparable with the planning graphs of Graphplan [BF97]. The major difference is that planning graphs always use ground instances of operators and are built in a forward-chaining manner. Instead of threats in the operator graph, Graphplan propagates a certain mutual exclusion relation among the nodes. The idea of greedy regression-match graphs [McD96], which also have similarities with operator graphs, has been applied to extract information about relevant operators and facts in order to reduce the size of planning graphs [NDK97]. In [Koe98] so-called goal agendas are computed from a relation on the goal atoms which is obtained from the planning graph or computed from the operator definitions. This atomic relation is not a necessary or possible ordering (Section 2) as it might contain relations like $A \prec B$ and $B \prec A$ at the same time. But from it an order over increasing subsets of goal atoms is derived which, in best case, can dramatically speed up Graphplan on some large problems. It seems that this approach is very good in highly recursive domains like Blocksworld or Tyreworld where our approach fails to find goal orderings. This can be explained by the fact that it only uses ground instances of operators. An important difference is, however, that for a computed goal agenda it cannot guaranteed that plans with minimal length[9] are still found while our approach computes necessary goal orderings which hold in every solution plan.

A limitation of our approach is that operator graphs depend on nearly the same simple domain representation language as the original Graphplan. In particular, it cannot deal with conditional effects or quantification which are commonly seen as important for being able to represent domains easily and naturally. Unlike most other planners CAPLAN and our operator graph analysis allows additional type hierarchies and type constraints. On one hand, one could think about extending the algorithms for operator graphs in a similar way as planning algorithms have been extended to deal with ADL operators in UCPOP and Graphplan descendants. On the other hand, there is also work in the area of automatically translating domain specifications with language elements like conditional effects or quantification into simpler domain representation languages [GK97] which could also be applied to domains to be processed with these operator graphs and CAPLAN.

Future work: We think about extending the operator graph such that in recursive domains some of the recursive operator nodes are unfolded into a certain number of partially instantiated operator nodes. This might produce better results in recursive domains.

References

[BF97] A. Blum and M. Furst. Fast planning through plan-graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[BW94] A. Barrett and D.S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[9]Notice, minimality of plans if defined differently for Graphplan and SNLP.

[CI89] Jie Cheng and K.B. Irani. Ordering problem subgoals. In *Proceedings of IJCAI-89*, pages 931–936, 1989.

[DC89] M. Drummond and K. Currie. Goal ordering in partially ordered plans. In *Proceedings of IJCAI-89*, pages 960–965, 1989.

[Etz93] O. Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62:255–301, 1993.

[GK97] B.C. Gazen and C.A. Knoblock. Combining the expressivity of ucpop with the efficiency of graphplan. In *Proc. of the 4rd European Conf. on Planning (ECP-97)*, pages 221–233, 1997.

[Hül98] J. Hüllen. Improving the efficiency of the partial-order planner CAPlan using operator graphs. Masters thesis (in German), University of Kaiserslautern, 1998.

[Kam95] S. Kambhampati. Admissible pruning strategies based on plan minimality for plan-space planning. In *Proceedings of IJCAI-95*, 1995.

[Koe98] J. Koehler. Solving complex planning tasks through extraction of subproblems. In *Proc. of the 4rd Intern. Conf. on AI Planning Systems (AIPS-98)*, pages 62–69, 1998.

[Kor87] R.E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[MAW96] H. Muñoz-Avila and F. Weberskirch. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In *Proc. of the 3nd Intern. Conf. on AI Planning Systems (AIPS-96)*, 1996.

[MBD94] S. Minton, J. Bresina, and M. Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[McD96] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. of the 3nd Intern. Conf. on AI Planning Systems (AIPS-96)*, pages 142–149, 1996.

[Min88] S. Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, 1988.

[MR91] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639, 1991.

[NDK97] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *Proc. of the 4rd European Conf. on Planning (ECP-97)*, pages 338–350, 1997.

[PW92] J.S. Penberthy and D.S. Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of KR-93*, 1992.

[RI92] K.R. Ryu and K.B. Irani. Learning from goal interactions in planning: Goal stack analysis and generalization. In *Proceedings of AAAI-92*, pages 401–407, 1992.

[SP93] D. Smith and M. Peot. Postponing threats in partial-order planning. In *Proceedings of AAAI-93*, pages 500–506, 1993.

[SP96] D. Smith and M. Peot. Suspending recursion in causal-link planning. In *Proc. of the 3nd Intern. Conf. on AI Planning Systems (AIPS-96)*, pages 182–190, 1996.

[VB94] M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proc. of the 2nd Intern. Conf. on AI Planning Systems (AIPS-94)*, pages 13–19, 1994.

[Web95] F. Weberskirch. Combining SNLP-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany, 1995.

[WMA98] F. Weberskirch and H. Muñoz-Avila. Extending problem specifications for plan-space planners. Technical Report LSA-98-02E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany, 1998.