Computing higher order exclusion relations in propositional planning

Ioannis Refanidis* and Ilias Sakellariou

University of Macedonia, Dept. of Applied Informatics, Thessaloniki, Greece

This article presents a systematic and complete algorithm to compute *all* higher order exclusion relations for a propositional planning problem, that is, sets of propositions that cannot hold simultaneously at specific time points, without any bound on the order of the exclusion relations. This algorithm is proved to allow for backtrack-free plan extraction, provided that all goals have to be achieved simultaneously. In particular, *leveled global consistency* is achieved, i.e., all exclusion relations between propositions within each time step are computed. However, achieving leveled global consistency is impractical for most non-trivial planning problems. Indeed, as our empirical evaluation over a variety of planning problems suggest, best performance is achieved when setting a bound on the order of the computed exclusion relations and using search to extract a plan. Additional statistics extracted from our experiments shed light on the internal dynamics of Graphplan-style planners.

Keywords: propositional planning; exclusion relations; Graphplan; consistency

1. Introduction

A propositional constraint satisfaction problem can be defined as follows: Given a set of propositions $\mathcal{P}$ and a set of propositional formulas (i.e. constraints) C over them, find an assignment to all propositions of $\mathcal{P}$ such that all formulas in C hold. A constraint satisfaction problem is i-consistent if every valid assignment to any subset of i-1 variables can be extended to a valid assignment of i variables, for any i-th variable. It is strongly i-consistent, if it is j-consistent for all $j \leq i$. Finally, it is globally consistent, if it is strongly n-consistent, where n is the number of the variables (Freuder 1978).

A propositional planning problem can be defined as a tuple $\langle \mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{P}$ is a set of propositions, $\mathcal{A}$ is a set of actions, $\mathcal{I} \subseteq \mathcal{P}$ describes the initial state and $\mathcal{G} \subseteq \mathcal{P}$ are the goals. For every action $a \in \mathcal{A}$, the sets $Precs(a) \subseteq \mathcal{P}$, $Dels(a) \subseteq \mathcal{P}$ and $Adds(a) \subseteq \mathcal{P}$ denote preconditions, delete effects and add effects respectively. Without loss of generality we assume that $Dels(a) \subseteq Precs(a)$. In the following, we will use the notation $a = \langle Precs(a), Dels(a), Adds(a) \rangle$ to shortly define an action, e.g. $a = \langle \{p,q\}, \{p\}, \{r,s\} \rangle$.

An action a is applicable to a state s if $Precs(a) \subseteq s$. The state that results from the application of a to s is defined by $\gamma(s,a) = s \cup Adds(a) - Dels(a)$. Action execution is considered instantaneous. A set of actions $A = \{a_1, a_2, \ldots, a_n\}$ can be executed simultaneously if neither of them deletes a precondition or an add effect of the others. The combined outcome from the simultaneous execution of a set of actions is identical to the outcome of any sequential execution of them, i.e., $\gamma(s, A) = \gamma(\gamma(s,a), A - \{a\})$, where $a \in A$. A plan π is defined as a sequence of action sets, $\pi = \langle A_0, A_1, \ldots, A_{N-1} \rangle$,

* Corresponding author. Email: yrefanid@uom.gr

with each A_i consisting of actions that can be executed simultaneously. A plan is considered valid if all actions in every set are applicable in the state resulting from the application of all actions of all previous sets, with the actions of A_0 being applicable in $\mathcal{I}$. A valid plan is considered a solution to the planning problem iff $\mathcal{G} \subseteq s_N$, where $s_N = \gamma(\mathcal{I},\pi) = \gamma(\gamma(\mathcal{I},A_0), \{A_1,A_2,\ldots A_{N-1}\})$. An optimal plan π^* is a solution plan that has the minimum number of steps. So, the computational problem consists of finding the optimal plan π^* for $\langle \mathcal{P},\mathcal{A},\mathcal{I},\mathcal{G} \rangle$.

A planning problem can be formulated as a propositional constraint satisfaction problem (McAllester 1990), with each proposition and each action at any time step being a Boolean variable. In particular, for any proposition p, we can define constraint variables $p_0, p_1, p_2, \ldots, p_N$, denoting whether p holds at time steps 0, 1, 2, …, N, where N denotes the steps of the plan. Similarly, for every action a, we can define constraint variables $a_0, a_1, a_2, \ldots, a_{N-1}$, denoting whether action a is applied at time steps 0, 1, 2, …, N. The values of the variables p_0 for the various propositions p are given by the initial state description. Various constraints encode the underlying physics, i.e. sets of propositions (actions) that cannot hold (be executed) simultaneously either eternally or up to a specific time point, sets of propositions that should be true at a time point in order for an action to be executed at the same time point and, finally, sets of propositions that take specific truth values as a consequence of executing an action.

The constraint satisfaction problem gradually evolves in time, with new sets of actions and propositions being added with each new time step. As long as the goal propositions appear in a new proposition level, without any constraint prohibiting them from being true simultaneously, an attempt to extract a complete solution, i.e., a plan, takes place using search like in Graphplan (Blum and Furst 1997) and SATPLAN (Kautz and Selman 1998). If no global consistency is enforced to the underlying constraint satisfaction problem, having established the goal propositions at a time step without any constraint prohibiting their co-existence is not a sufficient condition for the existence of a solution plan. So, it is common that after failing to extend the partial assignment to a total one, the constraint satisfaction problem is extended by an additional time step, i.e., a new pair of action and proposition levels.

In the past, several attempts were made to compute higher order exclusion relations working directly with a Conjunctive Normal Form (CNF) representation, usually by setting a bound on the order on the computed relations. However, none of them has been proved to be complete, i.e., they could not compute all higher order exclusion relations provided that no bound has been set. In this article we present a complete algorithm to compute all higher order exclusion relations (named *hoex* hereafter) between sets of propositions, sets of actions and sets of propositions and actions at the various time steps, without any bound on the order of the derived relations, thus establishing *leveled global consistency*. An exclusion relation between a set of propositions (actions) at a specific time step denotes that all the propositions (actions) cannot hold (executed) simultaneously at that time step. Having established leveled global consistency is a sufficient condition for backtrack-free solution extraction, provided that all goals have to be achieved simultaneously. This article provides a proof of this fact that is also demonstrated using a declarative implementation (in Prolog) of the proposed algorithm through a series of experiments.

Establishing leveled global consistency is however *not* practical for most realistic planning problem. So, besides its theoretical importance, the practical importance of this work lies in serving as the best starting point for bounded *hoex* computation. And, as it turns out from our empirical evaluation, for several planning problems computing binary exclusion relations is less efficient than computing higher order

ones. This observation can be exploited in other planning frameworks as, e.g., in heuristic planning for better guidance or in planning as satisfiability as additional constraints.

This article extends previous work (Refanidis and Sakellariou 2009) mainly in three dimensions. First, we tackle the problem from a constraint satisfaction perspective introducing also the notion of leveled global consistency. Second, we generalize search-based Graphplan to work with higher order exclusion relations. And, third, we present extensive experiments demonstrating the tradeoff between constraint propagation and search as well several interesting statistics.

The rest of the article is structured as follows: Section 2 gives a quick overview of Graphplan, whereas Section 3 reviews the literature with respect to higher order exclusion relation computation. Section 4 defines higher order exclusion relations and identifies some key properties of them. Section 5, which constitutes the core of the article, presents the details of the higher order exclusion relation computation method, including some implementation considerations, and two solution extraction methods, one backtrack-free and one search based. Section 6 illustrates the algorithm at work and presents some experimental results on simple problems from several domains. Finally, Section 7 concludes the article and discusses potential research challenges.

2. Graphplan Overview

A planning graph is a graph structure consisting of alternating propositional and action levels, with the zeroth propositional level comprising the propositions of $\mathcal{I}$. All propositions and actions are ground. Mutual exclusion relations or *mutexes* over pairs of actions or pairs of propositions of the same level are recursively defined as follows:

1. Two actions a and b are eternally mutually exclusive, denoted by $emutex(\{a,b\})$, if:
   – a deletes a precondition or an add effect of b, or
   – b deletes a precondition or an add effect of a.
2. Two propositions p and q are mutually exclusive at proposition level i, denoted by $mutex_i(\{p,q\})$, if there is no way to achieve them simultaneously at proposition level i, i.e., there is no single action or pair of non-mutexed actions applicable at action level i-1 that achieve p and q.
3. Two non emutexed actions a and b are mutually exclusive at action level i, denoted by $mutex_i(\{a,b\})$, if there is a pair of propositions p and q, such that $p \in Precs(a)$ and $q \in Precs(b)$, and a relation $mutex_i(\{p,q\})$ holds.

So, starting from the zeroth propositional level, an action a appears in action level i if all of its preconditions appear at proposition level i, without any exclusion relation holding between them. Similarly, a proposition p appears at proposition level i, if an action achieving p appears at action level i-1. Note that Graphplan introduces special actions called *noop* actions. In particular, for every proposition p a *noop* action is defined, having p as its single precondition and its single add effect. So, mutex relations between *noop* actions and normal actions are also considered.

As soon as a propositional level i is encountered, with all the propositions of $\mathcal{G}$ in it and without any mutual exclusion relation between any pair of them, a systematic backwards plan extraction phase is employed. The plan consists of sets of non-mutexed actions at the various action levels, such that each set is applicable at the corresponding time step and the application of the entire plan results in a state where the goals hold. If such a plan does not exist, the planning graph is expanded by another level of actions and propositions and the plan extraction phase starts from scratch.

At the time of their introduction, planning graphs were the dominant technology for propositional planning, as demonstrated by the systems that participated in the first international planning competition[2]: three of them, IPP (Koehler et. al. 1997), SGP (Anderson, Smith and Weld 1998) and STAN (Long and Fox 1999) were variations of the basic Graphplan algorithm, BLACKBOX (Kautz and Selman 1998) was using a planning graph to transform the planning problem into an equivalent satisfiability one, and only one, HSP (Bonet, Loerincs and Geffner 1997), was not using planning graphs and exclusion relations. In the following years, planning graphs have been exploited in the framework of domain independent heuristic state-space planning, with the FF planning system (Hoffmann and Nebel 2001) being the most successful case. Planning graphs have also been used to compute heuristics for plan-space planners (Nguyen and Kambhampati 2001), whereas extensions for temporal and conformant domains have been devised (Smith and Weld 1999; Weld and Smith 1998).

3. Related Work

In the literature there are many research efforts towards computing higher order exclusion relations. These efforts took mainly the form of computing invariants, i.e., expressions over a problem's propositions that remain true eternally. However, invariants cover one aspect of the problem, i.e., the eternal exclusion relations, whereas there are conditional exclusion relations as well, i.e., exclusion relations that vanish after some time step.

To the best of our knowledge, a complete algorithm to compute *all* these relations (eternal and conditional) while working directly with the CNF representation has been neither presented nor implemented. The common situation was to identify that incomplete algorithms could be extended to compute all higher order exclusion relations and, consequently, this could result in backtrack-free solution extraction. However, due probably to the expected computational cost, no attempt has ever been made until now. As we present in this article, this extension is not straightforward, whereas there are several opportunities for optimization; nevertheless, no code optimization can transform this problem to a tractable one.

One of the first approaches to generate state invariants was the DISCOPLAN system by Gerevini and Schubert (Gerevini and Schubert 1998). The approach consists of generating hypothetical state constraints by inspection of operator effects and preconditions, and checking each hypothesis against all operators and the initial conditions. State invariants are also computed by Fox and Long (1998) in the TIM system. In this case, they are extracted from the automatically inferred (by the STAN system; Long and Fox 1999) type structure of a domain. TIM takes a domain description in which no type information need be supplied and infers a rich type structure from the functional relationships between objects in the domain. If type information is supplied, TIM can exploit it as the foundation of the type structure and will often infer an enriched type structure on this basis. State invariants can be extracted from the way in which the inferred types are partitioned.

Probably the work on synthesizing invariants by Rintanen (Rintanen 2000) is the one most closely related to the work presented in this article. Rintanen's algorithm is an improvement of the approaches taken in (Fox and Long 1998) and (Gerevini and Schubert 1998). However, Rintanen's algorithm sacrifices completeness with respect

[2] ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html

to the invariants generated in order to maintain polynomial time, since it is targeted towards the implementation of practical planning systems. Sacrificing completeness is not only because there is a bound on the order of the computed invariants, neither due to the fact that the implementations of the functions '*extend*', '*update*', '*preserves*' and '*weaken*' are incomplete, as stated in that paper. It is mainly because Rintanen's algorithm iterates only over actions at each planning graph level. As we demonstrate in the Section 5, this is not sufficient: in order to guarantee completeness, iterating over the sets of consistent actions at each planning graph level is necessary.

A similar work is the one by Haslum and Geffner (2000), in which the authors compute a family of heuristics referred as h^m. These heuristics are based on computing admissible estimates of the costs of achieving sets of atoms of order m from the initial state. For higher values of m, similar results to those of our work could be obtained. However, even if m equals the number of propositions, underestimates might be produced. This is again due to the fact that the proposed recursive computation iterates over actions and not over sets of actions. Furthermore, no implementation for $m>2$ is provided, even for small problems, to provide empirical evidence.

Helmert (2009) presents an efficient algorithm for translating planning tasks specified using Boolean variables, into a compact finite domain representation. The translation is based on identifying sets of propositions which are pairwise mutually exclusive, so they can be easily encoded as a single state variable whose value specifies which of the propositions is true. The proof of invariance, i.e., that exactly one of the propositions is true at any time, is inductive. Bernardini and Smith (2011) extend Helmert's work to temporal and numeric domains.

Recently, the computation of long distance mutexes has been introduced in (Chen et. al. 2009). A long distance mutex, or *londex*, is a binary exclusion relation between actions and/or propositions in different levels of the planning graph. Long distance exclusion relations can be considered complementary to higher order exclusion relations, since both introduce additional constraints besides traditional binary mutexes. Furthermore, both approaches can be exploited to prune the search space in several planning frameworks. Combining the two approaches is worth investigating, in order to derive long distance higher order exclusion relations and thus achieving higher levels of general (not leveled) consistency for the underlying planning problem. General global consistency is expected to allow solving planning problems with temporally annotated goals in a backtracking free manner. Obviously, it is a far more expensive computational problem.

Using an alternative representation, enforcing leveled global consistency could be achieved through exhaustive construction of the space of the states, i.e., working with a Disjunctive Normal Form (DNF) representation, followed by a transformation of the DNF to an equivalent CNF representation (Darwiche and Marquis 2002), thus computing all higher order exclusion relations (note that in order to achieve exactly the same functionality, one needs to consider, while constructing the space of the states, all sets of compatible applicable actions to each state). This approach could take advantage of efficient symbolic forward exploration techniques such as BDD-based planning algorithms (Bryan 1985; Cimatti et. al. 1997; Edelkamp and Jabbar 2006; Fourman 2000]). However, there are deficiencies with this approach. The two spaces, i.e., the space of states and the space of exclusion relations, are dual: the former comprises what *can* be achieved, whereas the latter comprises what *cannot* be achieved. The two spaces are not of equal size for any planning problem. So, there are domains where a DNF representation is more compact, whereas there are other

domains where CNF is the best choice. It is apparently inefficient for problems that admit a more compact CNF representation to construct exhaustively the space of the states. Furthermore, there is no point in computing higher order exclusion relations through transformation from an exhaustive state-space construction, since after having found all the reachable states the planning problem is solved. On the other hand, a DNF to CNF transformation is not an option when one is interested in computing a subset of the *hoex* relations coupled with search to solve the planning problem, since with this transformation all *hoex* relations are always computed. Finally, exclusion relations can naturally be extended to temporal domains, while allowing for optimizing makespan (Smith and Weld 1999; Refanidis 2005); it is not clear how this functionality could be achieved using state space construction followed by a DNF to CNF transformation.

Finally, computing sets of non-achievable propositions at specific planning levels resembles the memoization technique used in the original Graphplan to extract plans from a planning graph (Blum and Furst 1997). However, whenever Graphplan determines that a set of propositions S is not achievable at level i and memoizes this piece of information, there is no guarantee as to whether S is of minimum size or i is the maximum possible level at which S is non-achievable. Furthermore, Graphplan does not discover all no-good sets of propositions but only those encountered during the search phase, thus this approach is not complete. Hence, although there are similarities between our work and the memoization technique of Graphplan, the two approaches are very different.

In this article we adopt a different perspective: Our goal is to provide a systematic and complete method to compute *all* higher order exclusion relations between the propositions and the actions of a planning problem for specific time steps, working directly with the CNF representation. We consider such a result important since (a) it clearly demonstrates the complexity of the task, (b) it formally proves that computing such information is a sufficient condition for generating plans in a backtrack-free manner, and (c) it will possibly rise new research challenges. For example, as, e.g., solving planning problems with exhaustive construction of the space of exclusion relations using efficient data structures; coupling *hoex* computation with search in several planning frameworks; or, finally, experimentally assessing the difficulty of planning domains and problems for domain independent planning.

4. Higher Order Exclusion Relations

We extend the definition of binary exclusion relations to arbitrary sets of propositions and actions. The order of each such set is its cardinality.

Definition 1 (Exclusion Relation over Propositions)**:** An exclusion relation over a set of propositions $P \subseteq \mathcal{P}$ at time t, denoted by $hoex_t(P)$ means that there is no plan with t parallel steps (t may be ∞) or less, that renders the set of propositions P true simultaneously.

For example, suppose that $hoex_t(\{p,q,r\})$ holds. Then propositions p, q and r cannot hold simultaneously at any proposition level of the planning graph up to level t. However, any subset of them, for instance $\{p, r\}$, may hold simultaneously at these levels.

There are two special cases of $hoex_t$ relations. The first is the case where $t=\infty$, i.e. the set or propositions can never be true simultaneously. The second case concerns singleton sets, e.g. $hoex_t(\{p\})$. The interpretation of unary *hoex* relations is that the involved proposition cannot be achieved at or before level t; this kind of information

is usually referred as "achievability analysis", but it can be seen as a specialization of the exclusion relation. In this article, both singleton and higher order *hoex* relations are treated uniformly.

The following Lemmas emanate naturally from Definition 1:

Lemma 1: $\forall\ P_1,P_2 \subseteq \mathcal{P} : P_1 \subseteq P_2 \wedge hoex_t(P_1) \Rightarrow hoex_t(P_2)$.

Lemma 2: $\forall\ P \subseteq \mathcal{P}$, $t_1,t_2 \in N$ and $t_2 < t_1$: $hoex_{t_1}(P) \Rightarrow hoex_{t_2}(P)$.

Lemma 1 is important since it imposes that only *minimal hoex* sets need to be considered. Furthermore, Lemma 2 imposes that only latest *hoexes* over a set of propositions should be retained. From now on, wherever we refer to *hoex* relations over sets of propositions, we will refer to minimal sets of propositions.

Similar definitions hold for *hoex* relations over actions:

Definition 2 (Exclusion Relation over Actions)**:** An exclusion relation over a set of actions $A \subseteq \mathcal{A}$ at time *t*, denoted by $hoex_t(A)$, means that there is no plan of *t* or less steps, where all actions in *A* can be executed simultaneously at time step *t* or earlier.

The following lemmas emanate naturally:

Lemma 3: $\forall\ A_1,A_2 \subseteq \mathcal{A} : A_1 \subseteq A_2 \wedge hoex_t(A_1) \Rightarrow hoex_t(A_2)$.

Lemma 4: $\forall\ A \subseteq \mathcal{A}$, $t_1,t_2 \in N$ and $t_2 < t_1$: $hoex_{t_1}(A) \Rightarrow hoex_{t_2}(A)$.

Similarly, we can characterize a *hoex* relation over a set of actions *A* as *minimal*, if no *hoex* relation between any proper subset of *A* holds at the same time. Furthermore, only latest *hoex* relations over sets of actions should be retained. In the sequel, *hoex* relations over sets of actions will refer to minimal sets of actions.

Proposition 1: A relation $hoex_t(A)$ holds over a minimal set of actions $A \subseteq \mathcal{A}$ at time *t*, iff:

a) *A* comprises two actions, a_1 and a_2, that are eternally mutexed, *or*

b) there is a set of propositions *P* such that $hoex_t(P)$ holds and, furthermore:

1. $P \subseteq \bigcup_{a_i \in A} Precs(a_i)$
2. $\forall a_i \in A, Precs(a_i) \cap P \neq \varnothing$
3. $\forall a_i \in A, (Precs(a_i) \cap P) - \bigcup_{\substack{a_j \in A, \\ j \neq i}} Precs(a_j) \neq \varnothing$

and, finally

c) there is no subset of *A* for which either (a) or (b) hold.

Proof: Case (a) derives from the definition. Regarding case (b), condition 1 imposes that all sets of actions in *A* cannot execute simultaneously, since *P* is a subset of the union of their preconditions. Conditions 2 and 3 ensure that the set of actions *A* is minimal, i.e., by removing any action from this set, condition 1 does not hold any more. Condition (c) ensures that there is no other set of *hoexed* propositions *P'*, such that condition (b2) holds for *P'* and a subset of *A*. ■

Finally, we can define a *hoex* relation over a set of actions *A* and a set of propositions *P* at time *t* as follows:

Definition 3 (Exclusion Relation over Propositions and Actions)**:** A set of non-*hoexed* actions $A \subseteq \mathcal{A}$ applicable at time *t* is *hoexed* with a set of non-*hoexed* propositions $S \subseteq \mathcal{P}$ at time *t*, denoted by $hoex_t(A,S)$, if there is no way to execute simultaneously all actions of *A* at *t*, while propositions of *S* being true at the same time *t*, i.e. at the start time of their execution.

Definitions 1 and 2 can be seen as generalizations of Definition 3, for the cases where $A=\varnothing$ or $S=\varnothing$ respectively.

Proposition 2: A set of non-*hoexed* actions $A \neq \varnothing$ may be *hoexed* with a set of non-*hoexed* propositions $S \neq \varnothing$ at time *t*, if there is a relation $hoex_t(P)$, such that:

1. $S \subset P$
2. $P - S \subseteq \bigcup_{a_i \in A} Precs(a_i)$
3. $\forall a_i \in A, Precs(a_i) \cap S = \varnothing$
4. $\forall a_i \in A, Precs(a_i) \cap (P - S) \neq \varnothing$
5. $\forall a_i \in A, (Precs(a_i) \cap (P - S)) - \bigcup_{\substack{a_j \in A \\ j \neq i}} Precs(a_j) \neq \varnothing$

Proof: In order to execute all actions of *A* at time *t*, all propositions of *P*-*S* must hold (conditon 2). So, all propositions of *S* cannot hold simultaneously, because in that case the $hoex_t(P)$ is violated due to condition 1. Conditions 3, 4 and 5 ensure that action set *A* is minimal. ■

Definition 3 can specialize for the case where set *A* comprises a single action, i.e., an action *a* is *hoexed* with a set of non-*hoexed* propositions $S \subseteq \mathcal{P}$ at time *t*, if there is no way to execute *a* at *t*, while propositions of *S* being true at the start time of the execution of *a*, i.e. at *t*.

Proposition 3: For each $hoex_t(a,S)$ relation and a set A of non-*hoexed* actions at time *t*, such that $a \in A$, $hoex_t(A,Q)$ also holds, where:

$$Q = S - \bigcup_{a_i \in A} Precs(a_i)$$

Proof: According to Proposition 2, $S \cap Precs(a) = \varnothing$, otherwise $hoex_t(a,S)$ is not minimal. Suppose now that some of the propositions in *S* appear in the preconditions of other actions of *A*, i.e. let's say that:

$$S' = \bigcup_{a_i \in A, a_i \neq a} Precs(a_i)$$

with $S \cap S' \neq \varnothing$. In that case, in order for actions in *A* to be simultaneously applicable, propositions in *S'* should all be true simultaneously at *t*. So, at least one of the propositions in $Q=S-S'$ must be false at *t*. Note that *Q* cannot be empty, since we assumed that actions of *A* are not *hoexed*. ■

Proposition 4: A set of propositions *P* is *hoexed* at time *t*, if there is no set *A* of non-*hoexed* actions at time *t*-1 that achieve a subset *S* of *P*, with the remaining propositions *P*-*S* neither being *hoexed* with *A* at *t*-1, nor being deleted by any of the actions in *A*.

Proof: If there is a set of actions *A* applicable simultaneously at *t*-1 that achieve *S* at *t*, and *P*-*S* is not *hoexed* with *A* at *t*-1, whereas no action $a \in A$ deletes any of the

propositions in P-S, then all propositions in P can be true simultaneously at t. If any one of the conditions does not hold, then at least one of the propositions in P cannot be true at t, so a relation $hoex_t(P)$ holds. ■

Finally, we define the notion of minimal *hoex* relations in a more formal way:

Definition 4 (Minimal Exclusion Relations)**:** Any relation $hoex_t(A,P)$ is minimal, if there is no other relation $hoex_{t'}(A',P')$, where $A' \subseteq A$ and $P' \subseteq P$ such that either of the following conditions hold:

- $|A'|+|P'|<|A|+|P|$ and $t' \geq t$.
- $|A'|+|P'|=|A|+|P|$ and $t'>t$.

Any *hoex* relation that it is not minimal can be subsumed by another *hoex* relation.

5. Computing higher order exclusion relations

This section comprises four parts. In the first, the algorithm that computes higher order *hoex* relations between propositions of the same time step and without any upper bound on their order, under a CNF representation, is presented. Then, we describe some implementation considerations for optimized code. Finally, we present two solution extraction methods, one without backtracking assuming unbounded exclusion relation computation and one using search for the case of bounded exclusion relation computation.

5.1. *Computing hoexes*

The detailed presentation of the *hoexes* computing algorithm requires some additional definitions and propositions that concern their propagation. In the following it is assumed that only minimal *hoexes* are retained.

Proposition 5 generalizes the well-known monotonicity property of mutexes in Graphplan, i.e. that the number of mutexes monotonically decrease (Long and Fox 1999). In this case, *hoex* relations only relax, i.e. their order increases or they disappear.

Proposition 5: For any *hoex* relation over a set of propositions P at level t of a planning graph, there exists at least one set of propositions $S \subseteq P$ that was *hoexed* at levet t-1 of the planning graph.

Proof: It is not possible for a set of propositions to be non-*hoexed* at some level t-1 of the planning graph, and being *hoexed* at the next level t, since if there is a plan of t-1 steps that achieves these propositions, then there is also a plan of t steps, with the last step being empty. So, either all propositions or some subset of them (a stronger condition) were *hoexed* at the previous level. ■

According to Proposition 5, new *hoex* relations are created only when earlier *hoex* relations break. Note that the initial state of a planning problem imposes unary *hoex* relations on all propositions that are not members of $\mathcal{I}$. So, each time a proposition is achieved for the first time at a planning graph level, new *hoex* relations that involve this proposition might arise. Similarly, each time a higher-order *hoex* relation breaks at some level, new *hoex* relations of an even higher order than the broken one might arise, each containing (among others) all the propositions of the broken relation.

Proposition 6: A relation $hoex_t(P)$ over a set of propositions P breaks at proposition level t+1, if there is an action a applicable at t that achieves a set of propositions S, such that $S \neq \varnothing$, $S \subseteq P$ and furthermore:

1. $hoex_t(a, P\text{-}S)$ does not hold.
2. $(P\text{-}S) \cap Dels(a) = \varnothing$

Proof: Since retained *hoex* relations are minimal, any subset of the involved entities can hold simultaneously. Thus, if the applicable action *a* achieves *S* at time *t*+1, we can assume that all propositions in *P-S*, which is a *proper* subset of P, could be true at *t*. Furthermore, since the set *P-S* is not *hoexed* with *a* at *t* as stated by the first condition, it is possible to apply *a* while all propositions in *P-S* hold. Finally, *a* does not delete any of the propositions in *P-S* (second condition), so they can hold even after *a*'s application. So, there is a plan of *t*+1 steps, with *a* being the single action of the last step, that achieves all propositions in *P*, so $hoex_{t+1}(P)$ does not hold. ■

Proposition 6 is very useful, because it states that a single action is needed to break a *hoex*, and this action just needs to achieve a single proposition of the *hoex*, without deleting any one of the remaining propositions, nor being *hoexed* with them.

Definition 5 (Applicable Set of Actions)**:** For a set of non-*hoexed* actions *A* applicable simultaneously at level *t*, with $negate_{t+1}(A)$ we denote the minimal set of sets of propositions that cannot hold simultaneously at *t*+1, provided that no action outside *A* has been applied at *t*. In this case, "minimal" refers to a set **S** of sets S_i, such that there are not two sets S_i, $S_j \in \mathbf{S}$ with $S_i \subseteq S_j$.

Proposition 7: For a set of actions *A* applicable at *t*, $negate_{t+1}(A)$ consists of the following sets:

1. $\{q\}$ for every $q \in Dels(a)$ of every $a \in A$
2. *Q*, for any relation $hoex_t(A,Q)$, such that

$$Q \bigcap \left(\bigcup_{a \in A} Adds(a) \right) = \varnothing$$

Proof: If actions of *A* are applied at *t*, then it is obvious that no delete effect of any of these actions can be true at *t*+1 (case 1). Furthermore, it no other action outside *A* has been applied at *t*, then any set of propositions *Q* that are *hoexed* with the set of actions *A* at *t* (consider Proposition 3 to progressively constructing such sets *Q*), should be added to $negate_{t+1}(A)$, provided that no action of *A* achieves any of them (in which case *Q* can be true at *t*+1, because any subset of *Q* could be true at *t*). We do not consider the case of any action of *A* deleting any of the propositions of *Q*, since we assume that all actions' delete effects are also their preconditions, so none of them should appear at any set *Q*, according to Proposition 2 (case 3). ■

Example 1: Suppose that there are two non-*hoexed* actions *a* and *b* applicable at *t*, with the following details: $a=\langle\{p,q\},\{p\},\{r\}\rangle$ and b $=\langle\{s\},\{\},\{u\}\rangle$. Suppose also that the following *hoexes* over propositions hold at *t*: $hoex_t(\{p,s,v\})$ and $hoex_t(s,r,y)$. According to Proposition 2, $hoex_t(\{a\},\{s,v\})$ holds for *a*, so $negate_{t+1}(\{a\}) = \{\{p\},\{s,v\}\}$. Similarly, $hoex_t(\{b\},\{r,y\})$ and $hoex_t(\{b\},\{p,v\})$ hold for *b*, so $negate_{t+1}(\{b\})=\{\{r,y\}, \{p,v\}\}$. Finally, $hoex_t(\{a,b\},\{v\})$ and $hoex_t(\{a,b\},\{r,y\})$ hold for the entire set of actions, so $negate_{t+1}(\{a,b\})=\{\{p\},\{v\}\}$. Note that $\{r,y\}$ is not included in $negate_{t+1}(\{a,b\})$, since action *a* achieves *r*.

The next two Propositions constitute the heart of the *hoex* computation algorithm. *Interesting* sets of actions are defined in Definition 7 and explained further in Section 5.2.

Proposition 8: For any relation $hoex_t(P)$ that breaks at level t+1 (according to Proposition 6), for all *interesting* sets of non-*hoexed* actions A_i at level t that can break this *hoex* relation (each set of actions A_i contains at least one action a that breaks $hoex_t(P)$), create new *hoex* relations till infinity for the sets of propositions $P \times \mathbf{N}$ where

$$\mathbf{N} = \prod_i negate_{t+1}(A_i)$$

and both $\times$ and $\prod$ refer to Cartesian products between sets. All new (existing) *hoex* relations subsumed by existing (new) relations should be ignored (retracted).

Proof: Set $\mathbf{N}$ is a set consisting of other sets. Each set $S \in \mathbf{N}$ comprises one element (set) from $negate_{t+1}(A_i)$ for every interesting non-*hoexed* set of actions A_i that break $hoex_t(P)$. Assuming that at least one of the sets of actions A_i will apply at t, then at least one of the sets that form S will have a proposition that is false at t+1. Otherwise, if no set of actions A_i applies at t, $hoex_t(P)$ won't break, so some proposition of P will be false at t+1. So, in any case and for any set $R \in P \times \mathbf{N}$, either some proposition of P will be false, or some proposition of each $S \in \mathbf{N}$ will be false, so new *hoex* relations for all sets of $P \times \mathbf{N}$ should be created. Minimality requirement imposes to keep only those *hoex* relations (new or existing) that are not subsumed by others. ■

Example 2: Continuing from example 1, suppose that u is a newly achieved proposition, i.e. $hoex_t(\{u\})$ breaks at t+1. There are two interesting ways to achieve u, through the following alternative sets of actions: $A_1=\{b\}$, $A_2=\{a,b\}$. In example 1 we have computed $negate_{t+1}(\{b\})$ and $negate_{t+1}(\{a,b\})$. So, their Cartesian product is:

$$\mathbf{N}= \{\{r,y,p\}, \{r,y,v\}, \{p,v\}, \{p,v\}\}$$

By removing duplicates, $\mathbf{N}$ becomes

$$\mathbf{N}= \{\{r,y,p\}, \{r,y,v\}, \{p,v\}\}$$

By taking the product $P \times \mathbf{N}$, new *hoexes* are created for the sets:

$$\{u,r,y,p\}, \{u,r,y,v\}, \{u,p,v\}$$

Changing slightly the example, let's assume that u was not a firstly achieved proposition, but there was a relation $hoex_t(\{u,z\})$ that broke at t+1 by the same set of sets of actions. In that case, nothing changes in the above analysis, except for the last step where the Cartesian product $P \times \mathbf{N}$ is computed. So, new *hoex* relations would be created at t+1 for the following sets of propositions:

$$\{u,z,r,y,p\}, \{u,z,r,y,v\}, \{u,z,p,v\}$$

Proposition 9: For any combination of relations $hoex_t(P_i)$ that break at level t+1 (according to Proposition 6), for all interesting sets of non-*hoexed* actions A_i at level t that can break simultaneously all these *hoex* relations (each set of actions A_i contains at least one action a_i for each broken relation $hoex_t(P_i)$ capable of breaking it), create new *hoex* relations at t+1 for the sets of propositions $P \times \mathbf{N}$ where P is the union of all propositions in P_i's and

$$\mathbf{N} = \prod_i negate_{t+1}(A_i)$$

and both $\times$ and $\prod$ refer to Cartesian products between sets. All new (existing) *hoex* relations subsumed by existing (new) relations should be ignored (retracted).

Proof: Proposition 9 generalizes Proposition 8 in that it considers simultaneously breaking of *hoex* relations. The only difference is that a set of *hoex* relations break simultaneously at t+1, so the union of their propositions can be true simultaneously at that time. This can be achieved by various alternative sets of actions A_i, imposing that some sets of propositions cannot be true simultaneously after A_i's application, i.e. $negate_{t+1}(A_i)$. So, similar to the proof of Proposition 8, either some of the A_i's has been applied, so any set in **N** has a false proposition originating from $negate_{t+1}(A_i)$, or none of the A_i's has been applied, so some of the $hoex(P_i)$ cannot break at t+1. In any case, the sets of the Cartesian product $P\times\mathbf{N}$ will always include a false proposition at t+1. ■

Lemma 5: If **N** is an empty set, no new *hoex* relations are created.

N might be an empty set, if for some set of actions A_i, $negate_{t+1}(A_i)=\varnothing$.

Lemma 6: If there is no set of actions A_i that break simultaneously all $hoex_t(P_i)$ relations, then a *hoex* relation over the union of some of the P_i's exists at t+1.

Definition 6 (Partial Ordering over Sets of Sets of Propositions)**:** Given two sets of sets of propositions, **R** and **S**, we say that **R** is more relaxed than **S**, denoted with $\mathbf{S}\prec\mathbf{R}$, iff for every $R\in\mathbf{R}$, there is an $S\in\mathbf{S}$ such that $S\subseteq R$.

For example, if $\mathbf{R}=\{\{p,q\},\{q,r\}\}$ and $\mathbf{S}=\{\{p\},\{r\}\}$, $\mathbf{S}\prec\mathbf{R}$ holds since for each set of **R** there is a subset of it in **S**. The "relaxed" relation imposes a partial ordering over sets of actions at the various time steps t, through their $negate_{t+1}$ sets.

Definition 7 (Interesting Sets of Actions)**:** A set of non-*hoexed* actions A at level t that break a set of $hoex_t(P_i)$ relations is interesting iff there is no other set of non-*hoexed* actions B at level t that break the same set of *hoex* relations such that $negate_{t+1}(\{B\})\prec negate_{t+1}(\{A\})$.

Proposition 10: Non-interesting sets of actions among those that break the same set of *hoex* relations can be ignored.

Proof: Suppose two sets of actions A and B breaking the same set of *hoex* relations at t+1, with $\mathbf{R}=negate_{t+1}(A)$ and $\mathbf{S}=negate_{t+1}(B)$, such that $\mathbf{S}\prec\mathbf{R}$**.** Any entry of the Cartesian product:

$$\mathbf{N_R}=\prod_{R\in\mathbf{R}}R$$

would be subsumed by some entry of the Cartesian product

$$\mathbf{N_S}=\prod_{S\in\mathbf{S}}S$$

Since we are interested in minimal *hoex* relations only, action set **R** is not interesting when breaking the particular set of *hoex* relations. ■

Lemma 7: If there is a set of actions A among those that break a set of *hoex* relations, such that $negate_{t+1}(A)=\varnothing$, no other set of actions needs to be considered.

We can now present an algorithm that computes all *hoex* relations between the propositions at each time step. The algorithm focuses on *hoex* relations over sets of propositions; however *hoex* relations over sets of actions or between sets of propositions and set of actions can be computed using Propositions 1 and 2.

Algorithm 1: Computing *hoex* relations over propositions.

1. Create $hoex_0(\{p\})$ relations for all propositions $p \notin \mathcal{I}$. Set t=0
2. While there are $hoex_t$ relations over $\mathcal{G}$ and the planning graph has not yet leveled off, repeat the following:
   a. Find the broken $hoex_t$ relations at level t+1, according to Proposition 6.
   b. Replace non-broken $hoex_t$ relations with $hoex_{t+1}$ relations.
   c. For each combination of broken $hoex_t$ relations, apply Proposition 9 to create additional new $hoex_{t+1}$ relations.
   d. Set t=t+1.

Step 1 concerns initialization, by recording all propositions that are not included in the initial state $\mathcal{I}$ as unary $hoexes_0$. Step 2 states the termination condition: Either achieving the goals $\mathcal{G}$, or leveling off the planning graph. Achieving $\mathcal{G}$ means that there is no $hoex_t$ relation over any subset of $\mathcal{G}$. Leveling off occurs if during the last iteration no *hoex* relation has broken.

Step 2a concerns breaking $hoex_t$ relations. According to Proposition 6, a single action is needed to break $hoex_t$, however we need to record all alternative actions that can be used for that purpose. Note that only newly applicable actions, i.e. actions that are firstly applied at the current planning graph level, can break unary *hoexes*.

Step 2b identifies the non-broken $hoex_t$ relations and replaces them with $hoex_{t+1}$ relations. Finally, step 2c generates new $hoex_{t+1}$ relations from the broken $hoex_t$ ones, by applying Proposition 9. As it is clear, this is the computationally most expensive step of the algorithm, so we will discuss it further in the section that concerns implementation details.

Algorithm 1 computes the *hoex* relations progressively, i.e. level by level. So, if the planning graph levels off at time step t, all $hoex_t$ relations could be considered eternal. If the algorithm terminates before the planning graph levels off, we cannot be verify which of the remaining $hoex_t$ relations are eternal and which are conditional.

5.2. *Implementation considerations*

The computationally demanding step of Algorithm 1 is step 2c, which iterates over each set of broken *hoexes*, computing all the alternative interesting sets of actions capable of breaking all the *hoexes* of the set. So, managing to reduce either the number of sets of broken *hoexes* that are considered, or the number of alternative sets of actions that break each set of *hoexes* is of great value.

Concerning the sets of broken *hoexes*, we can adopt a progressive approach, starting from small sets of broken *hoexes* and moving to larger ones. If for a set of broken *hoexes* there is no way to break all of them simultaneously, there is no reason to examine any superset of them (Lemma 6). This approach can easily be implemented using a depth-first search enumeration of the various subsets.

As for the alternative sets of actions that can break simultaneously a set of broken *hoexes*, one can start with minimal sets consisting of exactly one action for each broken *hoex* of the set (it might happen that the same action breaks more than one *hoexes* of the set). These are the initial seeds. Then, each such set A can be enhanced progressively with new actions a, such that a is non-*hoexed* with A and achieves something of $negate_{t+1}(A)$ (obviously not the propositions deleted by A's actions), such that $negate_{t+1}(A) \prec negate_{t+1}(A \cup a)$ does not hold. Propositions 3 and 7 help us to compute $negate_{t+1}(A \cup a)$ based on $negate_{t+1}(A)$ and $negate_{t+1}(a)$. In order to avoid considering adding the same sets of additional actions in different order, we can post a

lexicographic order on the new actions that are used to enhance the initial seeds.

However, even if alternative sets of actions are computed as described in the previous paragraph, it might happen that non-interesting sets of actions arise, originating from different initial seeds. So a check among the alternative sets of actions that break a set of *hoex* relations should be performed, before proceeding with the computation of the Cartesian product.

Finally, we found experimentally that a lot of time is spent in the computation of the Cartesian products. To be practically efficient, one could follow a divide-and-conquer approach, with the overall product being analyzed in subproducts that are computed recursively. After computing any such sub-product, deleting duplicates and subsumed sets of propositions accelerates significantly the overall process. This elimination is greatly facilitated by keeping the subproducts sorted based on the cardinality of their elements.

5.3. *Backtrack-free solution extraction*

As soon as no *hoex* exists among $\mathcal{G}$, we can proceed with the plan extraction phase. Having achieved leveled global consistency, plan extraction can be achieved in a backtrack-free way, using Algorithm 2.

Algorithm 2 works as follows: At each level a set of non-*hoexed* actions that break the related *hoex* relations of the previous level is selected (step 2c), with the requirement that no new *hoex* relations over the current set of open goals G is generated. The preconditions of the actions of A_i, as well as the not achieved open goals, constitute the new set of open goals for the previous level of the planning graph. Algorithm 2 *is not* a generalization of Graphplan's solution extraction procedure, since a plan generated by Graphplan may involve actions that achieve propositions not participating in any broken mutex at the current level.

Proposition 11: Provided that leveled global consistency has been enforced, for every set of not *hoexed* propositions G at planning graph level t, there is always a set of non-*hoexed* actions A_i at t-1 such that:

a) No subset of G appears in $negate_t(A_i)$
b) G' defined as in step 2d of Algorithm 2 is *hoex*-free at level t-1.

Algorithm 2: Backtrack-free solution extraction.

1. Set G be initially equal to $\mathcal{G}$. Let t be the first level of the planning graph, where all propositions of $\mathcal{G}$ appear without any *hoex* between them. Set $Plan=\{Step_0, Step_1, \ldots, Step_{t-1}\}=\{\varnothing, \varnothing, \ldots, \varnothing\}$
2. Execute repeatedly the following steps:
   a. If $G \subseteq \mathcal{I}$, return *Plan* and stop.
   b. Let *Broken* be the set of all *hoex* relations over the propositions of G at level t-1.
   c. Find an interesting set of actions A_{t-1} not-*hoexed* at t-1, that break simultaneously all *hoex* relations in *Broken*, such that no subset of G appears in $negate_t(A_{t-1})$.
   d. Let G' be the union of the preconditions of the actions in A_{t-1} and of the subset of propositions of G that are not achieved by any action of A_{t-1}, i.e.,

$$G' = \left(\bigcup_{a \in A_{t-1}} Precs(a) \right) \cup \left(G - \bigcup_{a \in A_{t-1}} Adds(a) \right)$$

   e. Let $Step_{t-1}=A_{t-1}$. Let $t=t$-1. Let $G = G'$.

Proof: Suppose that there is no *hoex* relation over propositions of G at level t-1. In this case the proof is trivial, since A_i can simply be the empty set of actions.

Now, suppose that *Broken* is the non-empty set of subsets of G that are *hoexed* at level t-1. Let $G_1 \subseteq G$ the subset of G comprising the union of all propositions appearing in *hoex* relations of *Broken*. So, we need to find a set of actions A_i that can break all *hoex* relations in *Broken* simultaneously. This set of actions should exist, since otherwise a *hoex* relation over G_1 should have been created at level t, according to Lemma 6. Furthermore, there should exist at least one option A_i that does not include any subset of the remaining propositions G-G_1 in $negate_t(A_i)$, otherwise if all options A_i would include at least one $S_i \subseteq G$-G_1 in $negate_t(A_i)$, then the Cartesian product would create, among others, a new *hoex* relation over the set $G_1 \cup S$, where S is the union of the various S_i's . Clearly, $G_1 \cup S$ is a subset of G, so this is a contradiction since we assumed that no *hoex* exists over G at t.

Since no subset of G appears in $negate_t(A_i)$, no *hoex* can hold between preconditions of A_i and sets of non-achieved propositions of G from actions of A_i. Indeed, if there was any *hoex* relation between some preconditions of A_i and some propositions $G_2 \subset G$ that are not achieved by any action of A_i, then propositions in G_2 should appear in $negate_t(A_i)$, which is a contradiction. Furthermore, no *hoex* relation exists between sets of not achieved propositions of G, since we assumed that A_i breaks all *hoex* relations between propositions of G, so A_i should achieve at least one proposition for each broken *hoex* relation. So, there is no *hoex* relation over the set comprising the preconditions of all actions of A_i and the not achieved propositions of G. If this set is referenced as G', we proved that by starting with a set of non-*hoexed* propositions at level t, we result in a new set of non-*hoexed* propositions at level t-1. ■

Lemma 8: Provided that leveled global consistency has been enforced up to the level where G appears without any *hoex* over its propositions, Algorithm 2 extracts a plan in a backtrack-free manner.

5.4. *Search based solution extraction*

Enforcing leveled global consistency for planning problems is impractical for most but the smallest problem instances. The most common approach is to compensate constraint propagation with search, i.e., more constraint propagation results in less search and vice versa. Traditional Graphplan computes binary mutexes only, so it heavily depends on search (and memoization) in order to extract plans. However, it is known that Graphplan is not very efficient with respect to other planning frameworks, e.g., heuristic state-space search or planning as satisfiability. So, one direction to improve Graphplan's performance is to compute higher order *hoex* relations up to a specific order, thus investing more time in constraint propagation in order to reduce search.

Algorithm 2 should be extended in order to cover the cases where a bound has been set on the order of the computed *hoex* relations. Algorithm 3 presents the generalized algorithm for solution extraction using search. Being a generalization of Graphplan's solution extraction procedure, Algorithm 3 differs in several ways from Algorithm 2. In particular, after failing to extract a plan when starting from a proposition level where all goal propositions appear without any *hoex* relation between them, it extends the planning graph by an additional pair of action and proposition levels, computes the new *hoex* relations and attempts to extract a solution again (step 4).

Most importantly, at each proposition level it considers achieving more propositions than just one for each broken *hoex* relation. In particular, after finding a not-*hoexed* set of actions that break all propositional *hoex* relations that were firstly broken at this proposition level, it tries to extend this set of actions with additional actions that achieve some of the remaining goals (step 3e). This is necessary because, without computing all the *hoex* relations it might happen that at some level t where solution extraction is attempted, no *hoex* between the goals is broken, which results in empty set A_{t-1}. It is obvious that for any broken relation between goal propositions at t, at least one action that achieves some of these propositions without deleting any other goal proposition should be included in the plan. However, in order to extract a solution plan, it might be necessary to achieve more goal proposition at the current proposition level. This requirement does not hold in case of Algorithm 2.

Finally, Algorithm 3 memoizes each unsuccessful attempt to achieve goal set G at level t, by creating a new $hoex_t$ relation over G (step 3h). Note that t+1 is a lower estimate for the time when the new $hoex_t$ relation breaks; as Algorithm 3 proceeds, it might happen that propositions of G or any subset of them are proven unachievable for even later time steps. This will result in retracting memoized *hoex* relations in the emergence of new stronger ones.

Algorithm 3: Solution extraction using search

1. Let L be the first level of the planning graph, where all propositions of $\mathcal{G}$ appear without any *hoex* between them.
2. Set $G=\mathcal{G}$. Set $t=L$. Set $Plan=\{Step_0, Step_1, \ldots, Step_{t-1}\}=\{\varnothing, \varnothing, \ldots, \varnothing\}$
3. Do:
   a. If $G \subseteq \mathcal{I}$, display the *Plan* and stop.
   b. Let *Broken* be the set of all *hoex* relations over the propositions of G at level t-1.
   c. Find (non-deterministically) an interesting set of actions A_{t-1} not-*hoexed* at t-1, that break simultaneously all *hoex* relations in *Broken*, such that no subset of G appears in $negate_t(A_{t-1})$.
   d. Let $NoOp \subseteq G$ be the propositions not achieved by any action $a \in A_{t-1}$.
   e. Enhance (non-deterministically) A_{t-1} with an interesting set of new actions B_{t-1}, such that the set of actions $A_{t-1} \cup B_{t-1}$ is compatible and for any action $b \in B_{t-1}$
      $$Adds(b) \cap NoOp \neq \varnothing$$
      $$Dels(b) \cap G = \varnothing$$
      A set B_{t-1} is interesting, if by removing any action $b \in B_{t-1}$, there is at least one proposition $p \in NoOp \cap Adds(b)$ that is no longer achieved by the remaining actions in B_{t-1}. The empty set, $B_{t-1}=\varnothing$, is always an interesting set.
   f. Let
      $$G' = \bigcup_{a \in A_{t-1} \cup B_{t-1}} Precs(a) - \left(G - \bigcup_{a \in A_{t-1} \cup B_{t-1}} Adds(a) \right)$$
      i.e., G' comprises the preconditions of the actions in $A_{t-1} \cup B_{t-1}$ and the subset of propositions of G that are not achieved by any action of $A_{t-1} \cup B_{t-1}$.
   g. Let $Step_{t-1} = A_{t-1} \cup B_{t-1}$. Call recursively step 3 with $t=t$-1 and $G=G'$.
   h. If no plan has been found that achieves G at t, create $hoex_t(G)$, retract any existing hoex relation subsumed by the new one, and return.
4. If no plan has been found in Step 3, set $L=L$+1, extend the planning graph by one action and proposition level and go to step 2.

Memoization creates *hoex* relations without any bound on their order, even if such a bound has been set for the *hoex* relations computed during the planning graph expansion. So, the lower is the bound on the order of *hoex* relations, the more significant is the contribution of the memoization to time needed for plan extraction. On the other hand, in case of unbounded *hoex* computation, memoization has nothing to contribute to the solution extraction procedure, since all the *hoex* relations between problem's propositions have been computed during the planning graph expansion phase.

6. Empirical Evaluation

This section comprises two parts. Section 6.1 demonstrates the soundness and completeness of the proposed algorithm, by solving some representative problems from various domains in a backtrack-free manner and reporting statistics on the number of *hoexes* created, as well as some instances of the computed *hoex* relations. Section 6.2 uses our implementation of Algorithm 3 to explore various tradeoffs between the bound set on the order of the computed *hoex* relations and the time needed to solve the planning problems.

At this point we want to emphasize that our implementation does not aim at competing with other state of the art planning systems. It is a declarative implementation in Prolog with no special data structures or code optimization. So, it is not surprising that running times for the case of binary mutexes are worse than of other planning systems. We expect that, using the experimental results as a proof of concept, state of the art planning systems will take advantage of them.

6.1. *Solving planning problems without backtracking*

We implemented the proposed algorithms, in particular Algorithms 1 and 3 (Algorithm 2 is a specialization of Algorithm 3) in Prolog. More specifically, we have used the ECLiPSe Constraint Logic Programming Platform[3] for running our experiments, although we haven't exploited any of the constraint programming features of the language. In this section we demonstrate in three domains the ability of our implementation for backtrack-free problem solving and report several statistics and instances of computed higher order *hoex* relations. We do not report solution time; this issue is tackled in Section 6.2. However, we can report that in all experiments mentioned in this Section the contribution of the plan extraction phase to the total time was negligible with respect to the planning graph expansion phase, since problems have been solved optimally in a backtrack-free manner[4].

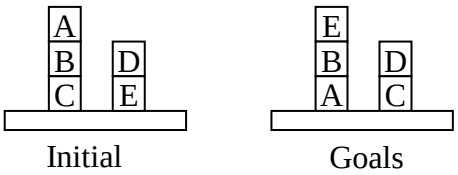


Figure 1. A simple blocks world problem with five blocks.

First we present an example from the blocks world domain represented with three operators (Figure 1). Table 1 summarizes the *hoex* relations found on this problem up to level 3, where a solution was found. The term *choex* refers to conditional *hoexes*,

[3] http://87.230.22.228/

[4] The code and the problem files used for the experimentation are available through http://eos.uom.gr/~yrefanid/hoex.zip.

i.e. hoex relations that were broken before the last level, whereas *ehoex* refers to the unbroken *hoex* relations.

Table 1. *hoexes* found for the five blocks problem up to goal achievement.

Order	*choexes*	*ehoexes*	Total
2	28	134	162
3	6	67	73
4	0	2	2
Totals	34	203	237

Continuing the execution of Algorithm 1 until the graph levels off, the *hoexes* shown in Table 2 were discovered.

Table 2. *hoexes* found for the five blocks problem up to level off.

Order	*choexes*	*ehoexes*	Total
2	52	110	162
3	113	20	133
4	94	30	124
5	0	24	24
Totals	259	184	443

Examples of instances of eternal *hoexes* of higher order are:

$$hoex_{\infty}(\{on(a,b),\ on(b,d),\ on(d,a)\})$$
$$hoex_{\infty}(\{on(a,b),\ on(b,d),\ on(d,e),\ on(e,a)\})$$
$$hoex_{\infty}(\{\ on(a,b),\ on(b,c),\ on(c,d),\ on(d,e),\ on(e,a)\})$$

whereas instances of broken *hoexes* are the following:

$$hoex_{2}(\{on(b,table),\ on(d,a),\ on(e,d)\})$$
$$hoex_{3}(\{on(c,e),\ on(d,a),\ on(e,d)\})$$
$$hoex_{3}(\{on(a,d),\ on(b,a),\ on(c,b),\ on(d,table)\})$$

We claim than broken *hoex* relations are more interesting for plan extraction, since they are related to the specific problem instance. On the other hand, eternal *hoex* relations encode domain knowledge and could be provided manually.

The second problem is *strips-gripper-x-1* from the gripper domain of the 1st International Planning Competition. According to the problem, there exist two rooms and a robot that can move between them. The robot has two grippers (*left* and *right*) that can carry balls. There are four balls in *rooma* that must be moved to *roomb*. The operators of the domain are *move* for the robot, *pick* and *drop* for each gripper. An optimal plan for this problem has 6 steps, where each triple of steps consists of two *pick*, one *move* and two *drop* actions. The graph levels off at level 8. Table 2 presents the *hoexes* after the graph has leveled off.

Table 3. *hoexes* found for the gripper problem up to level off.

Order	*choexes*	*ehoexes*	Total
2	12	45	57
3	52	0	52
4	52	0	52
Totals	116	45	161

Examples of *hoexes* found are the following:

$hoex_4$({*at(ball4,roomb),carry(ball2,right), carry(ball3,left)*})
$hoex_5$({*at_robby(roomb),at(ball4,roomb), carry(ball2, right), carry(ball3, left)*})

Finally, we present the application of the proposed algorithms to the *Flat Tire* domain and specifically the problem *fixit*, which was also used in (Blum and Furst 1997). This problem concerns changing a tire using actions such as jacking the car up and down, loosen and tighten the nuts, opening and closing the car boot, fetching and putting away objects in the car boot etc. Table 4 summarizes the numbers of *hoexes* computed for the various orders after the planning graph leveled off.

Table 4. *hoexes* found for the Flat Tire "fixit" problem.

Order	*choexes*	*ehoexes*	Total
2	38	28	66
3	51	0	51
4	28	0	28
5	8	0	8
Totals	125	28	153

Examples of *hoexes* found are the following:

$hoex_9$({*closed*(*boot*), *in*(*jack*,*boot*), *in*(*wheel1*,*boot*), *in*(*wrench*,*boot*), *loose*(*nuts*,*hub*)})
$hoex_8$({*in*(*jack*,*boot*), *in*(*wheel1*,*boot*), *in*(*wrench*,*boot*), *loose*(*nuts*, *hub*)})
$hoex_{11}$({*closed*(*boot*), *in*(*wrench*, *boot*), *on*(*wheel2*, *hub*), *tight*(*nuts*, *hub*)})
$hoex_4$({*closed*(*boot*), *in*(*wrench*, *boot*), *loose*(*nuts*, *hub*)})
$hoex_2$({*free*(*hub*), *on_ground*(*hub*)})
$hoex_2$({*closed*(*boot*), *have*(*wrench*)})

The set of examples is neither extensive nor diverse enough to extract strong conclusions. Nevertheless, it is worth mentioning that in all three cases, the total number of exclusion relations (sum of *choexes* and *ehoexes*) decreases with the order. Of course this is not a general rule; however it seems to be a situation common in many of the benchmark domains used in the planning literature. Our intuition suggests that breaking this rule, i.e., having more higher order exclusion relations than lower order ones, would indicate more complex problems. So, this analysis is an open issue for future research.

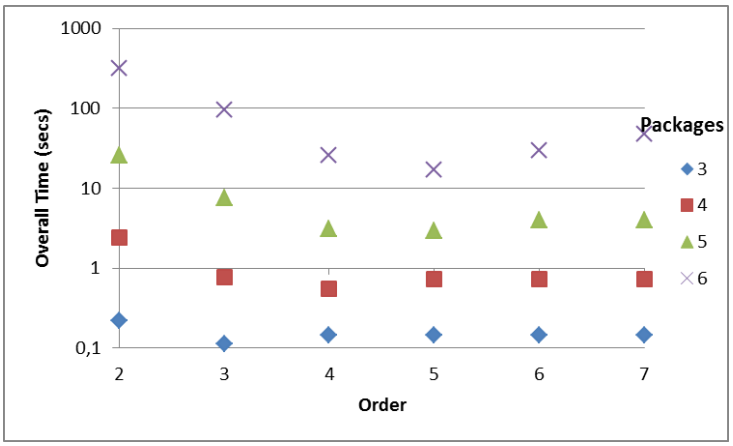
6.2. *Compensating constraint propagation and search*

In this section we explore some tradeoffs between constraint propagation and search for various planning problems in a Graphplan framework, using Algorithm 3. It is expected that for most planning problems, computing binary mutexes only and investing in search for extracting a plan is the best strategy. However, we also expect that there are some problems for which computing higher order *hoex* relations (not unbounded necessarily) is a more efficient option. Particularly, we expect that computing higher order *hoex* relations will be more efficient for problems for which binary mutex computation results in having all goal propositions non-mutexed at a much earlier planning graph level than the actual one, i.e., the one at which a solution can be found. In this case, the extra higher order *hoex* computation time will be

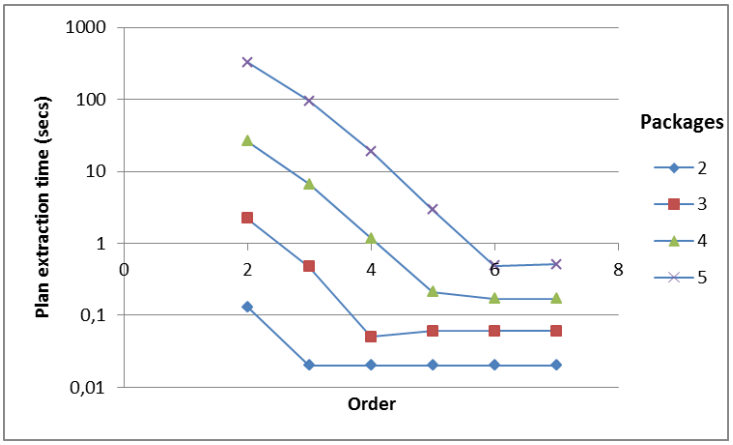
compensated by the significantly less search time.

For the experiment, we devised a simple logistics-type domain with a small number of trucks, two connected locations A and B and some packages. Initially all the trucks and packages are located in A and the goal is to transport the packages to B. The domain is modeled with the usual `load`, `unload` and `move` operators. However, there is a constraint on the capacity of each truck, i.e., at most one package can be loaded on every truck at any time.

Consider a problem instance with one truck and four packages; it is obvious that any pair of packages can be at location B in 7 steps: `load` the first package, `move` from A to B and `unload` the first package, `move` the truck back to A, `load` the second package, `move` from A to B and `unload` the second package. So, all binary mutexes between packages vanish at the 7th level of the planning graph. However, it is clear that in order to move all the four packages to location B, a plan of 15 steps is needed. Table 5 and Figure 2a present the time needed to solve the problem for one truck, several packages and several upper bounds on the order of the computed *hoex* relations. Furthermore, Table 5 analyses the running time in constraint propagation (i.e., *hoex* computation) time and search (i.e., plan extraction) time, whereas it also reports the number of backtracks for the search phase. The results have been taken on a Turion 64 X2 machine with 4GB physical memory, running Windows 7 OS.



(a)



(b)

Figure 2. (a) Overall time needed to solve the problems of Table 5. (b) Plan extraction time for the same problems. Vertical axes are in a logarithmic scale. Order 7 refers to inf.

Table 5. Statistics for solving logistics problems with one truck of unary capacity, two locations, several packages that need to be transported from one location to the other, for various upper bounds on the order of the computed *hoex* relations. The first line in each cell gives the total time to solve the problem. The second line analyzes this time in *hoex* computation time and plan extraction time, and the third line gives the total number of backtracks. Times are given in seconds.

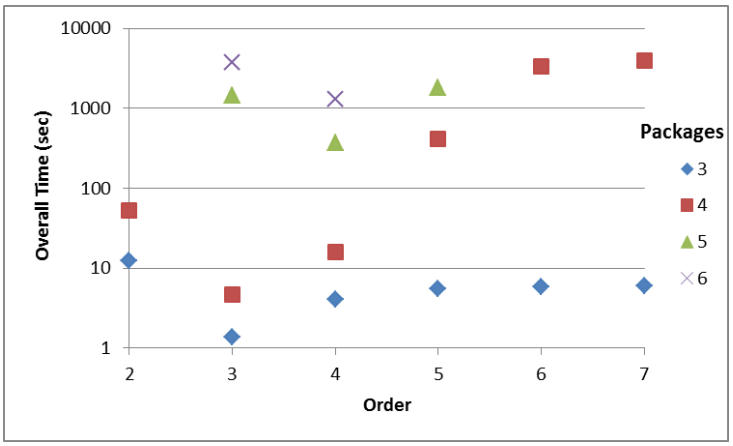
	Order					
Packages	2	3	4	5	6	inf
3	0.19 (0.06/0.13) 36	0.11 (0.09/0.02) 3	0.14 (0.12/0.02) 0	0.14 (0.12/0.02) 0	0.14 (0.12/0.02) 0	0.14 (0.12/0.02) 0
4	2.40 (0.17/2.23) 218	0.80 (0.33/0.47) 52	0.55 (0.50/0.05) 4	0.78 (0.72/0.06) 0	0.78 (0.72/0.06) 0	0.78 (0.72/0.06) 0
5	26.61 (0.38/26.23) 880	7.69 (1.01/6.68) 330	3.23 (2.05/1.18) 70	3.05 (2.84/0.21) 5	4.06 (3.89/0.17) 0	4.06 (3.89/0.17) 0
6	329.13 (0.75/328.38) 2938	98.06 (2.56/95.50) 1407	26.48 (7.50/18.98) 471	17.20 (14.27/2.93) 90	29.11 (28.62/0.49) 6	48.92 (48.41/0.51) 0

Table 6. Statistics for solving logistics problems with two trucks of unary capacity, two locations, several packages that need to be transported from one location to the other, for various upper bounds on the order of the computed *hoex* relations. The first line in each cell gives the total time to solve the problem. The second line analyzes this time in *hoex* computation time and plan extraction time, and the third line gives the total number of backtracks. Times are given in seconds. Time limit set to 90 mins.

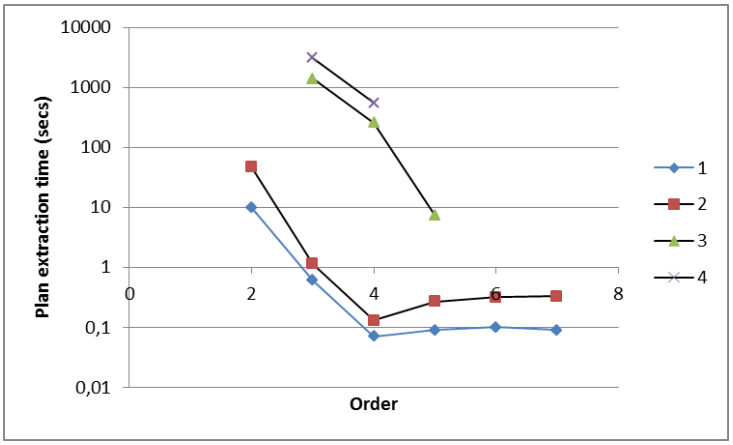
	Order					
Packages	2	3	4	5	6	inf
3	10.92 (0.09/10.02) 128	1.31 (0.7/0.61) 12	3.88 (3.81/0.07) 0	5.42 (5.33/0.09) 0	5.69 (5.59/0.10) 0	5.95 (5.86/0.09) 0
4	47.66 (0.28/47.38) 250	4.59 (3.45/1.14) 16	15.66 (15.53/0.13) 0	397.69 (397.42/0.27) 0	3297.12 (2396.8/0.32) 0	3928.73 (3928.4/0.33) 0
5	-	1401.72 (14.72/1387) 1048	355.61 (99.62/255.99) 349	1748.8 (1741.31/7.49) 30	-	-
6	-	3137.38 (55.81/3081.57)	1049.33 (499.76/549.57) 537	-	-	-

As it can be seen from these results, the function between the upper bound on the order of the computed *hoex* relations and the total time needed to solve the planning problem demonstrates a minimum. Furthermore, it is quite interesting that the order at which the minimum in the total time appears increases with the size of the problem. So, with three packages we have the minimum solution time when computing *hoex* relations of order 3, with four packages when computing *hoex* relations of order 4 and with five and six packages when computing *hoex* relations of order 5. The same behavior is observed also in the gripper and openstacks domains (see Table 9). However, extensive experiments are required in order to generalize this result. As expected, the percentage of the time devoted to search, as well as the number of backtracks significantly decrease for higher bounds on the order of the computed

exclusion relations. As can be seen from Figure 2b, it seems to get an exponential decrease in the time devoted to search, up to the order where all *hoex* relations have been computed.



(a)



(b)

Figure 3. (a) Overall time needed to solve the problems of Table 6. (b) Plan extraction time for the same problems. Vertical axis is in a logarithmic scale. Order 7 refers to inf.

The results with problem instances that involve two trucks are similar. Note that in this case the number of the problems' propositions increases (almost it doubles for the same number of packages), so the time needed to compute higher order *hoex* relations increases significantly. Table 6 and Figure 3 present the results. A limit of 90 mins has been set for solving these problems.

An interesting statistic concerns the analysis of backtracks per layer. Since space does not allow us to give these statistics for all problems, we use for demonstration the case with one truck and six packages. As can be seen from Table 7, the higher the bound on the order of the *hoexes* computed during the planning graph expansion phase, the closer to the last level backtracks occur.

Another interesting statistic concerns the number and the order of the *hoex* relations computed during the planning graph expansion phase and during the search phase through memoization. As an example, we consider once more the case of one truck and six packages. Table 8 presents the results for various bounds on the order of the *hoexes* computed during the planning graph expansion phase (no bound exists for the *hoexes* computed through memoization).

Table 7. Number of backtracks per layer of the planning graph for various bounds on the order of the computed *hoexes* during the planning graph expansion phase. A problem with one truck and six packages has been used for these data.

		Bound on *hoex* order during the planning graph construction					
		2	3	4	5	6	inf
Planning graph level	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	30	0	0	0	0	0
	7	424	0	0	0	0	0
	8	409	0	0	0	0	0
	9	394	0	0	0	0	0
	10	334	60	0	0	0	0
	11	274	274	0	0	0	0
	12	254	254	0	0	0	0
	13	234	234	0	0	0	0
	14	174	174	60	0	0	0
	15	114	114	114	0	0	0
	16	99	99	99	0	0	0
	17	84	84	84	0	0	0
	18	54	54	54	30	0	0
	19	24	24	24	24	0	0
	20	18	18	18	18	0	0
	21	12	12	12	12	0	0
	22	6	6	6	6	6	0

Table 8. Number of *hoex* relations computed for the same problem (one truck, four packages) through memoization vs unbounded *hoex* computation.

		Bound on *hoex* order during the planning graph construction					
		2	3	4	5	6	inf
hoex per order after memoization	2	105	105	105	105	105	105
	3	0	146	146	146	146	146
	4	0	0	190	190	190	190
	5	0	0	0	146	146	146
	6	1	1	1	1	64	64
	7	0	0	0	0	0	77
	8	454	334	174	54	6	0
	>8	0	0	0	0	0	0
Totals		560	586	616	642	657	728

Several interesting observations can be drawn from Table 8. First, the planning graph construction phase computes always the same number of *hoex* relations for any order not exceeding the bound. Taking into account that exactly these numbers of *hoexes* are computed in the case of no bound, this is an experimental indication that all *hoexes* up to the bound are computed. However, this observation does not generalize. Indeed, suppose a bound B and an N-ary *hoex* with $N>B$, i.e., this *hoex* is not computed. It is possible to have an action which, when considering the N-ary hoex, is inapplicable; however with the lower bound B is now applicable. This action might break a lower order *hoex* that should not break. So, setting a bound on the order of the computed *hoex* relations does not guarantee that all *hoex* relations up to this bound are computed.

Table 9. Experimental results for small problems from several domains. The first line in each cell gives the total time to solve the problem. The second line analyzes this time in *hoex* computation time and plan extraction time, and the third line gives the total number of backtracks. Times are given in seconds. Time limit set to 90 mins. All problems are available from http://eos.uom.gr/~yrefanid/hoex.zip.

Problem	Order					
	2	3	4	5	6	inf
Flat tire "fixit"	2.91 (0.14/2.77) 41	0.59 (0.48/0.11) 0	1.02 (0.92/0.10) 0	2.09 (1.97/0.12) 0	5.53 (5.42/0.11) 0	24.13 (24.03/0.10) 0
strips-gripper-x-1	5.52 (0.19/5.33) 86	1.73 (0.77/0.96) 13	2.19 (2.15/0.04) 0	2.69 (2.64/0.05) 0	2.69 (2.64/0.05) 0	2.69 (2.64/0.05) 0
strips-gripper-x-2	-	562.83 (11.4/551.43) 767	156.14 (85.98/70.16) 174	1523.3 (1503.5/19.8) 31	-	-
blocks-04	0.06 (0.06/0) 0	0.13 (0.11/0.02) 0	0.13 (0.11/0.02) 0	0.13 (0.11/0.02) 0	0.13 (0.11/0.02) 0	0.13 (0.11/0.02) 0
blocks-05	0.80 (0.78/0.02) 0	2.23 (2.22/0.01) 0	2.95 (2.92/0.03) 0	3.14 (3.12/0.02) 0	3.25 (3.22/0.03) 0	3.27 (3.23/0.04) 0
blocks-06	13.95 (13.86/0.09) 0	154.81 (154.24/0.57) 0	422.4 (421.79/0.61) 0	684.92 (684.3/0.62) 0	768.83 (768.19/0.64) 0	850.28 (849.65/0.63) 0
openstacks-03	0.78 (0.14/0.64) 31	0.89 (0.84/0.05) 0	1.63 (1.59/0.04) 0	2.09 (2.07/0.02) 0	2.48 (2.45/0.03) 0	2.83 (2.78/0.05) 0
openstacks-04	93.73 (0.64/93.09) 368	10.38 (9.86/0.52) 10	39.06 (38.92/0.14) 0	85.94 (85.73/0.21) 0	146.77 (146.57/0.2) 0	248.36 (248.14/0.22) 0
pegsol-01	0.47 (0.42/0.05) 0	0.72 (0.64/0.08) 0	0.97 (0.92/0.05) 0	1.36 (1.31/0.05) 0	1.84 (1.78/0.06) 0	6.34 (6.27/0.07) 0
pegsol-02	48.53 (45.57/2.96) 10	71.09 (69.58/1.51) 1	242.28 (240.27/2.01) 0	817.17 (815.72/1.45) 0	3017.63 (3016.3/1.33) 0	-
pegsol-03	99.39 (93.13/6.26) 13	529.61 (527.71/1.9) 0	2778.02 (2775.72/2.3) 0	-	-	-

A second observation is that memoization computes *hoex* relations of order 8, with negligible *hoex* computation of lower orders. The higher the bound on the order of the *hoexes* computed by the planning graph expansion phase, the less the *hoexes* computed by memoization. Furthermore, in case of unbounded *hoex* computation, no *hoex* of order 8 is computed. This means that all *hoex* relations that were computed by memoization are not minimal, i.e., they could be subsumed by other *hoex* relations of lower order. This is due to the completeness of the algorithms proposed in this article, compared to the incompleteness of the memoization technique. Nevertheless, we have to emphasize that memoization is of great importance when bounded *hoex* computation occurs in the graph expansion phase. For example, without memoization and allowing binary *hoex* computation only, the problem with one truck and six packages cannot be solved within the time limit. Finally, it is interesting to note that the total number of *hoexes* increases with the bound on their order, however their average order decreases.

The above results by no means should be considered as a general suggestion to compute higher order exclusion relations in planning problems in order to reduce search. Indeed, for most of the planning problems used as benchmarks in the literature

or in the International Planning Competitions[5], the combination of binary exclusion computation and search is the most efficient strategy. Of course, this observation should be taken into account by the planning community, since it seems that most of the domains used in the literature follow similar patterns that make them easily confronted by modern planners.

In Table 9 we present some experimental results on small problems from various domains, taken from the literature and from the planning competitions. As it can be seen, there are several cases where computing higher order exclusion relations in the graph expansion phase results in better overall performance than binary mutex computation. Indeed, problems "fixit", "strips-gripper-x-1" and "openstacks-04" are solved faster when ternary exclusion relations are computed, whereas "strips-gripper-x-2" is solved faster when quaternary exclusion relations are computed. On the other hand, "blocks" and "pegsol" problems are solved faster when only binary exclusion relations are computed. It is also interesting to note that blocks world problems are solved without backtracking even for the case when only binary mutexes are computed. However, this result does not generalize to larger instances from this domain.

7. Conclusions and Future Work

In this article we presented a systematic and complete method to compute, under a CNF representation, *all* higher order exclusion relations between propositions of a planning problem within each time step, without any bound on the order of the computed relations. We denote each such relation over a set of propositions P with $hoex_t(P)$, meaning that at least one of the propositions of P cannot be true at any planning graph level $t' \leq t$. We also defined *hoex* relations over sets of actions, as well as over sets of actions and sets of propositions, and we proved several interesting properties for them.

We have also presented an algorithm that extracts optimal parallel plans in a backtrack-free manner, thus proving the completeness of the proposed *hoex* computation method. We also presented a generalization of the Graphplan's plan extraction procedure, for the case of bounded *hoex* computation. As a proof of concept we implemented the proposed algorithms and illustrated their effectiveness in several small sized planning problems that were solved optimally and in a backtrack-free manner.

Solving planning problems in a backtrack-free manner is however impractical for most realistic planning problems. The practical importance of our work is that, by setting a bound on the order of the computed exclusion algorithms, search can be employed to solve the planning problem. So, we explored the tradeoffs between the order of the computed exclusion relations and the overall performance of a Graphplan style planner. We identified several cases where computing binary mutexes is not the optimal strategy; indeed, in some cases the optimal performance has been achieved when computing exclusion relations of order five. Finally, we showed that the traditional memoization technique of Graphplan, although very effective, results in computing non-minimal exclusion relations.

There is future work in several directions. Following an efficient implementation of the proposed complete *hoex* computation method, it is interesting to compare it to exhaustive construction of the state space using BDDs. We expect that working in the space of exclusion relations with suitable data structures will be more efficient in

[5] http://ipc.informatik.uni-freiburg.de/

several domains. We would like also to examine how the *hoex* computation algorithm could take advantage of precomputed *hoex* relations, which could be provided either by hand as domain knowledge or could be obtained automatically through domain analysis. Another research direction could be to combine our work on higher order exclusion relations with existing work in temporal planning graphs and long distance exclusion relations, in order to achieve general global consistency and confront problems with temporally annotated goals. We also plan to extend our experimentation with the various tradeoffs between constraint propagation and search to other planning frameworks, such as heuristic state space planners and satisfiability planners. The best tradeoff between *hoex* computation and search could serve as an indication of the difficulty to solve problems of particular domains with domain independent planners. An interesting idea is to interleave *hoex* computation with search, i.e., start with low order *hoex* computation and, each time search sticks, proceed with higher order *hoex* computation. Such a planner could take advantage of parallel machines.

References

Anderson, C.R., Smith, D.E. and Weld, D.S. (1998), 'Conditional Effects in Graphplan', in *Proceedings of the Artificial Intelligence Planning Systems* (AIPS) Conference.

Bernandini, S. and Smith, D.E. (2011), 'Finding Mutual Exclusion Invariants in Temporal Planning Domains', in *Proceedings of the 7th International Workshop on Planning and Scheduling for Space* (IWPSS 2011), Darmstadt, Germany.

Blum, A. and Furst, M. (1997), 'Fast planning through planning graph analysis' *Artificial Intelligence*, 90, 281-300.

Bonet, B., Loerincs, G. and Geffner, H. (1997), 'A robust and fast action selection mechanism for planning', in *Proceedings of AAAI-97*.

Bryan, R.E. (1985), 'Symbolic manipulation of Boolean functions using a graphical representation', in *DAC*, pages 688-694.

Chen, Y., Huang, R., Xing, Z. and Zhang W. (2009), 'Long-distance mutual exclusion for planning', *Artificial Intelligence*, 173 (2), 365-391.

Cimatti, A., Giunchiglia, E., Giunchiglia, F. and Traverso, P. (1997), 'Planning via model checking: a decision procedure for AR', in *Proceedings of European Conference on Planning* (ECP), pp. 130-142.

Darwiche, A. and Marquis, P. (2002), 'A Knowledge Compilation Map', *Journal of Artificial Intelligence Research 17*, 229-264.

Edelkamp, S. and Jabbar, S. (2006), 'Cost-Optimal External Planning', in *Proceedings of the 21st National (American) Conference on Artificial Intelligence* (AAAI), Boston, MA, USA, July 2006. AAAI Press, 821-826.

Fourman, M. (2000), 'Propositional Planning', in *Proceeding of the AIPS Workshop on Propositional Planning*.

Fox, M. and Long, D. (1998), 'The Automatic Inference of State Invariants in TIM', *Journal of Artificial Intelligence Research*, 9, 367-421

Freuder, E.C. (1978), 'Synthesizing constraint expressions', *Communications of the ACM*, 21(11):958-965.

Gerevini, A. and Schubert, L.K. (1998), 'Inferring State Constraints for Domain-Independent Planning', in Proceedings of the *15th National Conference on Artificial Intelligence / 10th Innovative Applications of Artificial Intelligence Conference*, AAAI/IAAI 98, Madison, Wisconsin, USA, pages 905-912, AAAI Press / The MIT Press.

Haslum, P. and Geffner, H. (2000), 'Admissible Heuristics for Optimal Planning', In *Proc. of the 5th Int. Conf. on AI Planning and Scheduling* (AIPS), Colorado, pp. 140-149. AAAI Press.

Helmert, M. (2009), 'Concise finite-domain representations for PDDL planning tasks', *Artificial Intelligence*, 173, 503-535.

Hoffmann, J. and Nebel, B. (2001), 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, 14, 253-302.

Kautz H. and Selman S. (1998), 'BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving', in working notes of the *Workshop on Planning as Combinatorial Search*, held in conjunction with AIPS-98, Pittsburgh, PA.

Koehler, J., Nebel, B., Hoffmann, J. and Dimopoulos, Y. (1997), 'Extending Planning Graphs to an ADL Subset', in *Proceedings of the European Conference on Planning* (ECP-97), Springer LNAI 1348, pages 273-285.

Long, D. and Fox, M. (1999), 'Efficient Implementation of the Plan Graph in STAN', *Journal of Artificial Intelligent Research*, vol.10, pp.87-115.

McAllester, D.A. (1990), 'Truth maintenance', in *Proceedings of the Eighth National Conference on Artificial Intelligence* (AAAI'90), 1109-1116.

Nguyen, X., and *Kambhampati* S. (2001), 'Reviving Partial Order Planning', in *Proceedings of IJCAI-2001.*

Refanidis, I. (2005), 'Stratified heuristic POCL temporal planning based on planning graphs and constraint programming', in *Proceedings of the ICAPS Workshop on Constraint Programming for Planning and Scheduling*, Monterey, California, Beck J.C., Davenport A. and Walsh T. Eds. 66-73.

Refanidis, I. and Sakellariou, I. (2009), 'A Systematic and Complete Algorithm to Compute Higher Order Exclusion Relations', in Proceeding of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS'2009), Greece, pp. 33-42

Rintanen, J. (2000), 'An iterative algorithm for synthesizing invariants', in Proceedings of the *17th National Conference on Artificial Intelligence / 12th Innovative Applications of AI Conference*, pages 806-811, AAAI Press.

Smith D.E. and Weld D.S. (1999), 'Temporal Planning with Mutual Exclusion Reasoning', in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*.

Weld, D.S. and Smith, D.E. (1998), 'Conformant Graphplan', in *Proceedings of AAAI-98*.