A Systematic and Complete Algorithm to Compute Higher Order Exclusion Relations

Ioannis Refanidis and Ilias Sekallariou

University of Macedonia, Dept. of Applied Informatics
Egnatia str. 156, 54006, Thessaloniki, Greece
yrefanid@uom.gr, iliass@uom.gr

Abstract

Planning graphs and binary mutual exclusion (mutex) relations presented a major breakthrough in automated planning technology. They have been exploited in three ways, i.e. extracting optimal parallel plans through search in the space of planning subgraphs; extracting informative heuristics to guide search either in state- or in plan-space; and converting the planning problem into an equivalent satisfiability one. This paper advances the area of planning graphs a step further, by providing for the first time a systematic and complete algorithm to compute exclusion relations (or *nands* as we call them) without a bound on their order. Computing all exclusion relations among the propositions of a planning problem is equivalent to enforcing strong *k*-consistency, with *k* being the number of propositions, i.e., global consistency. The expected consequence is that, if global consistency has been achieved, plans can be extracted in a backtracking free manner. This is proved in the paper for the proposed algorithm and demonstrated experimentally over various small problems from several planning domains. The obvious limitation is that it is practically impossible to compute all higher order exclusion relations for all problems but the smaller ones. So, our work is mainly of theoretical importance; its potential practical importance lies in serving as a starting point for obtaining unexplored relaxations, i.e., new families of heuristics, to be exploited by a graph or state-space based search procedure.

Introduction

Planning graphs were introduced more than a decade ago (Blum and Furst, 1997) and have significantly advanced automated planning technology, by making possible to solve large problem instances without domain specific heuristics, for the first time. At the time of their introduction, they were the dominant technology, as demonstrated by the systems participating in the first international planning competition[1]: three of them, IPP (Koehler et. al, 1997), SGP (Corin et. al. 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 et. al. 1997), was not using planning graphs and exclusion relations.

Even though, in the years that followed, domain independent heuristic state-space planners took the lead in the planning technology, they exploited planning graphs and mutual exclusion relations to extract more informative heuristics and recognize non-achievable propositions. FF (Hoffmann and Nebel, 2001) is a representative case, since its success was mostly based on applying the basic Graphplan algorithm in a relaxed version of the planning problem with no delete effects (and, thus, no mutexes at all. 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 (Weld and Smith, 1998; Smith and Weld 1999).

Both Graphplan and all of its variations are based on the computation of binary exclusion relations. Computing higher order exclusion relations has been considered inefficient, since the extra computational cost is not generally compensated by a faster plan extraction search phase. Although, some approaches to compute higher order relations, usually referred as invariants, have been introduced (presented at the Related Work section), to the best of our knowledge no systematic and complete method has been proposed till now and, consequently, no empirical evidence has been presented to support this claim.

This paper presents a systematic and complete method to compute higher order exclusion relations, or *nands* as we call them, in a propositional setting. The proposed method can be used to compute all exclusion relations of any order, without setting an upper bound on it. It is based on the well known monotonicity property of the exclusion relations (Long and Fox, 1999) and succeeds in minimizing the number of alternative sets of action combinations that need to be taken into account at each level of the planning graph. Computing all exclusion relations among the propositions of a planning problem is equivalent to enforcing strong *k*- consistency, with *k* being the number of

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

propositions, i.e., global consistency (Freuder, 1978). The expected consequence is that, if global consistency has been achieved, plans can be extracted in a backtracking free manner. This is proved in the paper for the proposed algorithm, and is also demonstrated using an implementation in Prolog.

The obvious limitation of the proposed algorithm, also well known attribute of any *k*-consistency enforcing algorithm for large values of *k*, is that it is practically impossible to compute all higher order exclusion relations for any problem but the smaller ones. So, our work is mainly of theoretical importance; we do not intend to outperform state-of-the-art planning systems in terms of time needed to solve problems. However, having such a systematic and complete algorithm to compute higher order exclusion relations might serve as a starting point for obtaining unexplored relaxations, i.e. new heuristics, to be exploited for plan extraction using search either in a graphplan setting or in a state-space or plan-space setting.

The rest of the paper is structured as follows: We start by reviewing the literature. Then we give a quick overview of the Graphplan and, next, we define higher order exclusion relations and identify some key properties of them. The section that follows, which constitutes the core of the paper, presents the details of the method that computes all higher order exclusion relations, the backtracking free plan extraction procedure, and gives some implementation hints that reduce the computational effort. The next section provides experimental evidence by applying the proposed algorithm on several simple planning problems used in the literature. Finally the last section concludes the paper and poses interesting future research directions.

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. To the best of our knowledge, a systematic and complete algorithm to compute these relations neither has been presented nor has been implemented. The common situation was to recognize that incomplete algorithms could be extended to compute all higher order exclusion relations and, consequently, this would result in backtracking free solution extraction. However, due to the expected computational cost, no attempt has been made. As we present in this paper, this extension is not straightforward, whereas there are several opportunities for optimizations; nevertheless, no 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 (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 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 synthesizing invariants reported in (Rintanen 2000) is closest to the work presented in our paper. Rintanen's algorithm is an improvement of the approaches taken by (Gerevini and Schubert 1998) and (Fox and Long 1998). However, the algorithm in (Rintanen 2000) sacrifices completeness wrt the invariants generated in order to maintain polynomial time and is targeted towards the implementation of practical planning systems. Sacrificing completeness is not only because there is a bound on the order of invariants, as well as because 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 the actions at each planning graph level. As we show in our paper, this is not enough: in order to ensure completeness, iterating over the interesting sets of consistent actions at each planning graph level are necessary.

Similar is the work of Haslum and Geffner (2000), in which they 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* the result could be similar to that of our work. However, these estimates are admissible, i.e., even if *m* equals the number of propositions, underestimates might get produced. This is because 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 give empirical evidence.

Under a different perspective, our work could be considered to be closer to symbolic forward exploration techniques such as BDD-based planning algorithms (Bryan 1985; Edelkamp and Reffel 1999). Indeed, both approaches consist of a time-consuming exhaustive constraint generation phase and a backtracking free plan extraction phase. Of course, the data structures used are very different, with the ones employed by the BDD approaches being more compact. However, the approach proposed in our paper is still preliminary, whereas it does not require to select a suitable ordering of the domain variables.

In this paper we adopt a somewhat different perspective: Our aim is not to provide a new planning algorithm with increased performance, or a faster way to compute invariants. Our goal is to compute *all* invariants (higher

order exclusion relations or *nand*s in our terminology), without caring about computational costs. We consider such a result important since (a) it will clearly demonstrate the complexity of the task, (b) it will prove that computing such information is a sufficient condition for generating plans in a backtrack free manner, and (c) it will possibly give rise to unexpected relaxations, i.e., to new families of heuristics.

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 sets of propositions S is not achievable at level i and memoizes this piece of information, there is no guarantee as of whether i is the maximum possible level at which S is non-achievable, or if S remains unachievable at i+1 etc. Furthermore, the way Graphplan memoizes no-good sets of propositions is neither systematic nor complete. So, although there are similarities between our work and the memoization technique of Graphplan, these are only superficial, with the two approaches having fundamental differences.

Background

Let $<\mathcal{P},\mathcal{A},\mathcal{I},\mathcal{G}>$ a planning problem, where $\mathcal{P}$ the set of propositions, $\mathcal{A}$ the set of actions, $\mathcal{I}\subseteq\mathcal{P}$ the initial state and $\mathcal{G}\subseteq\mathcal{P}$ the goals. For every action $a\in\mathcal{A}$, with $Precs(a)$, $Dels(a)$ and $Adds(a)$ we denote its preconditions, delete effects and add effects respectively. In the following, we will use the notation $a=\langle Precs(a), Dels(a), Adds(a)\rangle$ to shortly define an action, e.g. $a=\langle\{p,q\}, \{p\}, \{r,s\}\rangle$.

A planning graph is a graph structure consisting of alternating propositional and action levels, all being grounded. Starting from propositional level 0 comprising the propositions of $\mathcal{I}$, an action a appears at action level i if all of its preconditions appear at proposition level i, without any exclusion relation holding among them. Similarly, a proposition p appears at proposition level i, if an action achieving p appears at action level i-1.

Mutual exclusion relations or *mutexes* over pairs of actions and pairs of propositions can be defined recursively 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, equivalently there is no pair of non-mutexed actions a and b at action level i-1, such that a achieves p and b achieves q. In case there is a single action a applicable at action level i-1 that achieves both p and q, then no mutex between p and q exists at proposition level i.
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.

Note that the basic Graphplan algorithm foresees special actions called *noop* actions. For every proposition p a *noop* action is defined, having p as its single precondition and its single add effect. *Noop* actios are an elegant way to define *mutex* relations between normal actions and propositions.

Higher Order Exclusion Relations

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

Definition 1. An exclusion relation over a set of propositions $P\subseteq\mathcal{P}$ at time t, denoted by $nand_t(P)$ 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 $nand_t(\{p,q,r\})$ 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 example $\{p, r\}$, may hold simultaneously at these levels.

We adopt the term *nand* (a common abbreviation for *not and*) for higher order exclusion relations because "*mutual*" refers to pairs of things, so it does not seem suitable for higher order exclusion relations. On the other hand, *nand* precisely reflects the semantics of exclusion relations, since for a set of propositions P, $nand_t(P)$ means that their conjunction cannot be true at planning graph level t (and before), thus the negation of their conjunction holds.

There are two special cases for $nand_t$ 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. $nand_t(\{p\})$. The meening of this *nand* relation is that proposition p 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 generalization of the exclusion relation. In this paper, both singleton and higher order *nands* are treated uniformly.

The following Lemmas emanate naturally from Definition 1:

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

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

From an implementation point of view, Lemma 1 is important since it imposes that only minimal *nand* sets need to be retained in memory. Furthermore, Lemma 2 imposes that only latest *nands* over a set of propositions should be retained. From now on, wherever we refer to *nand* relations over sets of propositions, we will refer to minimal sets of propositions.

Similar definitions hold for *nand* relations over actions:

Definition 2. An exclusion relation over a set of actions $A \subseteq \mathcal{A}$ at time t, denoted by $nand_t(A)$, means that there is no plan of t parallel steps or less, where all actions in A can be executed simultaneously at any step of the plan.

The following lemmas emanate naturally:

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

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

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

Proposition 1. Relation $nand_t(A)$ holds over a set of actions $A \subseteq \mathcal{A}$ at time t, iff:

- A comprises two actions, a_1 and a_2, that are eternally mutexed, or
- there is a set of propositions P such that $nand_t(P)$ holds and, furthermore:
  1. $P \subseteq \bigcup_i Precs(a_i)$
  2. $\forall a_i \in A, Precs(a_i) \cap P \neq \varnothing$
  3. $\forall a_i \in A, (Precs(a_i) \cap P) - \bigcup_{j \neq i} Precs(a_j) \neq \varnothing$

Proof: The first case is trivial. As for the second case, 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. ■

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

Definition 3. A set of non-*nanded* actions $A \subseteq \mathcal{A}$ applicable at time t is *nanded* with a set of non-*nanded* propositions $S \subseteq \mathcal{P}$ at time t, denoted with $nand_t(A,S)$, 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-*nanded* actions $A \neq \varnothing$ may be *nanded* with a set of non-*nanded* propositions $S \neq \varnothing$ at time t, if there is a relation $nand_t(P)$, such that:

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

Proof: In order to execute all actions of A at time t, all propositions of P-S must hold (condition 2). So, all propositions of S cannot hold simultaneously, because in that case the $nand_t(P)$ 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 *nanded* with a set of non-*nanded* propositions $P \subseteq \mathcal{P}$ at time t, if there is no way to execute a at t, while propositions of P being true at the start time of the execution of a, i.e. at t.

Proposition 3. For each $nand_t(a,P)$ relation where a is an action of a non-*nanded* set of actions A at time t, $nand_t(A,S)$ also holds, where:

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

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

$$P' = \bigcup_{a_i \in A, a_i \neq a} \mathrm{Pr}ecs(a_i)$$

with $P \cap P' \neq \varnothing$. In that case, in order for actions A to be simultaneously applicable, propositions in P' should all be true simultaneously at t. So, at least one of the propositions in $S=P-P'$ must be false at t. Note that S cannot be empty, otherwise actions of A are *nanded* (Proposition 2). ■

Proposition 4: A set of propositions P is *nanded* at time t, if there is no set A of non-*nanded* actions at time t-1 that achieve a subset S of P, with the remaining propositions P-S neither being *nanded* 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 and achieving S, and P-S is not *nanded* with A at t-1, whereas no action $a \in A$ deletes any of the propositions in P-S, then is it straightforward that all propositions in P can hold at t. Otherwise, they cannot. ■

Finally, let's define the notion of minimal *nand* relations in a more formal way:

Definition 4. Any relation $nand_t(A,P)$ is minimal, if there is no other relation $nand_{t'}(A',P')$, 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 $t'>t$.

For any *nand* relation that it is not minimal, we can say that it is subsumed by another *nand* relation.

Computing *nands*

Before going into the details of the *nands* computing algorithm, we need some more definitions and propositions that concern their propagation. We assume that only minimal *nands* are retained.

The following Proposition generalizes the well known monotonicity property of mutexes in Graphplan, i.e. mutexes monotonically decrease. In this case, *nand* relations only relax, i.e. they increase their order or disappear.

Proposition 5. For any *nand* 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 *nanded* at levet t-1 of the planning graph.

Proof: It is not possible for a set of propositions to be non-*nanded* at some level t-1 of the planning graph, and being *nanded* 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 not containing any action. So, either all propositions or some subset of them (even stronger condition) were *nanded* at the previous level. ■

According to Proposition 5, new *nand* relations are created only when old *nand* relations break. Note that the initial state of a planning problem involves *nand* relations of order 1 for 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 *nand* relations that involve this proposition might arise. Similarly, each time a higher-order *nand* relation breaks at some level, new *nand* relations of an even higher order than the broken one might arise, each one of them containing all the propositions of the broken relation.

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

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

Proof: We assumed that *nand* relations are minimal, so any subset of the involved entities can hold simultaneously. So, 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, the set P-S is not *nanded* with a at t (first condition), so 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 there is no $nand_{t+1}(P)$. ■

Proposition 6 is very useful, because it says that a single action is needed to break a *nand*, and this action just needs to achieve a single proposition of the *nand*, without neither deleting any one of the remaining propositions, nor being *nanded* with them.

Definition 5. For a set of non-*nanded* actions A applicable simultaneously at level t, we denote with $negate_{t+1}(A)$ the minimal set of sets of propositions that cannot hold simultaneously at time t+1, provided that no action outside A has been applied at t. In this case, "minimal" refers to a set $\mathbf{S}$ 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. S, for any relation $nand_t(A,S)$, such that $S \bigcap \left(\bigcup_{a \in A} Adds(a) \right) = \varnothing$

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 S that are *nanded* with the set of actions A at t, should be added to $negate_{t+1}(A)$, provided that no action of A achieves any of them (in which case S can be true at t+1, because any subset of S could be true at t). We do not consider the case of any action of A deleting any of the propositions of S, since we assume that all actions' delete effects are also their preconditions, so none of them should appear at any set S, according to Proposition 2. ■

Example 1. Suppose that there are two non-*nanded* 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 *nands* over propositions hold at t: $nand_t(\{p,s,v\})$ and $nand_t(s,r,y)$. According to Proposition 2, $nand_t(\{a\}, \{s,v\})$ holds for a, so $negate_{t+1}(\{a\}) = \{\{p\}, \{s,v\}\}$. Similarly, $nand_t(\{b\}, \{r,y\})$ and $nand_t(\{b\}, \{p,v\})$ hold for b, so $negate_{t+1}(\{b\}) = \{\{r,y\}, \{p,v\}\}$. Finally, $nand_t(\{a,b\}, \{v\})$ and $nand_t(\{a,b\}, \{r,y\})$ 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 following two Propositions constitute the heart of the *nand* computing algorithm. *Interesting* sets of actions are defined in Definition 7 and explained further in the last subsection.

Proposition 8. For any relation $nand_t(P)$ that breaks at level t+1 according to Proposition 6, for all *interesting* sets of non-*nanded* actions A_i at level t that can break this *nand* relation, create new *nand* relations at t+1 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) *nand* relations subsumed by existing (new) relations should be ignored (retracted).

Proof: Set **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-*nanded* set of actions A_i that break $nand_t(P)$. 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, $nand_t(P)$ 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 *nand* relations for all sets of $P \times \mathbf{N}$ should be created. Minimality requirement imposes to keep only those *nand* relations (new or existing) that are not subsumed by others. ■

Example 2. Continuing example 1, suppose that u is a newly achieved proposition, i.e. $nand_t(\{u\})$ breaks at t+1. Suppose that 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 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, **N** becomes

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

By taking the product $P \times \mathbf{N}$, new *nands* 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 $nand_t(\{u,z\})$ 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 *nand* 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 $nand_t(P_i)$ that break at level t+1 according to Proposition 6, for all interesting sets of non-*nanded* actions A_i at level t that can break simultaneously all these *nand* relations, create new *nand* 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) *nand* relations subsumed by existing (new) relations should be ignored (retracted).

Proof: Proposition 9 generalizes Proposition 8 in that it considers simultaneously breaking *nand* relations. The only difference is that a set of *nand* 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, each one of them 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 $nand(P_i)$ 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 *nand* 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 $nand_t(P_i)$ relations, then a *nand* relation over P exists at t+1.

Definition 6. 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 any $R \in \mathbf{R}$, there is some $S \in \mathbf{S}$ 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 in S. The "relaxed" relation imposes a partial ordering over sets of actions, through their $negate_t$ sets.

Definition 7. A set of non-*nanded* actions A at level t that break a set of $nand_t(P_i)$ relations is interesting iff there is no other set of non-*nanded* actions B at level t that break the same set of *nand* relations such that $negate_{t+1}(\{A\}) \prec negate_{t+1}(\{B\})$.

Proposition 10. Non-interesting sets of actions among those that break the same *nand* relation can be ignored.

Proof: Suppose two sets of actions A and B, with $\mathbf{R}=negate_{t+1}(A)$ and $\mathbf{S}=negate_{t+1}(B)$, such that $\mathbf{S} \prec \mathbf{R}$. Any entry $R \cup S$ of the Cartesian product $\mathbf{R} \times \mathbf{S}$, with $R \in \mathbf{R}$ and $S \in \mathbf{S}$, would be subsumed by the entry $R \cup S_R$=R, where $S_R \in \mathbf{S}$ such that $S_R \subseteq R$, since R$\subseteq R \cup S$ and we are interested only in minimal *nand* relations. ■

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

So, we can now present the algorithm that computes all *nand* relations. The algorithm focuses on *nand* relations over sets of propositions; however *nand* 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 *nand* relations over propositions.

1. Create $nand_\infty(\{p\})$ relations for all propositions $p \notin \mathcal{I}$. Let t=0
2. While there are *nand* relations over $\mathcal{G}$ and the planning graph has not yet leveled off, repeat the following:
   a. Find the broken $nand_\infty$ relations at level t+1, according to Proposition 6. Each one of them is replaced by a $nand_t$ relation.

b. For each combination of broken $nand_\infty$ relations, apply Proposition 9 to create new $nand_\infty$ relations.
c. Let t=t+1.

Step 1 concerns initialization, by recording all propositions that are not included in the initial state $\mathcal{I}$ as unary *nands*. 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 *nand* relation over any subset of $\mathcal{G}$. Leveling off occurs if during the last iteration no *nand* relation has broken.

Step 2a concerns breaking *nand* relations. According to Proposition 6, a single action is needed to break a *nand* relation, 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 *nands*.

Finally, step 2b generates new *nand* relations. For each set of broken *nands*, all interesting sets of actions that can break them should be considered. 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 *nand* relations progressively, i.e. new relations are always $nand_\infty$, until they break. So, if the algorithm terminates before the planning graph levels off, there might remain some $nand_\infty$ relations, although these relations would possibly break at subsequent levels.

Extracting a plan

If we are interested in finding a plan, then as soon as no *nand* exists among $\mathcal{G}$, we can proceed with the plan extraction phase. This can be done in a backtracking free way by the following algorithm, which results in optimal parallel plans.

Algorithm 2. Extracting a plan.

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

Algorithm 2 is a generalization of the plan extraction phase of Graphplan. Indeed, at each level a set of non-*nanded* actions that break the related *nand* relations of the previous level is selected (step 2c), with the requirement that no new *nand* relations over the current set of open goals 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.

Proposition 11. Provided that all *nand* relations have been computed up to level t, for any set of not *nanded* propositions G at planning graph level t, there is always a set of non-*nanded* actions A_i at t-1 such that that no subset of G appears in $negate_t(A_i)$. Furthermore, the set G' comprising the union of the preconditions of the actions in A_i and the subset of propositions of G that are not achieved by any action of A_i is *nand*-free at level t-1.

Proof: Suppose that there is no *nand* 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.

So, suppose that *Broken* is the non-empty set of subsets of G that are *nanded* at level t-1. Let $G_1 \subseteq G$ the subset of G comprising the union of all propositions appearing in *nand* relations of *Broken*. So, we need to find a set of actions A_i that can break all *nand* relations in *Broken* simultaneously. This set of actions should exist, since otherwise a *nand* 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 other, a new *nand* 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 *nand* exists over G at t.

Since no subset of G appears in $negate_t(A_i)$, no *nand* 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 *nand* relation between some preconditions of Ai 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 *nand* relation exists between sets of not achieved propositions of G, since we assumed that A_i breaks all *nand* relations between propositions of G, so A_i should achieve at least one proposition for each broken *nand* relation. So, there is no *nand* 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-*nanded* propositions at level t, we can result with a new set of non-*nanded* propositions at level t-1. ■

Lemma 8. Provided that all *nand* relations have been computed up to the level where $\mathcal{G}$ appears without any *nand* over its propositions, Algorithm 2 extracts a plan in a backtracking-free way.

It is expected that for large size problems, it will be practically impossible to compute all *nand* relations. In that case, one can set a limit on the order of the *nand* relations computed by Algorithm 1, and ignore any relation above this limit. In such situations, the property of Algorithm 2 for backtracking-free plan extraction vanishes. In that case, a common Graphplan-like search phase could be adopted, taking into account the higher order *nand* relations whenever sets of actions are selected at the various planning graph levels.

Implementation Details

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

Concerning the sets of broken *nands*, we can adopt a progressive approach, starting from small sets of broken *nands* and going to larger ones. If for a set of broken *nands* there is no way to break all of them simultaneously (case of Lemma 6), there is no reason to examine any superset of them. 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 *nands*, one can start with minimal sets consisting from at most one action for each broken *nand* of the set (it might happen that the same action breaks more than one *nands* 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-*nanded* with A and achieves something of $negate_{t+1}(A)$ (obviously not the propositions deleted by A's actions), such that $negate_{t+1}(A) \prec negate_{t+1}(A\text{-}a)$ does not hold. 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 should be performed, before proceeding with computing the Cartesian product.

Finally, it seems that of great complexity is the computation of the Cartesian product. To be as efficient as possible, one should 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 greatly accelerates the overall process. This elimination is greatly facilitated by keeping the subproducts sorted based on the cardinality of their elements.

Experimentation

We implemented the proposed algorithms (computing *nands* and plan extraction) in Prolog. Particularly, we used the ECLiPSe Constraint Logic Programming Platform for running our experiments, although we haven't exploited any of the constraint programming features of the language. In our experiments we were not interested in measuring the solution time; this is an issue of future research, using a more optimized version of our code. However, we can report that in all experiments mentioned in this Section, solving time varied from a few seconds to a few minutes, with the contribution of the plan extraction phase to the total time being negligible[2]. All problems have been solved optimally in a backtracking free manner.

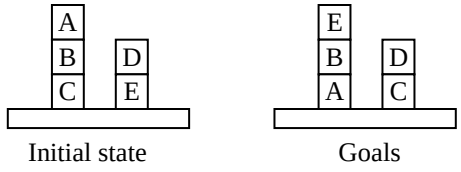

Figure 1. A simple blocks world problem with five blocks.

First we present an example from the blocks world problem and the version with the three operators (Figure 1). Table 2 summarizes the *nand* relations found on this problem up to level 3, where a solution was found. The *cnand* refers to broken *nands*, whereas *enand* refers to the eternal *nand* relations.

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

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

If we continue running Algorithm 1 until the graph levels off, the following *nands* can be found:

[2] The code and the problem files used for the experimentation are available through contacting the authors.

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

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

Examples of eternal *nands* of higher order are:

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

whereas exampled of broken *nands* are the following:

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

We claim than broken *nand* relations are more interesting for plan extraction, since they are related to the specific problem instance. Indeed, eternal *nand* relations could be considered domain knowledge and be provided manually, whereas broken *nand* relations should always be computed from scratch for each problem.

The second problem is *strips-gripper-x-1* from the gripper domain of IPC-1. There are 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 *nands* after the graph has leveled off.

Order	*cnands*	*enands*	Total
2	12	45	57
3	52	0	52
4	52	0	52
Totals	116	45	161

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

Examples of *nands* found are the following:

$nand_{4}(\{at(ball4,roomb),carry(ball2,right),$
$carry(ball3,left)\})$

$nand_{5}(\{at\_robby(roomb),at(ball4,roomb),$
$carry(ball2, right), carry(ball3, left)\})$

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 *nands* computed for the various orders after the planning graph leveled off.

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

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

Examples of *nands* found are the following:

$nand_{9}(\{closed(boot), in(jack,boot), in(wheel1,boot),$
$in(wrench,boot), loose(nuts,hub)\})$

$nand_{8}(\{in(jack,boot), in(wheel1,boot),$
$in(wrench,boot), loose(nuts, hub)\})$

$nand_{11}(\{closed(boot), in(wrench, boot),$
$on(wheel2, hub), tight(nuts, hub)\})$

$nand_{4}(\{closed(boot), in(wrench, boot), loose(nuts, hub)\})$

$nand_{2}(\{free(hub), on\_ground(hub)\})$

$nand_{2}(\{closed(boot), have(wrench)\})$

Although the set of examples is neither extensive nor diverse enough to extract general conclusions, we will risk stating a few of them based on, apart from the actual results, our intuition. In all three experiments, the number of *cnands* exceeds the number of *enands*, when the planning graph levels off. Although this is not a general rule, it is expected that this is the most common case. Table 1 is not an exception, since it does not concern a leveled-off graph.

Another interesting observation concerns the numbers of total exclusion relations (sum of *cnand*s and *enand*s) when the planning graphs level off. In all three cases, the total number decreases with the order. Again, this is not a general rule but it seems that it is a situation common in many of the benchmark domains used in the planning literature. Our intuition suggest that breaking this rule, i.e., having more higher order exclusion relations than lower order ones, would indicate more complex problems.

However, this analysis was not the goal of this paper; indeed, we were interested only in providing a complete and systematic algorithm to compute all higher order exclusion relations, with the examples being selected only for demonstration purposes. In a subsequent paper we will perform such an analysis based on experimental results from more domains and from several problems of various sizes from each domain.

Conclusions and Future Work

In this paper we presented a systematic method to compute all higher order exclusion relations between the propositions of a planning problem. We denote each such relation over a set of propositions P with $nand_t(P)$, since the conjunction of propositions of P cannot be true at any planning graph level $t' \leq t$. We also defined *nand* relations over sets of actions, as well as over sets of actions and sets of propositions, and we proved several interesting properties for them.

Having computed all the *nand* relations for a problem, we presented an algorithm that extracts optimal parallel plans, and we proved that this algorithm is backtracking free. Finally, we implemented our algorithm and tested it on several planning problems from the literature. For each such problem we observed its behavior in terms of number of *nand* relations computed and their distribution over the various orders. As expected, all these problems were solved optimally and in a backtracking free manner.

Our next step concerns trying to optimize our code in terms of efficiency of the *nand* computing algorithm, using some form of intermediate results caching and by reiplementing it in a more efficient platform. Next, we plan to experiment with several relaxations as, e.g., setting limits on the order of the computed *nands* and analyzing the overall tradeoff between the time spent for *nand* computation and for plan extraction in different domains. The same analysis can be performed for state-space heuristic planners, using the *nand* relations for heuristic guidance; for graphplan-based planners when optimal parallel plans are requires; and for satisfiability planners, where *nand* relations constitute higher order constraints.

Another line of research would be to examine how the algorithm that computes the *nands* can take advantage of precomputed *nand* relations, that could be provided either by hand or could be obtained automatically through domain analysis. Finally, analyzing the order and the number of *nand* relations for various problems of the same domain and how they depend on the size of the problem might be an interesting indicator of the difficulty to solve problems from this domain.

References

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.

Edelkamp, S. and Reffel, F. 1999. Deterministic State Space Planning with BDDs. In *Proc. of the 5th European Conference on Planning (ECP)*, Durham, pages 381-382.

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.

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. 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.

Nguyen, X., and *Kambhampati* S., 2001. Reviving Partial Order Planning. In *Proceedings of IJCAI-2001.*

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*.