Lifted Successor Generation by Maximum Clique Enumeration

Simon Ståhlberg[a;*]

[a]Linköping University, Sweden

Abstract. Classical planning instances are often represented using first-order logic; however, the initial step for most classical planners is to transform the given instance into a propositional representation. For example, action schemas are converted into ground actions, aiming to generate as few ground actions as possible without eliminating any viable solutions to the problem. This step can become a bottleneck in some domains due to the exponential blowup caused by the grounding process. A recent approach to alleviate this issue involves using the *lifted* (first-order) representation of the instance and generating all applicable ground actions on-the-fly during the search for each expanded state. In this paper, we propose a method that addresses this problem by enumerating all maximum cliques of a graph encoding the state and the action schema's preconditions. We compare our method with state-of-the-art across 47 domains, showcasing improved performance in 23 domains. In some cases, simply changing the maximum clique enumeration algorithm results in a significant speedup compared to the state-of-the-art.

1 Introduction

Classical planning instances often consist of a *domain*-specific part and a *problem*-specific part [18]. The domain provides problem-independent structure, such as a set of *predicate symbols* and *action schemas*, both expressed in first-order logic. In contrast, the problem consists of a set of *objects*, *initial state*, and *goal*, all described in propositional logic. This format enables efficient representation of the *state-transition graph* of a *planning instance* (domain and problem); for instance, each action schema implicitly defines up to n^m ground actions, where n is the number of ground atoms and m is the action schema's arity. However, in practice, significantly fewer than n^m ground actions are applicable in states reachable from the initial state; the *grounding problem* involves enumerating all *relevant* ground actions given an action schema.

The STRIPS [13] and SAS$^+$ [1] formalisms are commonly used for classical planning, and both are defined over a propositional representation. As a result, many planning techniques, such as heuristic functions, are only defined for propositional representations. In practice, most state-of-the-art planners ground the first-order representation into a propositional representation as a preprocessing step (e.g., [3, 26, 21, 38]). An exception is the *Powerlifted planner* [8], which operates with the *lifted representation*: given a state, the implemented search algorithms enumerate all applicable ground actions from an action schema on-the-fly, and heuristic functions do not have access to any ground actions. This approach benefits from the ability to tackle *hard-to-ground* problems, i.e., problems where the preprocessing step will not finish within the given (reasonable) resource limits. As a result, Powerlifted is able to outperform state-of-the-art ground planners on hard-to-ground domains in practice [8].

Lifted successor generation is not only of interest to lifted planners with heuristics (e.g., [33, 27, 41]), but also to policies that only need to consider the current state and all immediate successors (e.g., [29, 2, 15, 34, 35, 36, 9, 10, 11]). Given a proper policy, plans can be found by first enumerating all successor states, and then greedily follow the transition recommended by the policy. In this case, grounding action schemas beforehand is unnecessary, and sometimes a lifted successor generator is needed to apply policies on large instances.

In this paper, we address the problem of enumerating all applicable ground actions for a specific state, i.e., the *lifted successor generation* step. Our proposed method consists of two steps: (1) constructing a *substitution consistency graph* based on the action schema's parameters and preconditions, the problem's objects, and the given state; and (2) enumerating all *maximum cliques* present in the graph. The graph is designed in such a way that each maximum clique corresponds to a potentially applicable ground action in the given state. If all atoms in the action's precondition are at most binary, every maximum clique will result in an applicable ground action. However, if any atom has an arity greater than 2, our method will *over-approximate* the set of applicable ground actions, necessitating verification of each resulting ground action's actual applicability. In our experiments, this over-approximation does not substantially reduce overall performance. We report empirical results for a preliminary implementation and compare our method to the lifted successor generator implemented in Powerlifted [8], which is based on techniques from database theory. To our knowledge, this method represents state-of-the-art. However, it is worth noting that there exists a lifted successor generator based on enumerating all solutions to CSPs as well [14]. We compare the methods on two benchmarks: 4 299 instances over 47 domains from IPC, and 324 instances over 5 hard-to-ground domains [27]. The experiments demonstrate that our approach surpasses the state-of-the-art in 22 IPC domains while also being complementary due to their distinct performance characteristics. This is expected, given that the problem is *NP-hard*, rendering it improbable for a single method to consistently outperform all others. Therefore, it is crucial to offer multiple successor generators and choose the best one for the task.

The remainder of the paper is organized as follows. We review related work, classical planning, and describe the substitution consistency graph, how ground actions are enumerated in this graph, and present soundness and completeness proofs for the proposed method. This is followed by experiments, a discussion and a summary.

* Corresponding Author. Email: simon.stahlberg@liu.se.

2 Related Work

The ground representation of an instance can be exponentially larger than its lifted counterpart, posing a significant challenge. Ground planners address this issue by enumerating only the ground actions applicable in states reachable from the initial state. Efficient preprocessing techniques, such as reachability analysis, are employed to obtain an *over-approximation* of the desired ground actions [21, 22, 32]. However, due to the exponential blowup, this approach may not always be feasible. The requirement of enumerating all ground actions that are applicable in some reachable state can be made more strict. Some methods attempt to learn how to ground with the objective of enumerating only those actions that participate in optimal solutions (e.g., [19]). Although it is challenging to provide guarantees, the goal is to preserve solvability while sacrificing completeness.

In this work, we address this issue by employing lifted successor generators. To the best of our knowledge, the only other lifted successor generators are based on database techniques [8] and CSPs [14]. The first method is founded on the observation that a state and the precondition of an action schema can be regarded as a database and a conjunctive query, respectively. Although the complexity of conjunctive query evaluation is NP-hard [5], acyclic queries can be evaluated in time polynomial with respect to input and output size [42]. This method has been implemented in the *Powerlifted* planner [8], and we incorporate our approach within this planner as well.

Conceptually, our method resembles the *micro-structures* approach for *constraint satisfaction problems* (CSPs) proposed by [23]. A micro-structure is an undirected graph wherein each vertex represents a substitution, and edges indicate *compatible* substitutions concerning the constraints of the CSP. Solutions for the CSP can be obtained by identifying maximal cliques within the micro-structure of the CSP. Although discovering all maximal cliques is an NP-hard task [25], some classes of undirected graphs, such as triangulated graphs, render the problem tractable [17].

3 Preliminaries

A classical planning instance is defined as a tuple $\langle \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. The *domain* and the *problem* components of the instance are respectively denoted by $\langle \mathcal{P}, \mathcal{A} \rangle$ and $\langle \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where:

- $\mathcal{P}$ represents a set of *predicate symbols*, where each symbol $P \in \mathcal{P}$ has an associated *arity* n. An *atom* is denoted as $P(x_1, \ldots, x_n)$; if no term x_i is a variable, then it is a *ground atom*.
- $\mathcal{A}$ is a set of *action schemas*. Each action schema $A \in \mathcal{A}$ comprises a set of variables *params*(A) and two sets of atom literals, *pre*(A) and *eff*(A), over *params*(A). The *precondition* and the *effect* of A are represented by the former and latter sets, respectively. The *arity* of A is defined as $|$*params*$(A)|$.
- $\mathcal{O}$ denotes a set of *objects* (constants).
- $\mathcal{I}$ is the *initial state* and $\mathcal{G}$ is the *goal*. Both are *states*, which are sets of ground atoms over $\mathcal{P}$ and $\mathcal{O}$. A *state* $\mathcal{S}$ is considered a *goal state* if and only if $\mathcal{G} \subseteq \mathcal{S}$.

A *ground action* $a = [A/X]$ of an action schema A is a complete substitution X of all variables in *params*(A) with objects in $\mathcal{O}$. The precondition *pre*$(a) = [$*pre*$(A)/X]$ and the effect *eff*$(a) = [$*eff*$(A)/X]$ of a are sets of ground atom literals where each free variable has been substituted with an object in X.

We use $\mathit{pre}^+(\cdot)$, $\mathit{pre}^-(\cdot)$, $\mathit{eff}^+(\cdot)$, and $\mathit{eff}^-(\cdot)$ to represent the *atoms* of positive and negative literals in the precondition and effect, respectively (note that these sets contain atoms, not literals). Given the polarity (sign) s and the atom p of some literal l, we state that l holds in a state $\mathcal{S}$ if and only if $s = (p \in S)$. A ground action a is considered *applicable* in a state $\mathcal{S}$ if and only if all literals in *pre*(a) hold in $\mathcal{S}$ (i.e., $\mathit{pre}^+(a) \subseteq \mathcal{S}$ and $\mathit{pre}^-(a) \cap \mathcal{S} = \emptyset$). The result of *applying* a in $\mathcal{S}$ is the *successor state* $\mathcal{S}' = (\mathcal{S} \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$. A sequence of ground actions $a_1, \ldots, a_n$ is applicable in a state $\mathcal{S}$ if each a_i is applicable in the state resulting from applying $a_1, \ldots, a_{i-1}$ in $\mathcal{S}$ in the given order. If applying a sequence of ground actions from $\mathcal{I}$ results in a state $\mathcal{S}$ such that $\mathcal{G} \subseteq \mathcal{S}$, then the sequence is a *solution*. We define that an atom $p_1(x_1, \ldots, x_n)$ *matches* an atom $p_2(y_1, \ldots, y_n)$ if and only if $p_1 = p_2$ and for every i, $1 \leq i \leq n$, x_i or y_i is free, or $x_i = y_i$.

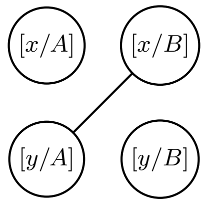

Figure 1: The substitution consistency graph for Example 1. The only maximum clique is $\{[x/B], [y/A]\}$.

4 Substitution Consistency Graph

The first step of our method is to encode the precondition of an action schema A and the ground atoms of a state $\mathcal{S}$ into an undirected graph $G_{A,\mathcal{S}} = \langle V, E \rangle$. We refer to this graph as the *substitution consistency graph*. The vertices $V = \{[x/o] : x \in \mathit{params}(A), o \in \mathcal{O}\}$ represent all possible substitutions of the free variables, and the edges $E = V^2 \setminus I$ denote substitutions that are *consistent* with one another, where I denotes pairwise *inconsistent* substitutions. Specifically, $I = I^{\neq} \cup I^+ \cup I^-$:

- $I^{\neq} = \{\{[x/o_1], [x/o_2]\} : x \in \mathit{params}(A), o_1, o_2 \in \mathcal{O}\}$, i.e., edges that assign different values to the same variable;
- $I^+ = \{\{v_1, v_2\} : v_1, v_2 \in V, \exists p_1 \in \mathit{pre}^+(A), \forall p_2 \in \mathcal{S}, [p_1/\{v_1, v_2\}]$ *does not match* $p_2\}$, i.e., partial substitutions that do not match any ground atom in the state for some positive literal in the precondition;
- $I^- = \{\{v_1, v_2\} : v_1, v_2 \in V, \exists p \in \mathit{pre}^-(A), [p/\{v_1, v_2\}] \in \mathcal{S}\}$, i.e., complete substitutions that result in ground atoms that are in the state for negative literals in the precondition.

The graph $G_{A,\mathcal{S}}$ is k-partite, where k is the arity of the action schema, since $I^{\neq}$ disallows edges between substitutions of the same variable, thus the vertices can be partitioned into k independent sets. Note that, contrary to I^+, not all negative literals are taken into account in I^-, only those with arity at most 2. This graph does not, in general, encode the precondition and the state perfectly, but it does allow for an *over-approximation* of all applicable ground actions with respect to A and $\mathcal{S}$. This is why we focus on *excluding* edges (rather than including), and we will take a closer look at over-approximation in the experiments.

Example 1. *The domain* Blocksworld *is comprised of: the predicates On/2, On-Table/1, Clear/1, Hand-Empty/0, Holding/1; and the actions Pick-Up/1, Put-Down/1, Stack/2, Unstack/2. The problem consists of two objects A and B, and the state $\mathcal{S}$ contains the ground atoms: Clear(B), On(B, A), Hand-Empty(), On-Table(A). The only ground action applicable in $\mathcal{S}$ is Unstack(B, A), where* *params*$(Unstack) = \{x, y\}$ *and pre*$(Unstack) = \{On(x, y),$ *Clear*(x)*, Hand-Empty*$()\}$.

Figure 1 illustrates the substitution consistency graph. There is a single edge in the graph between $[x/B]$ and $[y/A]$, which denotes that these substitutions are consistent with one another. In this case, the set $I = I^{\neq} \cup I^{+} \cup I^{-}$ is:

- $I^{\neq} = \{\{[x/A], [x/B]\}, \{[y/A], [y/B]\}\}$
- $I^{+} = \{\{[x/A], [y/A]\}, \{[x/B], [y/B]\}, \{[x/A], [y/B]\}\}$
- $I^{-} = \{\}$

The condition used in I^{+} takes into account the atom $On(x, y) \in$ pre$(Unstack)$ and the ground atom $On(B, A) \in \mathcal{S}$, and determines that these substitutions cannot satisfy the precondition. The nullary and unary atoms in the precondition are also taken into account: consider the case when Hand-Empty$() \notin \mathcal{S}$ and the substitution $s = \{[x/B], [y/A]\}$, then $s \in I^{+}$ since [Hand-Empty()/s] does not match any ground atom in $\mathcal{S}$.

A naive implementation for checking if an edge is consistent with the atoms in the state can be computationally expensive, as it requires iterating over all atoms in the state. To speed up this process, we create a bitset for each predicate symbol, where each position in the bitset represents substituting two parameters with actual objects. A bit is set to true only if there is a ground atom in the state that matches the atom generated by the substitution; otherwise, it is set to false. In other words, given a substitution, we can determine if a ground atom exists in the state that matches the atom generated by the substitution. This data structure is efficiently constructed by initializing all bits to false. Next, we iterate over each atom in the state and all possible pair combinations of its arguments, setting the corresponding bit to true. Now, given a substitution and a literal, we can identify the positions of the predicate it substitutes and the objects it replaces. This provides a position in a bitset, and the substitution is consistent if the corresponding bit is set to true.

5 Ground Action Enumeration

The second and final step of our method involves finding all maximum cliques in $G_{A,\mathcal{S}}$ for a given action schema A and state $\mathcal{S}$. A *clique* in an undirected graph $G = \langle V, E\rangle$ is defined as a subset of vertices $C \subseteq V$ where $C^2 \subseteq E$. That is, all vertices within the clique are pairwise connected. In our specific context, a clique corresponds to a substitution of *params*(A), with each single substitution is consistent with all others. A *maximum* clique represents a complete substitution of the parameters. Additionally, if the arity of each literal in *pre*(A) is no greater than 2 and C is a maximum clique in $G_{A,\mathcal{S}}$, then $[A/C]$ is applicable in the given state. In this scenario, we can enumerate all applicable ground actions by finding all maximum cliques. By design, we know that the size of *maximum* cliques in $G_{A,\mathcal{S}}$, if one exists, is $k = |$*params*$(A)|$. However, determining the existence of such cliques is an NP-complete problem [25]. We also observe that if $k = 2$, then each edge constitutes a maximum clique, yielding a ground action.

We consider two algorithms for enumerating all maximum cliques in $G_{A,\mathcal{S}}$. The first algorithm is Bron-Kerbosch [4], an efficient recursive backtracking method for identifying maximal cliques in undirected graphs. It operates by maintaining three dynamic vertex sets: the current potential clique R, the set of candidate vertices P that can potentially be added to the current clique, and the set of already processed vertices X. It iterates through the vertices in the candidate set P, adding each to the potential clique R, and recursively calls itself with the updated sets. The algorithm backtracks when the candidate set P and the processed set X become empty, indicating that a maximal clique has been found or no more cliques can be extended from the current state. The algorithm can be optimized by introducing pivoting, which allows us to reduce the number of recursive calls by considering only candidates not in the neighborhood of the pivot. In our implementation, a pivot is selected arbitrarily from X if $X \neq \emptyset$, otherwise from P. Other common pivot selection strategies are to select a pivot v that maximizes $|N(v)|$, where $N(v)$ is the neighborhood of v, and to minimize $|P \setminus N(v)|$. However, we did not find that the extra computational effort involved with these pivot strategies was worthwhile. Lastly, we are only interested in maximum cliques of size k, so in our implementation, we backtrack early if $|R \cup P| < k$, which indicates that the current branch cannot lead to any maximum cliques.

The second algorithm, k-Clique k-Partite, takes advantage of the fact that our graph is k-partite, and our goal is to identify cliques with a size of exactly k. There are algorithms specifically tailored for these kinds of graphs [20, 30]. To find a clique of size k, we must choose one vertex from each partition. We can achieve this efficiently by examining the partitions in an arbitrary sequence, systematically selecting a vertex from the current partition that is compatible with the current potential clique, and then proceeding to the next partition. By arranging the partitions in this way, we implicitly disregard all vertices in earlier partitions as potential candidates.

In our implementation of the Bron-Kerbosch algorithm, sets are represented by sorted vectors of integers, with each integer signifying a vertex (a single substitution). We experimented with representing sets as bitsets but found that this approach was marginally slower in practice. The graph is represented using adjacency lists, where each vertex list is sorted to ensure efficient set operations. Additionally, we employ degeneracy ordering [12], in which the vertices are sorted in ascending order by their degree, and we process the vertices in this order while executing the Bron-Kerbosch algorithm. This method helps reduce the search space and, consequently, speeds up the algorithm. Overall, our implementation is standard, and there may be potential for further optimization.

Our implementation of k-Clique k-Partite follows the presentation in [30], where representing sets as bitsets is essential for achieving low asymptotic complexity. In this version, the graph is represented as an adjacency matrix, where each row is a bitset to make the set operations within the algorithm efficient. Overall, our implementation here is also very standard.

6 Soundness

We now show that the maximum cliques we find in the substitution consistency graph yield applicable ground actions when the action schema has an arity of at least 2 and all literals in the precondition have an arity of at most 2. Our method incorrectly handles unary action schemas since single substitutions are maximum cliques without necessarily resulting in applicable ground actions; however, this case can easily be fixed by simply removing such vertices from the graph.

Lemma 1. *Let A be an action schema with arity $k \geq 2$, let $\mathcal{S}$ be a state, and let C be a clique of size k in $G_{A,\mathcal{S}}$. Then, $[A/C]$ is a ground action.*

Proof. The subset $I^{\neq}$ of I includes all edges that replace the same variable with different values. Suppose a free variable x appears multiple times in C; then all pairs in C where x appears represent edges that must belong to $I^{\neq}$. Consequently, each variable must appear at most once in C. Since the cardinality of C is equal to k, i.e., $|C| = k$, it follows that $[p/C]$ is a ground atom. □

We now show that the ground actions generated by maximum cliques are applicable when the arity of each literal in the precondition is at most 2.

Theorem 1. *Let A be an action schema with arity $k \geq 2$, p denote the atom of a literal $l \in pre(A)$ with arity 1 or 2, $\mathcal{S}$ be a state, and C represent a clique of size k in $G_{A,\mathcal{S}}$. Then, in $\mathcal{S}$, $[p/C]$ holds with respect to the polarity of l.*

Proof. We first note that $[p/C]$ is a ground atom according to Lemma 1, and et $G_{A,\mathcal{S}} = \langle V, E\rangle$ be the substitution consistency graph. We use proof by contradiction to show with two cases, where the arity of l is unary or binary.

Unary case. Since l is unary, we get $[p/C] = [P(x)/C] = P(o)$, where P is the predicate symbol of p. Consequently, the substitution $v = [x/o]$ must be in C. Now, we need to consider two subcases: l being either positive or negative. In the positive case, assume that l does not hold, i.e., $P(o) \notin \mathcal{S}$. Since $k \geq 2$, there must exist an edge $e \in E$ such that $v \in e$ and $e \subseteq C$. By the definition of I^+ (the partial substitution of some positive literal does not match any ground atom in $\mathcal{S}$), the edge e must be in I^+ since $[p/C] \notin \mathcal{S}$ (otherwise, it would hold). This leads to a contradiction, as $e \notin E$ and $e \in E$ simultaneously. The negative case is analogous and has been omitted for brevity.

Binary case. Since p is binary, we get $[p/C] = [P(x_1, x_2)/C] = P(o_1, o_2)$, where P is the predicate symbol of p. Consequently, the substitutions $e = \{[x_1/o_1], [x_2/o_2]\} \subseteq C$, and thus $e \notin I$ (since $e \in E$). Furthermore, $e \notin I^+$ and $e \notin I^-$ since $e \notin I$. We have two subcases to consider: l is either positive or negative. In the positive case, assume that l does not hold, i.e., $P(o_1, o_2) \notin \mathcal{S}$. However, the definition of I^+ states that e must be in I^+ since e does not match any ground atom in $\mathcal{S}$. In the negative case, assume that l does not hold, i.e., $P(o_1, o_2) \in \mathcal{S}$. This also leads to a contradiction, as the definition of I^+ states that e must be included in it. □

Nullary literals are also taken into account, but we do not show this explicitly to save space. In our implementation, we evaluate nullary literals before grounding action schemas: if they do not hold, then all possible instantiations result in inapplicable ground actions. If an action schema contains a literal in the precondition with an arity greater than 2, then maximum cliques of size k might produce an inapplicable ground action. Fortunately, even in such cases, this method still reduces the number of potential substitutions, and to guarantee applicability, it is sufficient to test literals of higher arity to determine if they hold with respect to the state. In our benchmarks, most action schemas have literals of arity between 0 and 2 in the precondition.

7 Completeness

We will now show that our method enumerates all applicable ground actions (although it may not enumerate only the applicable ones).

Theorem 2. *Let a be a ground action from a k-ary schema A, where $k \geq 2$, that is applicable in state $\mathcal{S}$. Then, there exists a clique C of size k in $G_{A,\mathcal{S}}$ such that $a = [A/C]$.*

Proof. We note that Lemma 1 implies that $[A/C]$ is a ground action. Assume that $a = [A/C]$, but C is not a clique in $G_{A,\mathcal{S}} = \langle V, E\rangle$. This implies that there exists an edge $e = \{[x_1/o_1], [x_2/o_2]\} \subseteq C$, where $x_1 \neq x_2$, such that $e \notin E$. Since E is defined as all possible edges except those in $I = I^{\neq} \cup I^+ \cup I^-$, e must belong to one of these sets, and we consider these three cases separately.

First case. The set $I^{\neq}$ contains substitutions of the same variable, but e cannot belong to $I^{\neq}$ since $x_1 \neq x_2$, i.e., a contradiction.

Second case. The set I^+ contains substitutions for which there is a positive literal p in the precondition, and $[p/e]$ does not match any ground atom in $\mathcal{S}$. However, since a is applicable, it must match some ground atom in $\mathcal{S}$ by the definition of applicability. This also results in a contradiction.

Third case. The set I^- contains substitutions for which there is a negative literal p in the precondition, and $[p/e] \in \mathcal{S}$. However, since a is applicable, p cannot exist. This case also leads to a contradiction.

In conclusion, we find a contradiction as e cannot belong to I. □

Theorem 2 states that our method enumerates every ground action applicable in the state. However, it does not guarantee that every enumerated ground action is actually applicable. In fact, our method enumerates all and only applicable ground actions if the arity of every literal in the precondition is at most 2, as stated by Theorem 1. This implies that, in the general case, our method *over-approximates* the set of applicable ground actions.

8 Experimental Results

We have implemented our successor generator[1] within the lifted planner Powerlifted [8] to enable a fair comparison between the two methods. However, as their successor generator does not support negative preconditions, we have limited the benchmark set to STRIPS domains that include equalities, inequalities, and types. Our method treats equalities, inequalities, and types as static predicates, handling them without any special considerations. We compare our method with the top-performing successor generators, *Full Reducer* and *Yannakakis*, which are implemented in Powerlifted [8].

We evaluate using two benchmark sets:

1. IPC Benchmark: We use 47 domains from the International Planning Competition (IPC) that can be handled by both successor generators, the difficulty of these problems is scaled using Autoscale [39] to ensure that modern ground planners do not achieve perfect coverage; and
2. HTG Benchmark: We use 5 domains developed specifically to assess lifted planners [27]. This benchmark contains domains and problems that are hard-to-ground (HTG) due to aspects such as large action schema arity, large predicate arity, and many objects within the problems. (We used all domains uploaded to Zenodo; however, some domains are missing from this upload.)

We are interested in determining the amount of time spent identifying all (and only) applicable ground actions. To achieve this, we modified the breadth-first search algorithm to report the total time spent generating all successors for each g-value (shortest distance from the initial state). For each domain, we report the total time taken to reach the highest g-value that *all* methods could reach. This ensures that each method expanded exactly the same states, though the order may differ. The experiments were ran on an Intel Xeon Gold 6130 processor with 16 GB of memory, limited to 30 minutes.

8.1 IPC Benchmark

The total time and the number of expanded states for the IPC benchmark are presented in Table 1. As anticipated, no single method dom-

[1] Code: https://zenodo.org/record/8178591

Table 1: Time spent (in seconds) generating ground actions across 47 IPC domains using 4 methods. Each method expands the exact same states, with the total number displayed in the "expanded" column. The "OA" column indicates the *over-approximation* factor for our clique methods applied to the *largest problem* within the domain. The last 5 columns present the number of predicates with specific arities. The symbol # represents the number of instances in each domain where all methods could fully expand all states with the same g-value, $g > 0$.

			Clique Methods		Database Methods			Pred. count by arity				
Domain	#	OA	Bron-Kerbosch (s)	k-Clique k-Partite (s)	Yannakakis (s)	Full Reducer (s)	Expanded	0	1	2	3	4
agricola	98	1.0	7 276	5 019	3 280	**2 911**	34 678 658	3	16	12	1	-
airport	50	1.0	5 804	**5 444**	15 360	15 202	14 222 412	-	4	7	-	-
barman	168	1.0	6 256	5 734	4 085	**3 076**	90 206 739	-	6	9	-	-
blocksworld	95	1.0	3 715	3 945	3 276	**2 479**	208 133 095	1	3	1	-	-
childsnack	90	1.0	891	**578**	1 599	1 381	9 822 714	-	10	3	-	-
depots	82	1.0	4 052	2 382	2 433	**1 639**	26 775 199	-	2	4	-	-
driverlog	80	1.0	935	1 112	775	**596**	8 384 330	-	1	5	-	-
elevators	130	1.0	6 857	**3 277**	24 430	8 850	44 757 369	-	-	8	-	-
ferry	132	1.0	**2 766**	2 918	21 108	14 222	470 564 308	1	4	2	-	-
floortile	130	1.0	2 802	1 938	2 254	**1 620**	25 945 919	-	2	7	-	-
freecell	136	1.0	6 259	**2 875**	4 673	4 408	3 714 030	-	6	5	-	-
ged	74	1.0	1 223	1 140	1 031	**768**	49 916 377	8	12	2	-	-
goldminer	144	1.0	**5 413**	5 674	17 212	14 638	276 802 870	4	7	1	-	-
grid	169	1.0	**106**	1 630	330	297	3 265 494	1	7	4	-	-
gripper	60	1.0	**159**	663	506	339	11 786 834	-	5	2	-	-
hanoi	30	1.0	1 145	1 112	975	**881**	28 320 364	-	1	2	-	-
hiking	217	1.0	27 318	6 362	9 478	**5 752**	98 239 560	-	2	5	1	-
logistics	123	1.0	695	918	870	**588**	6 272 238	-	6	3	-	-
miconic	207	1.0	**8 952**	9 206	144 662	123 483	463 642 291	-	3	3	-	-
movie	30	1.0	0.19	**0.19**	0.43	0.39	3 600	9	5	-	-	-
mprime	94	1.0	23 887	**1 376**	3 625	2 666	17 035 094	-	-	8	-	-
mystery	30	1.0	309	206	106	**81**	763 565	-	5	7	-	-
no-mprime	35	1.0	979	172	**112**	116	426 805	-	5	7	-	-
no-mystery	30	1.0	333	201	116	**94**	763 565	-	5	7	-	-
nomystery	100	233.3	10 033	**1 252**	12 719	7 090	1 964 825	-	-	4	2	-
npuzzle	30	1.0	509	734	566	**409**	19 668 924	-	1	2	-	-
openstacks	140	1.0	12 757	**5 729**	8 816	8 813	62 906 805	-	5	2	-	-
organic-synthesis-split	47	1.0	**1 647**	1 654	1 668	1 878	2 811 167	1 081	31	3	-	-
parcprinter	40	1.0	1 767	**1 596**	3 878	3 011	45 674 374	1	1	7	2	-
parking	140	1.0	1 592	1 722	1 374	**1 095**	19 709 327	-	3	2	-	-
pegsol	60	1.0	**138**	289	214	199	7 118 285	1	3	-	1	-
petri-net-alignment	20	1.0	20 963	20 494	13 991	**13 817**	321 274 302	1	2	-	-	-
pipesworld-notankage	50	1.0	5 415	**1 346**	3 216	2 389	10 485 305	-	5	6	1	-
pipesworld-tankage	50	1.0	25 542	1 473	1 701	**1 285**	3 346 486	-	7	6	2	-
psr-small	50	1.0	162	**149**	170	169	15 757 819	107	-	-	-	-
rovers	100	1.3	2 010	**1 690**	5 920	4 580	19 572 619	-	11	12	2	-
satellite	96	1.0	686	609	1 154	**522**	7 530 299	-	3	5	-	-
scanalyzer	65	1.0	26 553	317	373	**252**	3 222 559	-	1	3	-	2
sokoban	80	1.8	5 230	1 177	546	**529**	2 628 776	-	4	1	1	-
spanner	132	1.0	3 511	2 711	2 863	**2 118**	45 528 422	-	3	3	-	-
thoughtful	52	1.0	12 915	1 783	1 654	**1 533**	964 573	-	9	7	-	-
tpp	90	1.5	33 593	1 899	3 037	**1 625**	20 548 407	-	-	4	3	-
transport	177	1.0	10 868	**4 141**	6 070	4 363	87 003 504	-	-	5	-	-
trucks	30	1.0	**144**	156	173	177	1 945 028	4 363	-	-	-	-
visitall	126	1.0	**42 245**	45 631	52 895	48 841	308 144 016	-	2	1	-	-
woodworking	130	1.0	14 921	**1 244**	2 149	2 886	7 273 773	-	4	10	-	-
zenotravel	20	1.0	281	127	127	**106**	789 555	-	4	4	-	-
Total	4 299	-	352 883	161 829	387 594	313 799	2 910 312 583	-	-	-	-	-

Table 2: Time spent (in seconds) generating ground actions across 5 HTG domains using 4 methods. Each method expands the exact same states, with the total number displayed in the "expanded" column. The "OA" column indicates the *over-approximation* factor for our clique methods applied to the *largest problem* within the domain. The last 5 columns present the number of predicates with specific arities. The symbol # represents the number of instances in each domain where all methods could fully expand all states with the same g-value, $g > 0$.

			Clique Methods		Database Methods			Pred. count by arity				
Domain	#	OA	Bron-Kerbosch (s)	k-Clique k-Partite (s)	Yannakakis (s)	Full Reducer (s)	Expanded	0	1	2	3	4
blocksworld-large-simple	40	1.0	47.57	45.37	20.99	**15.34**	46 040	1	3	1	-	-
childsnack-contents	144	1.0	2 388.06	**1 108.32**	3 974.88	2 709.03	30 553 449	-	5	6	-	-
logistics-large-simple	40	1.0	7.36	16.81	1.22	**1.17**	40	-	6	3	-	-
rovers-large-simple	40	1.0	83.43	209.68	**4.29**	4.38	839	-	11	12	2	-
visitall-multidimensional	60	1.0	756.83	1 014.27	348.45	**263.91**	7 201 010	-	-	1	2	-
Total	324	-	3 283.25	2 394.44	4 349.83	2 993.82	37 801 378	-	-	-	-	-

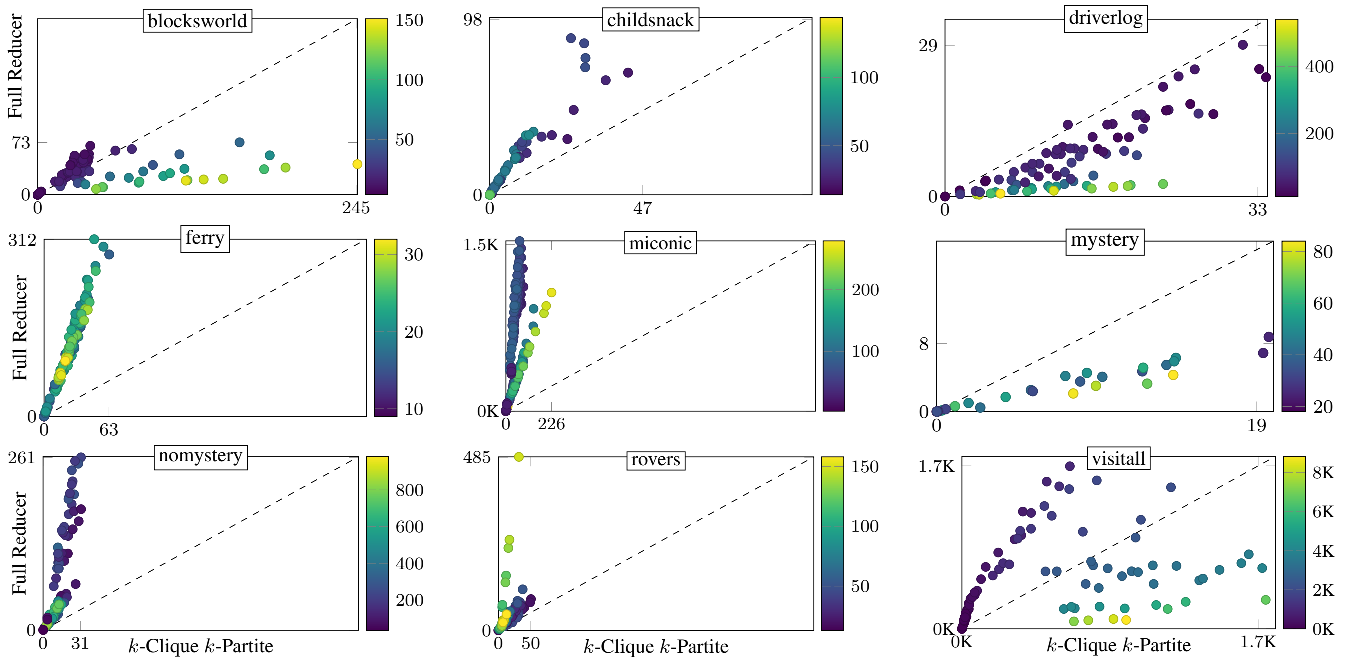

Figure 2: Scatter plots for 9 different IPC domains compare the performance of k-Clique k-Partite to Full Reducer. Each dot's position represents the total time in seconds required to expand the same number of states for both methods in a specific problem, while the color (indicated by the colorbar) denotes the number of objects in the given problem.

inates all others, which is consistent with the inherent difficulty of outperforming every other algorithm for a specific NP-hard problem. Our proposed algorithms demonstrate improved performance over competing algorithms in certain domains, as they may efficiently exploit the structure of instances in these domains. However, this also applies to methods based on database techniques. That said, the results in Table 1 validate the effectiveness of our approach, as it achieves the lowest total time in 22 domains.

A noteworthy observation is the considerable difference between the Bron-Kerbosch and k-Clique k-Partite methods. In most domains, the latter method either strictly outperforms or is on par with the former. However, in a few cases (e.g., grid, gripper, npuzzle, and pegsol), total time increases significantly. This is unexpected since the algorithm should be better suited for exploiting the graph's structure, as it is explicitly tailored for finding cliques of size k in k-partite graphs, unlike Bron-Kerbosch. That said, k-Clique k-Partite significantly improves the total time in the majority of domains, sometimes by a very significant amount (e.g., hiking, mprime, no-mprime, nomystery, pipesworld-notankage, pipesworld-tankage, scanalyzer, sokoban, thoughtful, tpp, and woodworking). This highlights that two distinct algorithms for the same NP-hard problem can exhibit markedly different performance characteristics, making both variations valuable for lifted planners.

With respect to methods based on database techniques, Full Reducer nearly dominates Yannakakis, with the latter only obtaining the lowest total time in one domain. While Full Reducer achieves the lowest total time in most domains, there are instances where it is substantially slower than the top-performing method (e.g., airport, elevators, grid, miconic, visitall). This observation is also mirrored in the aggregated total time, where k-Clique k-Partite scores lower than all other methods by a notable margin. Although one should not place much emphasis on the actual number, it does suggest that this method experiences fewer domains with extremely poor performance.

We also investigated whether the aggregated times in Table 1 conceal a more nuanced perspective on the performance of the methods. Figure 2 contains scatter plots for a few domains where the performance between k-Clique k-Partite and Full Reducer differed significantly. The plots depict the performance of both methods for individual problems, as opposed to an aggregated score. Additionally, the color of each point indicates the number of objects in the problem, which roughly correlates with the problem's difficulty, although not necessarily. We note that since all methods are required to expand all states up to the same g-value, more difficult problems tend to take less time in the plots, as reaching the next g-value simply takes too much time or memory. In some domains, such as blocksworld, driverlog and visitall, the performance ratio between the methods is not simply a fixed constant but depends on the number of objects in the problem. However, for most domains, we note that the number of objects does not appear to heavily influence the performance ratio.

According to the memory usage reported by Powerlifted, all methods used nearly the same amount of memory across all domains. It is worth noting that this number includes both the open list and the closed set, whose memory footprint often overshadow that of the successor generators.

8.2 *HTG Benchmark*

The total time and number of expanded states for the HTG benchmark are presented in Table 2. These instances are generally much larger, making grounding more challenging. Yannakakis and Full Reducer achieve the lowest total time in all domains except for childsnack. Interestingly, our methods demonstrated the best performance for rovers and visitall in the IPC benchmark, but not in the HTG benchmark. This may suggest that Yannakakis and Full Reducer scale better with the number of objects. However, as indicated by the predicate arities in Table 2, the encoding for childsnack and visitall differs from the IPC encoding. Hence, it is difficult to compare the results for these domains across the two benchmarks.

8.3 Over-Approximation

We also take a closer look at whether *over-approximation* of the set of applicable actions poses a problem in practice. Both Table 1 and Table 2 display the over-approximation *factor* for the *largest* problem in the domain (column "OA"). We focus on the largest problem to ensure that smaller problems, which may be less susceptible to over-approximation, do not skew the factor towards 1.0. A factor of 1.0 signifies that only applicable ground actions were generated by our methods. As mentioned earlier, over-approximation occurs only when there are predicates with an arity greater than 2. Consequently, the tables also provide information about the predicates in the domain (the last five columns). Surprisingly, the ratio is precisely 1.0 for most domains containing predicates of higher arity.

In nearly all domains, over-approximation is not an issue. The notable exception is nomystery, where our method exhibits an over-approximation factor of 233.3. Intriguingly, in this domain, our method also outperforms the database methods by a significant margin despite over-approximation. The ternary predicate in nomystery is *fuelcost*, which results in our method generating multiple *drive* ground actions for various fuel transitions. Nonetheless, this is the *only* atom in the precondition of the generated ground action that needs to be tested for its validity in the given state. Overall, over-approximation does not seem to be a practical bottleneck.

9 Future Work

The results of our approach are promising, as it outperforms the state-of-the-art in many domains. We will briefly discuss the design of an exact method (without over-approximation) and present several options for enumerating maximum cliques.

9.1 Exact Methods

Interestingly, for many domains with predicates of higher arities, our method does not over-approximate significantly, if at all. Nevertheless, to ensure that only applicable ground actions are enumerated, we can generalize the substitution consistency graph. Assume that the arity of all atoms is bounded by n. Then, instead of considering only single substitutions in the consistency graph, we let each vertex to represent up to $n-1$ consistent substitutions. An edge exists between two sets of substitutions if they are consistent with one another. The rules for omitting an edge can also be generalized. For instance, two substitution sets are inconsistent if they substitute the same variable (even with the same value). Our goal is then to find a maximal clique such that the cardinality of the union of all substitution sets is equal to the action schema's arity.

Clearly, constructing this graph is more costly, so the extra effort might not be worthwhile and should likely only be undertaken when over-approximation is extremely severe. We believe that over-approximation can be reliably detected during runtime by simply running the current method on a few states. If many inapplicable ground actions are generated, then one could switch to a different method (or simply a better performing one).

9.2 Clique Algorithms

The performance differences between Bron-Kerbosch [4] and k-Clique k-Partite [20, 30] emphasize the importance of the maximum clique enumeration algorithm. The number of maximal cliques in a graph with n vertices is at most $3^{n/3}$ [31], and with a good pivot function, the Bron-Kerbosch algorithm's worst-case complexity matches this bound [37]. However, exploiting the k-partite property of substitution consistency graphs and other properties with tailored algorithms can lead to better performance.

Another property is the *degeneracy* of the graph, which is the smallest value k such that for every subgraph, there exists a vertex with a degree of at most k. Degeneracy is a measure of *sparsity*; if k is small, the graph is considered sparse. In our case, the degeneracy can be large: consider an action schema with an empty precondition, then the degree of each vertex is exactly $n(m-1)$, with n being the number of objects and m the arity of the action schema. Hence, an algorithm that does not depend on k might be suitable (e.g., [24]).

We also note that there are classes of graphs where the maximal clique enumeration problem is tractable. For instance, a graph is *triangulated* if and only if every cycle of at least length 4 has an edge between two non-consecutive vertices in the cycle. For such graphs, the number of maximal cliques is linear in input size [16], and they can be found in time linear in input size [17]. It might be the case that for some domains, the substitution consistency graphs fall into some tractable fragment that can be exploited.

Lastly, we note that the Bron-Kerbosch algorithm is not an output-sensitive algorithm (i.e., its time complexity does not depend on the output size), which poses a problem in domains where the number of applicable actions per state is small, and it is unclear whether the k-Clique k-Partite algorithm is output-sensitive. There are output-sensitive algorithms (e.g. [40, 7, 24, 28, 6]) for enumerating all maximum cliques, or equivalently all independent sets in the inverse graph, and they might result in better performance where the number of applicable actions does not depend on input size (e.g., visitall).

10 Conclusions

We demonstrated that maximum clique enumeration can be effectively used to generate all applicable actions in a lifted planner. Our method outperformed the state-of-the-art in many domains, proving to be highly competitive. We also examined whether our method suffers from over-approximation in practice; it appears that it does not.

We observed that the choice of algorithm used to enumerate all maximum cliques significantly impacts performance, and that no single algorithm is likely to dominate all others. Consequently, it is probable that we can achieve much better performance in some domains by simply changing the clique enumeration algorithm. Fortunately, it is easy to select the best successor generator by simply running each one on a few states and measuring their performance. This should be done on a per-action schema basis, rather than for the entire problem. This is not limited to just our methods, but to all lifted successor generators, and it illustrates the importance of having a diverse set to choose from.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at the National Supercomputer Centre at Linköping University partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973.

References

[1] Christer Bäckström and Bernhard Nebel, 'Complexity results for SAS+ planning', *Computational Intelligence*, **11**, 625–656, (1995).

[2] Blai Bonet, Guillem Francès, and Hector Geffner, 'Learning features and abstract actions for computing generalized plans', in *The 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pp. 2703–2710, (2019).

[3] Blai Bonet and Hector Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**(1), 5–33, (2001).

[4] Coenraad Bron and Joep Kerbosch, 'Finding all cliques of an undirected graph (algorithm 457)', *Communications of the ACM*, **16**(9), 575–576, (1973).

[5] Ashok K. Chandra and Philip M. Merlin, 'Optimal implementation of conjunctive queries in relational data bases', in *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77)*, pp. 77–90, (1977).

[6] Lijun Chang, Jeffrey Xu Yu, and Lu Qin, 'Fast maximal cliques enumeration in sparse graphs', *Algorithmica*, **66**, 173–186, (2012).

[7] Norishige Chiba and Takao Nishizeki, 'Arboricity and subgraph listing algorithms', *SIAM Journal on Computing*, **14**(1), 210–223, (1985).

[8] Augusto B. Corrêa, Florian Pommerening, Malte Helmert, and Guillem Francès, 'Lifted successor generation using query optimization techniques', in *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, pp. 80–89, (2020).

[9] Dominik Drexler, Jendrik Seipp, and Hector Geffner, 'Expressing and exploiting the common subgoal structure of classical planning domains using sketches', in *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR'21)*, pp. 258–268, (2021).

[10] Dominik Drexler, Jendrik Seipp, and Hector Geffner, 'Learning sketches for decomposing planning problems into subproblems of bounded width', in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, pp. 62–70, (2022).

[11] Dominik Drexler, Jendrik Seipp, and Hector Geffner, 'Learning hierarchical policies by iteratively reducing the width of sketch rules', in *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR'23)*, (2023).

[12] David Eppstein, Maarten Löffler, and Darren Strash, 'Listing all maximal cliques in sparse graphs in near-optimal time', in *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, eds., Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, volume 6506 of *Lecture Notes in Computer Science*, pp. 403–414, (2010).

[13] Richard E. Fikes and Nils J. Nilsson, 'Strips: A new approach to the application of theorem proving to problem solving', in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71)*, pp. 608–620, (1971).

[14] Guillem Francès, *Effective planning with expressive languages*, Ph.D. dissertation, Pompeu Fabra University, Spain, 2017.

[15] Guillem Francès, Blai Bonet, and Hector Geffner, 'Learning general planning policies from small examples without supervision', in *35th AAAI Conference on Artificial Intelligence (AAAI'21)*, pp. 11801–11808, (2021).

[16] Delbert R. Fulkerson and Oliver A. Gross, 'Incidence matrices and interval graphs', *Pacific Journal of Mathematics*, **15**(3), 835–855, (1965).

[17] Fănică Gavril, 'Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph', *SIAM Journal on Computing*, **1**(2), 180–187, (1972).

[18] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins, 'PDDL - the planning domain definition language - version 1.2', *Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control*, (1998).

[19] Daniel Gnad, Álvaro Torralba, Martín Ariel Domínguez, Carlos Areces, and Facundo Bustos, 'Learning how to ground a plan - partial grounding in classical planning', in *The 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, pp. 7602–7609, (2019).

[20] Tore Grünert, Stefan Irnich, Hans-Jürgen Zimmermann, Markus Schneider, and Burkhard Wulfhorst, 'Finding all k-cliques in k-partite graphs, an application in textile engineering', *Computers & Operations Research*, **29**(1), 13–31, (2002).

[21] Malte Helmert, 'The fast downward planning system', *Journal of Artificial Intelligence Research (JAIR)*, **26**, 191–246, (2006).

[22] Malte Helmert, 'Concise finite-domain representations for PDDL planning tasks', *Artificial Intelligence*, **173**(5-6), 503–535, (2009).

[23] Philippe Jégou, 'Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems', in *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pp. 731–736, (1993).

[24] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou, 'On generating all maximal independent sets', *Information Processing Letters*, **27**(3), 119–123, (1988).

[25] Richard M. Karp, 'Reducibility among combinatorial problems', in *Proceedings of a symposium on the Complexity of Computer Computations*, pp. 85–103, (1972).

[26] Henry A. Kautz and Bart Selman, 'Planning as satisfiability', in *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pp. 359–363, (1992).

[27] Pascal Lauer, Álvaro Torralba, Daniel Fiser, Daniel Höller, Julia Wichlacz, and Jörg Hoffmann, 'Polynomial-time in PDDL input size: Making the delete relaxation feasible for lifted planning', in *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pp. 4119–4126, (2021).

[28] Kazuhisa Makino and Takeaki Uno, 'New algorithms for enumerating all maximal cliques', in *Scandinavian Workshop on Algorithm Theory (SWAT'04)*, pp. 260–272, (2004).

[29] Mario Martín and Hector Geffner, 'Learning generalized policies from planning examples using concept languages', *Applied Intelligence*, **20**(1), 9–19, (2004).

[30] Mohammad Mirghorbani and Pavlo A. Krokhmal, 'On finding k-cliques in k-partite graphs', *Optimization Letters*, **7**, 1155–1165, (2013).

[31] John W. Moon and Leo Moser, 'On cliques in graphs', *Israel Journal of Mathematics*, **3**, 23–28, (1965).

[32] Silvia Richter and Matthias Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *Journal of Artificial Intelligence Research (JAIR)*, **39**, 127–177, (2010).

[33] Bernardus Ridder, *Lifted heuristics: towards more scalable planning systems*, Ph.D. dissertation, King's College London, UK, 2014.

[34] Simon Ståhlberg, Blai Bonet, and Hector Geffner, 'Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits', in *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, pp. 629–637, (2022).

[35] Simon Ståhlberg, Blai Bonet, and Hector Geffner, 'Learning generalized policies without supervision using gnns', in *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning (KR'22)*, pp. 474–483, (2022).

[36] Simon Ståhlberg, Blai Bonet, and Hector Geffner, 'Learning general policies with policy gradient methods', in *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR'23)*, (2023).

[37] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi, 'The worst-case time complexity for generating all maximal cliques and computational experiments', *Theoretical Computer Science*, **363**(1), 28–42, (2006).

[38] Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp, 'Efficient symbolic search for cost-optimal planning', *Artificial Intelligence*, **242**, 52–79, (2017).

[39] Álvaro Torralba, Jendrik Seipp, and Silvan Sievers, 'Automatic instance generation for classical planning', in *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, pp. 376–384, (2021).

[40] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa, 'A new algorithm for generating all the maximal independent sets', *SIAM Journal on Computing*, **6**(3), 505–517, (1977).

[41] Julia Wichlacz, Daniel Höller, and Jörg Hoffmann, 'Landmark heuristics for lifted classical planning', in *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI'22)*, pp. 4665–4671, (2022).

[42] Mihalis Yannakakis, 'Algorithms for acyclic database schemes', in *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB'81)*, pp. 82–94, (1981).