Lifted Stackelberg Planning

Philipp Sauer[1], Marcel Steinmetz[1], Robert Künnemann[2], Jörg Hoffmann[1,3]

[1] Saarland University, Saarland Informatics Campus, Germany
[2] CISPA Helmholtz Center for Information Security, Germany
[3] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
s8phsaue@stud.uni-saarland.de, robert.kuennemann@cispa.saarland, {steinmetz,hoffmann}@cs.uni-saarland.de

Abstract

In Stackelberg planning, a leader and a follower each choose a plan in the same planning task, the leader's objective being to maximize plan cost for the follower. This formulation naturally captures, among others, security-related scenarios where the leader defends an infrastructure against subsequent attacks by the follower. Indeed, Stackelberg planning has been applied to the analysis of email infrastructure security. At web scale, however, the planning tasks involved easily contain tens of thousands of objects (hosts), so that grounding becomes the bottleneck. Here we introduce a lifted form of Stackelberg planning to address this. We devise leader-follower search algorithms working at the level of the PDDL-style input model to the extent possible. Our experiments show that, in Stackelberg tasks with many objects, including in particular models of web infrastructure security, our lifted algorithms outperform grounded Stackelberg planning.

Introduction

Stackelberg planning (Speicher et al. 2018a) is a framework inspired by Stackelberg security games (Tambe 2011). It models a single exchange of adversarial plan choice between two agents acting in the same planning task, the *leader* and the *follower*. The follower has a goal while the leader pursues the objective to maximize the follower's plan cost. The solution to such a task is the Pareto front of leader/follower plan pairs where the leader cannot decrease their own cost without also decreasing the optimal follower-plan cost. Speicher et al. (2018a) introduced *leader-follower search* to find such solutions, for inputs modeled in an extension of PDDL. This algorithm interleaves two search levels, running state-space search at the leader level, where each state spawns a classical planning task at the follower level. Speicher et al. also devised branch-and-bound style pruning and partial-order reduction for the leader-level search.

Stackelberg planning is a natural tool for analyzing countermeasures in security applications, e.g., network penetration testing (Boddy et al. 2005; Hoffmann 2015) or email-infrastructure threat analysis (Speicher et al. 2018b), the leader modeling defense mechanisms against attack vectors represented by the follower. Recently, Tizio et al. (2022) used Stackelberg planning for analyzing the open web infrastructure, but found that off-the-shelf planners suffer from poor scalability. With instances that easily contain many thousands of objects, Tizio et al. quickly identified the grounding preprocess as a major bottleneck.

Lifted planning methods, which work at the level of the PDDL input as much as possible, and hence do not require a grounding preprocess, are the natural remedy to this problem. The approach has a long history (e.g., Penberthy and Weld 1992; Russell and Norvig 1995; Younes and Simmons 2003; Ridder and Fox 2014). It has recently moved into focus in heuristic search planning, based on the effective lifted forward state-space search algorithm by Corrêa et al. (2020), which enabled the investigation of lifted heuristic functions (e.g., Corrêa et al. 2021; Lauer et al. 2021).

Here, we show how to apply these ideas to Stackelberg planning. We devise leader-follower search algorithms working at the lifted level to the extent possible. To this end, we leverage Corrêa et al.'s (2020) lifted search methodology in both leader and follower search. For leader search, we devise lifted variants of Speicher et al.'s (2018a) branch-and-bound pruning and partial-order reduction methods.

We run experiments on Speicher et al.'s (2018a) Stackelberg variants of IPC and pentesting benchmarks, on variants of these benchmarks scaled to have many objects, and on web infrastructure defense-attack models as per Tizio et al. (2022). Our experiments show that, while lifted Stackelberg planning is inferior on small tasks, it is typically vastly superior on large tasks. In particular, on the models of web infrastructure security, our lifted algorithms outperform grounded Stackelberg planning.

Detailed proofs and descriptions of our benchmark set are available in a technical report (Sauer et al. 2023).

Preliminaries

A lifted classical planning task (Corrêa et al. 2020) is a tuple $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{P}$, $\mathcal{O}$, and $\mathcal{A}$ are finite sets of *predicate symbols*, *objects*, and *action schemas* respectively, $\mathcal{I}$ is the *initial state*, and $\mathcal{G}$ is the *goal*. Predicates symbols $p \in \mathcal{P}$ are parameterized by variables $\mathbf{x}_p$. Objects and variable symbols are called *terms*. Given a tuple of terms θ, with $|\theta| = |\mathbf{x}_p|$, $p\theta$ is an *atom*. $p\theta$ is a *ground atom* if θ contains no variables. The set of all ground atoms $\mathcal{P}^{\mathcal{O}}$ is given by the instantiations of all predicates with all possible combinations of objects. $|\mathcal{P}^{\mathcal{O}}|$ is in general exponential in the size of Π. Abusing notation, we typically refer to ground atoms with letters p, q, omitting the parameter specification if not

relevant for the discussion. A *state* $s \subseteq \mathcal{P}^{\mathcal{O}}$ is a set of ground atoms. The initial state and the goal both are states of Π.

Each action schema $a \in \mathcal{A}$ is associated with parameter variables $\mathbf{x}_a$; a *precondition* pre_a, *add effect* add_a and *delete effect* del_a, sets of atoms whose free variables are in $\mathbf{x}_a$; and a positive *cost* $c_a \in \mathbb{R}^+$ (we assume non-0 cost for brevity's sake). For a tuple of terms θ, with $|\theta| = |\mathbf{x}_a|$, we denote by $a\theta$ the *instance* of a, obtained by replacing in all components of a the variables by the corresponding terms. $a\theta$ is a *ground action* if θ contains no variables. For an instance $a\theta$, we write $[a\theta]^{\mathcal{O}}$ for the set of all ground actions of a whose parameters match the θ object assignment. The set of all ground actions is denoted $\mathcal{A}^{\mathcal{O}}$. For brevity, we omit the parameter specification if not relevant. A ground action $a \in \mathcal{A}^{\mathcal{O}}$ is applicable in a state s if $pre_a \subseteq s$. The set of all ground actions applicable in s is denoted $\mathcal{A}^{\mathcal{O}}(s)$. For $a \in \mathcal{A}^{\mathcal{O}}(s)$, the application of a in s results in the state $s[\![a]\!] = (s \setminus del_a) \cup add_a$. For a set of ground atoms $P \subseteq \mathcal{P}^{\mathcal{O}}$, the *regression* of P by a is defined by $\textit{Regress}(P, a) = (P \cup pre_a) \setminus add_a$ if $del_a \cap P = \emptyset$, and undefined otherwise. We assume here that $add_a \cap pre_a = \emptyset$, which comes w.l.o.g. because all ground atoms in $add_a \cap pre_a$ can be equivalently omitted from add_a. All these definitions are extended to sequences of ground actions π in an iterative manner. The cost of π is the sum of costs of its actions. If π is applicable in s and $\mathcal{G} \subseteq s[\![\pi]\!]$, or equivalently if $\textit{Regress}(\mathcal{G}, \pi)$ is defined and $\textit{Regress}(\mathcal{G}, \pi) \subseteq s$, then π is a *plan* for s. π is *optimal* if it has minimal cost. A plan for Π is a plan for the initial state $\mathcal{I}$. If Π has no plan, we say that Π is *unsolvable*.

Lifted Stackelberg Planning

Prior works on Stackelberg planning have exclusively considered the grounded case. For brevity's sake, we do not provide a separate definition of ground Stackelberg planning tasks, but directly introduce our lifted variant. Our definition straightforwardly extends the one by Speicher et al. (2018a).

A lifted Stackelberg planning task is a tuple $\Pi_{\text{LF}} = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}_{\text{L}}, \mathcal{A}_{\text{F}}, \mathcal{I}, \mathcal{G}_{\text{F}} \rangle$ with similar components as before, except that the action schemas are partitioned into ones belonging to the *leader*, $\mathcal{A}_{\text{L}}$, and ones belonging to the *follower* $\mathcal{A}_{\text{F}}$. Let s be a state of Π_{LF}. A *leader plan* for s is a sequence of ground leader actions π_{L} such that π_{L} is applicable in $\mathcal{I}$ and $\mathcal{I}[\![\pi_{\text{L}}]\!] = s$. We denote by $L^*(s)$ the cost of the cheapest leader plan for s, if such a plan exists, and we define $L^*(s) = \infty$ otherwise. The s-follower task is the lifted classical planning task $\Pi_{\text{F}}(s) = \langle \mathcal{P}, \mathcal{O}, \mathcal{A}_{\text{F}}, s, \mathcal{G}_{\text{F}} \rangle$, i.e., the task with s as initial state and including only action schemas associated with the follower. A *follower plan* in s is a plan π_{F} for $\Pi_{\text{F}}(s)$. We denote by $F^*(s)$ the cost of an optimal follower plan; $F^*(s) = \infty$ if $\Pi_{\text{F}}(s)$ is unsolvable. A pair of cost values $\langle l, f \rangle$ *dominates* another pair $\langle l', f' \rangle$ if $l \leq l'$ and $f \geq f'$ and one of the inequalities is strict. A state s dominates a state t if $\langle L^*(s), F^*(s) \rangle$ dominates $\langle L^*(t), F^*(t) \rangle$. The solution to Π_{LF} is the *Pareto frontier* $\mathcal{F}^*$ consisting of all leader-reachable non-dominated states.

Background: Leader-Follower Search

We briefly revisit Speicher et al.'s (2018a) *leader-follower search (LFS)* algorithm and pruning optimizations for computing the pareto frontier of grounded Stackelberg tasks.

LFS is a two-fold search method. The outer (the "leader") search performs a Dijkstra-like exploration of the leader state space, visiting all states s reachable from $\mathcal{I}$ by applying leader actions only, and computing $L^*(s)$ along the way. For every visited state s, the associated follower classical planning task $\Pi_{\text{F}}(s)$ is solved optimally (the inner "follower search"). During this process, the algorithm maintains the set of states $\mathcal{F}$ not dominated so far. Upon termination, the algorithm has considered all leader-reachable states, guaranteeing that $\mathcal{F} = \mathcal{F}^*$. Speicher et al. (2018a) extend this basic algorithm by three correctness-preserving pruning methods:

Follower-search pruning The $F^*(s)$ computation is skipped if some entry in $\mathcal{F}$ dominates $\langle L^*(s), u_{\text{F}}(s) \rangle$, where $u_{\text{F}}(s) \geq F^*(s)$ is given by the cost of s's parent follower plan if that remained valid, and ∞ otherwise.

Leader-search pruning s is pruned from the leader search entirely if $L^*(s) > u_{\text{L}}$, where $\langle u_{\text{L}}, u_{\text{F}}^{\forall} \rangle$ estimates the $\mathcal{F}^*$ entry with maximal follower-plan cost. The upper bound $u_{\text{F}}^{\forall}$ on the maximal follower-plan cost is obtained prior to the LFS by computing $F^*(\mathcal{I}^-)$, where $\mathcal{I}^- \subseteq \mathcal{I}$ and $\mathcal{I}^- \cap del_a = \emptyset$ for all $a \in \mathcal{A}_{\text{L}}^{\mathcal{O}}$ that are delete-relaxed-reachable from $\mathcal{I}$ via ground leader actions. u_{L} is computed in a branch-and-bound fashion during the leader search, starting with $u_{\text{L}} = \infty$, and setting $u_{\text{L}} = L^*(s)$ whenever a state s with $F^*(s) = u_{\text{F}}^{\forall}$ is encountered.

Partial-order reduction Speicher et al. (2018a) devised a variant of *strong stubborn sets (SSSs)* (Valmari 1989; Wehrle and Helmert 2014) to further prune the leader search space. An SSS in a state s characterizes a set of ground leader actions $G \subseteq \mathcal{A}_{\text{L}}^{\mathcal{O}}$, whose consideration at s in the leader search suffices for preserving the $\mathcal{F} = \mathcal{F}^*$ termination guarantee.

Strong Stubborn Sets

An SSS G in s is a fixed point of the following iterative procedure: (1) starting from a set of ground leader actions necessarily appearing on any path from s to any state in $\mathcal{F}^*$; iteratively consider all ground actions $a \in G$, (2) if a is not applicable in s, include all ground leader actions potentially needed to enable a; (3) otherwise, include all ground leader actions whose application is not commutative with a. The actions to add in (2) and (3) are characterized via the same notions as in classical planning SSSs: let $a_1, a_2 \in \mathcal{A}_{\text{L}}^{\mathcal{O}}$, and $p \in pre_{a_2}$. a_1 *disables* a_2 if $del_{a_1} \cap pre_{a_2} \neq \emptyset$; a_1 *enables* a_2 on p if a_1 does not disable a_2 and $p \in add_{a_1}$; a_1 and a_2 *conflict* if $add_{a_i} \cap del_{a_j} \neq \emptyset$ for $i \neq j$; and a_1 and a_2 *interfere* if they conflict or one disables the other.

To seed the computation, Speicher et al. (2018a) made the following observation. Consider any state s^* whose optimal leader plan passes through s. Let π_{F}^* be an optimal follower plan for s. Due to the selection of s^*, $L^*(s^*) > L^*(s)$. So, if $s^* \in \mathcal{F}^*$, then $F^*(s^*) > F^*(s)$. But then, π_{F}^* cannot be a leader plan for s^*, i.e., for (1), one can simply start with the ground leader actions that invalidate $\textit{Regress}(\mathcal{G}_{\text{F}}, \pi_{\text{F}}^*)$:

Definition 1. *Let s be a state of Π_{LF}, and let π^*_{F} be an optimal follower plan for s. Let $G \subseteq \mathcal{A}^{\mathcal{O}}_{\text{L}}$ be a set of ground leader actions. Then G is an* SSS *in s for π^*_{F} if*

(1) Let $a \in \mathcal{A}^{\mathcal{O}}_{\text{L}}$ be any ground leader action. If $del_a \cap$ Regress$(\mathcal{G}_{\text{F}}, \pi^*_{\text{F}}) \neq \emptyset$, then $a \in G$.
(2) Let $a \in (G \setminus \mathcal{A}^{\mathcal{O}}_{\text{L}}(s))$. There is some $p \in (pre_a \setminus s)$ s.t. G contains all ground leader actions that enable a on p.
(3) Let $a \in (G \cap \mathcal{A}^{\mathcal{O}}_{\text{L}}(s))$. G contains all ground leader actions that interfere with a.

Theorem 1 (Speicher et al. (2018a))**.** *Let s^* be a state with an optimal leader plan passing through s. Consider a leader-action sequence with $s^* = s[\![\langle a_1, \dots, a_n \rangle]\!]$. There is an i: $a_i \in G$ and $s^* = s[\![\langle a_i, a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n \rangle]\!]$.*

In other words, the leader actions applicable in s but not contained in G can simply be reordered after the actions in G, so can be skipped during s's expansion in the search.

Lifted Leader-Follower Search

The LFS algorithm interfaces with the planning task at exactly three places: (1) the successor state computation during the leader state-space traversal; (2) generating and solving the follower tasks, and (3) the pruning methods. To *lift* LFS, we need to provide appropriate interface replacements operating at the lifted task description directly. The remaining algorithm steps work as before.

(1) and (2) are straightforward. For (1), we can immediately plug in the lifted forward-search methods developed by Corrêa et al. (2020) for classical planning. Regarding (2), naturally, lifted LFS will generate lifted follower tasks. Solving them can be delegated to any off-the-shelf lifted classical planner without necessitating any grounding preprocess.

The challenging part is (3). Follower-search pruning works out-of-the-box. To adapt leader-search pruning, we leverage prior work in (a) replacing in the computation of $\mathcal{I}^-$ the grounded delete-relaxed exploration by Corrêa et al.'s (2021) lifted variant, and afterwards (b) computing $F^*(\mathcal{I}^-)$ by calling a lifted classical planner on $\Pi_{\text{F}}(\mathcal{I}^-)$.

Lifted Strong Stubborn Sets

To be able to construct an SSS directly from the lifted task description, we devise conditions on leader-action-schema *instances* S, rather than ground actions, guaranteeing that the represented ground actions $[S]^{\mathcal{O}}$ satisfy Definition 1.

We characterize relevant parameter instantiations through FOL substitutions. This is, for a set of atoms P with free variables $\mathbf{x}$ and a (partial) assignment θ of $\mathbf{x}$ to objects, $P\theta$ denotes the set of atoms resulting from replacing variables by objects according to θ. We say that θ *joins* P with a ground atom p if $p \in P\theta$. θ is *minimal* if there is no θ' that joins P with p and $\theta' \subset \theta$. The set of all minimal joins is denoted as $J^*(P, p)$, which can be computed efficiently by finding the unifiers of p with the corresponding atoms in P.

We lift the SSS conditions via three sets: For $p \in \mathcal{P}^{\mathcal{O}}$, we denote by $\mathit{Pre}(p) = \{a\theta \mid a \in \mathcal{A}_{\text{L}}, \theta \in J^*(pre_a, p)\}$ the set of action-schema instances $a\theta$ whose parameters are instantiated just enough for the ground atom p to appear in the precondition of $a\theta$. $\mathit{Add}(p)$ and $\mathit{Del}(p)$ are defined similarly.

Definition 2 (Lifted SSS)**.** *Let s be a state with optimal follower plan π^*_{F}. A set of leader-action-schema instances S is a* lifted SSS *in s for π^*_{F} if*

(i) For every $p \in$ Regress$(\mathcal{G}_{\text{F}}, \pi^*_{\text{F}})$, $\mathit{Del}(p) \subseteq S$.
(ii) Let $a\theta \in ([S]^{\mathcal{O}} \setminus \mathcal{A}^{\mathcal{O}}_{\text{L}}(s))$. There is some $p \in (pre_{a\theta} \setminus s)$ such that for all $a'\theta' \in \mathit{Add}(p)$, it holds that $a'\theta' \in S$ or θ' joins $del_{a'}$ with some $q \in pre_{a\theta}$.
(iii) Let $a\theta \in ([S]^{\mathcal{O}} \cap \mathcal{A}^{\mathcal{O}}_{\text{L}}(s))$. For every $p \in (pre_{a\theta} \cup add_{a\theta})$, $\mathit{Del}(p) \subseteq S$. Vice versa, for every $p \in del_{a\theta}$, $(\mathit{Add}(p) \cup \mathit{Pre}(p)) \subseteq S$.

That (i) – (iii) form sufficient conditions to Definition 1 follows from the definitions of *Pre*, *Add*, and *Del*. Consider (i). Suppose $a\theta \in \mathcal{A}^{\mathcal{O}}_{\text{L}}$ is a ground leader action such that $del_{a\theta} \cap$ Regress$(\mathcal{G}_{\text{F}}, \pi^*_{\text{F}}) \neq \emptyset$. Consider any $p \in (del_{a\theta} \cap$ Regress$(\mathcal{G}_{\text{F}}, \pi^*_{\text{F}}))$. As $p \in del_{a\theta}$, there must exist a minimal join θ^* of del_a with p that agrees with θ on all object assignments, i.e., $a\theta \in [a\theta^*]^{\mathcal{O}}$. By definition, $a\theta^* \in \mathit{Del}(p)$, so due to (i), $a\theta^* \in S$. In other words, $a\theta \in [S]^{\mathcal{O}}$, showing that $[S]^{\mathcal{O}}$ satisfies (1). With the same argument, the action-schema instances added in (ii) cover all ground actions enabling $a\theta$ on the selected ground atom p, and the ones added in (iii) cover the interfering ground actions.

Theorem 2. *If S is a lifted SSS, then $[S]^{\mathcal{O}}$ is an SSS.*

In the construction of S, conditions (ii) and (iii) process every ground action represented by S. Although the *Pre*/*Add*/*Del* parameter preselections confine the space of possible groundings, our experiments showed that this step can sometimes cause a severe overhead. We have explored an alternative, which attempts to reduce this overhead by considering in (ii) and (iii) possibly non-ground *specializations* $a\theta'$ of each $a\theta \in S$ rather than ground actions. The specializations that partition $a\theta$ into inapplicable/applicable instances can be computed efficiently by choosing parameters disconnecting/connecting the atoms $pre_{a\theta}$ and the state s. With appropriate extensions of the sets $\mathit{Pre}, \mathit{Add}, \mathit{Del}$ to arbitrary atoms, the remaining steps then apply unchanged. In our experiments, this variant, however, resulted in significantly less pruning. We stick to Definition 2 in the following.

Experimental Evaluation

Our implementation is based on the *PowerLifted (PL)* (Corrêa et al. 2020) lifted planner. We compare to Speicher et al.'s (2018a) LFS implementation in *Fast Downward (FD)* (Helmert 2006), which grounds Stackelberg tasks as a preprocess. By default, we enable all LFS-pruning methods, which Speicher et al. report to work best. For ablation study purposes, we additionally run our PL implementation without lifted-SSS pruning. Like Speicher et al., we consider optimal and satisficing planner configurations to solve the follower tasks. For FD, we reuse the same configurations: A* with LM-cut (Helmert and Domshlak 2009) for optimal, and GBFS with h^{FF} (Hoffmann and Nebel 2001) and preferred operators for satisficing planning. We mirror these configurations in PL, but due to unavailability, replace LM-cut by the weaker h^{max} heuristic (Haslum and Geffner 2000).

	Domain	#	FD-Grounding # grnd.	$\lvert\mathcal{A}^{\mathcal{O}}_{\mathrm{L}}\rvert$	$\lvert\mathcal{A}^{\mathcal{O}}_{\mathrm{F}}\rvert$	Coverage Optimal PL FD	S	$\neg S$	Satisficing PL FD	S	$\neg S$
A	Logist	290	290	91.7	447.1	**160**	0	0	**260**	248	235
	NoMyst	284	284	77.3	6.9k	**181**	0	0	**195**	82	81
	Pentest	643	426	80.7	72.8k	**423**	48	48	423	**563**	501
	Rovers	264	264	100.0	490.3	**104**	0	0	264	264	264
	TPP	282	282	91.6	449.4	**165**	2	2	**257**	252	248
	Transp	296	296	100.2	1.8k	**195**	42	40	**291**	287	285
	VisitAll	290	290	86.5	116.6	**204**	59	59	274	**278**	**278**
B	Logist	300	172	210.1k	211.5k	23	24	**31**	25	161	**288**
	Pipesw	50	12	12.2k	165.4k	0	**10**	**10**	1	21	**23**
	Rovers	360	144	8.9k	15.0k	**133**	103	101	142	**333**	**333**
	VisitAll	750	348	4	149.1k	105	143	**144**	153	**259**	**259**
C	Web	1240	295	104.6	242.9k	180	**408**	**408**	249	**1108**	1054

Table 1: Per-domain aggregated statistics. "#" total number of instances; "# grnd." instances successfully grounded by FD with "$|\mathcal{A}^{\mathcal{O}}_{\mathrm{L}}|$" and "$|\mathcal{A}^{\mathcal{O}}_{\mathrm{F}}|$" average number of generated ground leader/follower actions; and coverage results for optimal and satisficing Stackelberg planning: "FD" grounded LFS with all pruning methods, "PL" lifted LFS w/ "S"and w/o "$\neg S$" lifted-SSS pruning. Best results per track in bold.

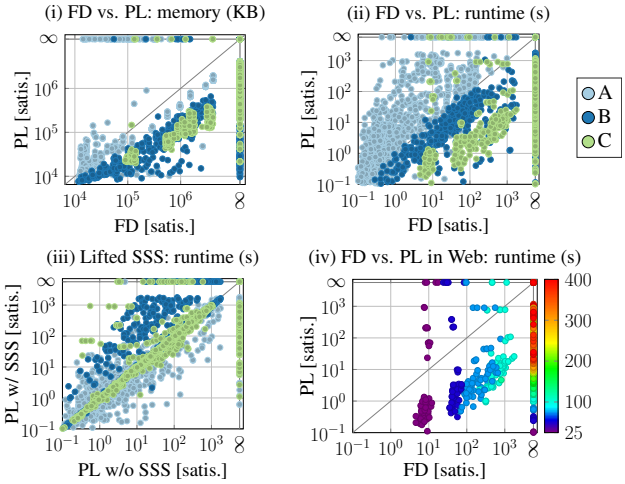

Figure 1: Per-instance comparisons of memory/runtime (i), (ii), (iv): between satisficing FD and PL using all pruning methods; and (iii) between satisficing PL w/ and w/o SSS pruning. Mark colors in (i)–(iii) distinguish the benchmark sets; (iv) the URL-number instance parameter. "∞" out of time or memory.

We experiment with three Stackelberg-PDDL benchmark sets: (A) that by Speicher et al. (2018a) (mostly easy to ground); (B) variants of Lauer et al.'s (2021) hard-to-ground (HTG) classical-planning benchmarks with Speicher et al.'s (2018a) Stackelberg adaptations, but ignoring domains where the adaptations did not make sense. (C) Tizio et al.'s (2022) *web infrastructure security* model. We generated 1240 synthetic instances, systematically scaling the number of URLs and other entities.

All experiments were run on Intel Xeon E5-2650 v3 CPUs, with time and memory limits of 30 min and 4 GB. Source code and benchmarks are publicly available[1].

Table 1 and Figure 1 show the results. Consider coverage (Table 1). We make three observations: 1. for optimal planning, PL is completely outclassed by FD on the (A) benchmarks, while it has slight edge on (B) and a considerable one on (C); 2. for satisficing planning, PL is generally competitive with FD, and starts to outclass FD the harder grounding becomes; and 3. the effect of lifted-SSS pruning is mixed.

Regarding 1., the performance difference on (A) is not surprising, given that the instances are mostly easy to ground, while PL uses a much weaker optimal follower search configuration. PL was in many cases not even able to solve the initial follower task. On the other hand, many instances of parts (B) and (C) are infeasable for FD's grounding method, some of which, however, remain comparatively easy to solve, and this is where PL's optimal configuration can shine. Regarding 2., the inadmissible but much stronger satisficing follower search configuration significantly boosts the performance of PL. On (A), PL performs marginally worse than FD, the more expensive lifted search taking its toll, but the difference is significant only in NoMyst. On the other hand, PL yields the much better performance when grounding becomes a factor. Notably, the latter is the case in the Pentest domain of Speicher et al.'s (2018a) original benchmark set (A). On parts (B) and (C), PL vastly outperforms FD. Peak memory usage and runtime (displayed in Figure 1 (i) respectively (ii) and (iv)) reflect these observations. Regarding 3., attributed to the weak baseline performance, SSS pruning shows no notable effect on optimal PL. For satisficing PL, SSS pruning is beneficial in parts (A) and (C), with significant coverage wins in Pentest and Web. The mentioned grounding overhead becomes apparent on (B), where the lifted-SSS construction often leads to runtime slowdowns of several orders of magnitude (cf., Figure 1 (iii)). That overhead grows with the number of ground leader actions. The latter is largest in Logist and Pipesw, where the overhead also manifests in coverage.

[1]https://doi.org/10.5281/zenodo.7701541

Conclusion

Stackelberg planning constitutes a versatile framework that enjoys wide applicability in security. As a remedy to the inability of existing, grounding-based, approaches of dealing with increasingly large instances, we introduced lifted Stackelberg planning: a lifted variant of the leader-follower search algorithm and suitable adaptations of the branch-and-bound pruning and partial-order reduction methods. Our experiments demonstrated the supremacy of lifted Stackelberg planning on large benchmark instances. Notably, our SSS adaptations are not restricted to Stackelberg planning. Exploring lifted-SSS pruning in lifted classical planning is an interesting line of future work. Regarding lifted Stackelberg planning, a promising way to further improve the LFS algorithm is Torralba et al.'s (2021) information-sharing scheme.

Acknowledgments

This work was supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0245, and by the German Research Foundation (DFG) under grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science).

References

Boddy, M.; Gohde, J.; Haigh, T.; and Harp, S. 2005. Course of Action Generation for Cyber Security Using Classical Planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, 12–21. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, 80–89. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*, 94–102. AAAI Press.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 140–149. AAAI Press, Menlo Park.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Hoffmann, J. 2015. Simulated Penetration Testing: From "Dijkstra" to "Turing Test++". In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 364–372. 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.

Lauer, P.; Torralba, Á.; Fiser, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. 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)*, 4119–4126.

Penberthy, J. S.; and Weld, D. S. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, 103–114. Morgan Kaufmann.

Ridder, B.; and Fox, M. 2014. Heuristic Evaluation based on Lifted Relaxed Planning Graphs. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 244–252. AAAI Press.

Russell, S.; and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall. ISBN 0-13-103805-2.

Sauer, P.; Steinmetz, M.; Künnemann, R.; and Hoffmann, J. 2023. Lifted Stackelberg Planning (ICAPS'23): Technical Report, Source Code, and Benchmarks. https://doi.org/10.5281/zenodo.7701541.

Speicher, P.; Steinmetz, M.; Backes, M.; Hoffmann, J.; and Künnemann, R. 2018a. Stackelberg Planning: Towards Effective Leader-Follower State Space Search. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, 6286–6293. AAAI Press.

Speicher, P.; Steinmetz, M.; Künnemann, R.; Simeonovski, M.; Pellegrino, G.; Hoffmann, J.; and Backes, M. 2018b. Formally Reasoning about the Cost and Efficacy of Securing the Email Infrastructure. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P'18)*, 77–91.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press. ISBN 978-1-10-709642-4.

Tizio, G. D.; Speicher, P.; Simeonovski, M.; Backes, M.; Stock, B.; and Künnemann, R. 2022. Pareto-Optimal Defenses for the Web Infrastructure: Theory and Practice. *ACM Transactions on Privacy and Security*.

Torralba, Á.; Speicher, P.; Künnemann, R.; Steinmetz, M.; and Hoffmann, J. 2021. Faster Stackelberg Planning via Symbolic Search and Information Sharing. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, 11998–12006. AAAI Press.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*, 491–515.

Wehrle, M.; and Helmert, M. 2014. Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 323–331. AAAI Press.

Younes, H. L. S.; and Simmons, R. G. 2003. VHPOP: Versatile Heuristic Partial Order Planner. *Journal of Artificial Intelligence Research*, 20: 405–430.