Purely Declarative Action Representations are Overrated: Classical Planning with Simulators

Guillem Francès[1], Miquel Ramírez[2], Nir Lipovetzky[2] and Hector Geffner[1,3]
[1]Universitat Pompeu Fabra, Spain; [2]University of Melbourne, Australia; [3]ICREA
{guillem.frances,hector.geffner}@upf.edu, {miguel.ramirez,nir.lipovetzky}@unimelb.edu.au

Abstract

Classical planning is concerned with problems where a goal needs to be reached from a known initial state by doing actions with deterministic, known effects. Classical planners, however, deal only with classical problems that can be expressed in declarative planning languages such as STRIPS or PDDL. This prevents their use on problems that are not easy to model declaratively or whose dynamics are given via simulations. Simulators do not provide a declarative representation of actions, but simply return successor states. The question we address in this paper is: can a planner that has access to the structure of states and goals only, approach the performance of planners that also have access to the structure of actions expressed in PDDL? To answer this, we develop domain-independent, black box planning algorithms that completely ignore action structure, and show that they match the performance of state-of-the-art classical planners on the standard planning benchmarks. Effective black box algorithms open up new possibilities for modeling and for expressing control knowledge, which we also illustrate.

1 Introduction

Planning is a key component of human intelligence [Seligman *et al.*, 2016] and one of the key areas of artificial intelligence since its beginnings [Newell and Simon, 1963]. Planning is normally conceived as "thinking before acting", and in AI, as the model-based approach to intelligent behavior, where model predictions are used to figure out what to do next [Geffner and Bonet, 2013]. Models come in a variety of forms, from classical to temporal and POMDP models, but what is common to most planners is that the models are described compactly using declarative languages [Nilsson and Fikes, 1971; McDermott, 2000]. Planning languages serve two roles: on the one hand, they are *general* so that different types of problems and domains can be fed into planners, on the other, they reveal *problem structure* that can be exploited computationally. Indeed, the planners that address the simplest (classical) planning models and scale up best all exploit in some way the problem structure encoded in action preconditions, effects, and goals. This includes from the first means-end and partial-order planners [Newell and Simon, 1963; Tate, 1977; Nilsson, 1980] to the latest SAT, OBDD, and heuristic search planners [Kautz and Selman, 1996; Edelkamp and Kissmann, 2009; Richter and Westphal, 2010; Rintanen, 2012]. The focus on standard, declarative planning languages has resulted in a dramatic progress in scalability, but has a downside too: some problems are not easy to model, and the runtime of planners is often very sensitive to the encodings. Moreover, there are many simulators available, that in many cases describe classical planning models (given initial state, deterministic actions, and reachability goals), but classical planners cannot be used for planning in them. These include simulators for the Atari video games [Bellemare *et al.*, 2013], the games of the General Video Game competition [Perez-Liebana *et al.*, 2016], Minecraft [Johnson *et al.*, 2016], and Universe [OpenAI, 2016].

Domain-independent planners able to plan with simulators rather than planning languages would be a key addition to the planning toolbox while broadening the scope of planners. Such planners would be insensitive to the syntax of action descriptions, since they would not see such descriptions at all. Indeed, the result of applying an action to a state can be obtained from declarative descriptions or procedures, whatever is more convenient, reducing the challenge of modeling. The key question we pose is: is it possible to achieve the goal of *planning with simulations* while retaining the scalability of planners that make use of *factored action representations*? In other words, can a domain-independent planner that has access to the *structure of states and goals only*, approach the performance of planners that have also access to the *structure of actions*? To address this question, we develop planning algorithms that have no access to the structure of actions and compare them with state-of-the-art planners over standard planning benchmarks.

The paper is organized as follows. We review state models and classical planners, introduce factored state models, formulate and test the black box planning algorithms, and discuss the results and implications.

2 State Models and Classical Planners

A *classical planning model* $\mathcal{S} = \langle S, s_0, S_G, Act, A, f, c\rangle$ is made up of a set of states S, the initial state $s_0 \in S$, the set of goal states $S_G \subseteq S$, the set of actions Act, the subsets $A(s)$ of

actions applicable in state s, the transition function f, where $f(a, s)$ represents the state s' that results from doing action a in state s, and the cost function c, where $c(a, s)$ is the cost of applying a in s. The solution to a classical planning model $\mathcal{S}$, or *plan*, is a sequence of actions that maps the initial state into a goal state, i.e., $\pi = \langle a_0, \ldots, a_n \rangle$ is a plan if there is a sequence of states $s_0, \ldots, s_{n+1}$ such that $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$ for $i = 0, \ldots, n$, and $s_{n+1} \in S_G$. The cost of plan π is given by the sum of action costs $c(a_i, s_i)$, and a plan is optimal if there is no plan with smaller cost. The classical planning model is the vanilla planning model, and differs from other models like temporal, conformant, contingent or POMDP models in its lack of uncertainty and sensing, and its focus on plans as action sequences.

A *classical planner* is a program whose input is a compact representation of a classical planning model $\mathcal{S}$ and whose output is a plan. Compact representations are expressed in a declarative planning language such as STRIPS or PDDL [McDermott, 2000]. The planning language provides a way to represent problems over completely different domains and reveals problem structure. For example, heuristic estimators [McDermott, 1999; Bonet and Geffner, 2001b], "helpful" actions [Hoffmann and Nebel, 2001], and landmarks [Hoffmann *et al.*, 2004], all key techniques in modern planners such as LAMA [Richter and Westphal, 2010], are derived from the "delete-relaxation" of the compact problem representation but *cannot be derived from the underlying planning model alone*. The same holds for other planning techniques such as SAT, OBDD, and partial-order planning. Despite the PSPACE-completeness of the classical planning problem [Bylander, 1994], modern planners have been shown to scale up to problems that define huge state spaces.

3 Factored State Models and Simulators

Purely declarative planning languages are not the only way to represent classical planning models in compact form. Indeed, state models $\mathcal{S} = \langle S, s_0, S_G, Act, A, f, c \rangle$ can also be represented in general by using state variables and by encoding the functions $A(s)$, $f(a, s)$, and $c(a, s)$ as *black box* procedures. Such a representation of actions has been seldom used in domain-independent planning, and in what follows we address the question of whether effective, general planning methods can be developed for it.

For this, we define a *factored state model* as a tuple $\mathcal{F} = \langle V, D, s_0, G, Act, A, f, c \rangle$ where V is a set of variables X, D represents the domains D_X of the variables $X \in V$, s_0 is an assignment to the variables compatible with their domains, G is a set (conjunction) of goal conditions expressed as Boolean functions of V, Act is a set of actions, and $A : S \mapsto 2^{Act}$, $f : A \times S \mapsto S$, and $c : S \to \mathbb{R}$ are functions encoded as *procedures* that represent the set of actions applicable in each state, the state successor function, and the cost function. The tuple $\mathcal{F}$ provides a compact representation of the state model $\mathcal{S} = \langle S, s_0, S_G, Act, A, f, c \rangle$ where S is the set of variable assignments compatible with their domains, and S_G is the set of assignments satisfying each goal condition in G. The goal conditions in G do not have to be atoms of the form $X = x$, but can be *arbitrary* procedures mapping states into Booleans.

A problem in STRIPS is a tuple $P = \langle F, I, G, O \rangle$ where F is a set of atoms, I represents the set of atoms that are true initially, G the set (conjunction) of atoms to be made true, and O is the of operators characterized by three sets of atoms: the precondition list, the add list, and the delete list. A STRIPS problem can be easily converted into a factored state model $\mathcal{F} = \langle V, D, s_0, G, Act, A, f, c \rangle$. For this, $V = F$, all domains in D are Boolean, s_0 is determined by I, $Act = O$, $c(a, s) = 1$, and the action applicability and successor state functions A and f are derived in linear time from the information in action preconditions and effects. This simple and polynomial translation is not bidirectional.

In what follows, a *simulator* will be a factored state model $\mathcal{F} = \langle V, D, s_0, G, Act, A, f, c \rangle$ where the functions A, f and c are given by black box procedures. The focus on black box methods that plan with simulators does not mean that these functions cannot be described declaratively or through hybrids involving procedures, but rather that simulation-based planning algorithms make no assumption about the form of such descriptions.

4 Width-based Search

The methods developed for planning in simulated environments like ALE and GVG-AI are based mainly on blind and heuristic search, Monte Carlo Tree Search (MCTS), width-based search, and hybrids of these [Bellemare *et al.*, 2013; Perez-Liebana *et al.*, 2016; Soemers *et al.*, 2016]. Blind search methods can be applied directly to factored state models that just encode weighted directed graphs in compact form. Heuristic search methods, on the other hand, require heuristics, which are not easy to derive from simulations. MCTS methods potentially combine the benefits of blind and heuristic search: they can be used off-the-shelf as the former, but scale up better as value functions akin to heuristic functions are incrementally learned and used to guide the search. Usually, however, MCTS performs this bootstrapping slowly when rewards are sparse and there is no domain-dependent knowledge in the form of informed base policies. This explains why MCTS is not used in classical planning.

Width-based methods have been used both in simulated environments like ALE and GVG-AI and in classical planning [Lipovetzky and Geffner, 2012; Lipovetzky *et al.*, 2015; Geffner and Geffner, 2015; Shleyfman *et al.*, 2016; Jinnai and Fukunaga, 2017]. The most basic width-based search method is IW(k), a breadth-first procedure that prunes states with novelty greater than k, and which runs in time and space that are exponential in k (unlike breadth-first search, which is exponential in the number of variables). The novelty $w(s)$ of a state in the search is given by the size of the smallest subset (conjunction) Q of atoms $X = x$ such that Q is true in s (i.e., all atoms in Q are true in s) and false in all the states of the search generated before s. For repeated states s, $w(s)$ is set to $|V| + 1$. Interestingly, IW(k) with $k = 2$ has been shown to find plans over many of the planning benchmark domains when the goal features a single atom. In turn, the procedure Serialized IW (SIW), that calls IW(k) sequentially with $k = 1, 2, 3$, for achieving joint goals incrementally,

one by one, has been shown to be competitive with a simple baseline heuristic search planner guided by state-of-the-art heuristics [Lipovetzky and Geffner, 2012]. Unlike heuristic search planners, SIW can work directly with simulators, but its performance on classical benchmarks does not compare well with the best planners that use other techniques as well.

5 Classical Planning with BFWS(f)

Recently, state-of-the-art performance over classical benchmarks has been achieved by combining width and heuristic search [Lipovetzky and Geffner, 2017]. Best-first width-based search is a standard best-first search that uses novelty measures to rank the nodes in OPEN. The novelty $w(s)$ of a newly generated state s *given the functions* $h_1, \ldots, h_n$ is defined as the size of the smallest set (conjunction) of atoms that is true in s but false in all the states s' generated before s *that have the same* h_i *values*, i.e., such that $h_i(s) = h_i(s')$ for $i = 1, \ldots, n$. This novelty measure is also written as $w_{\langle h_1,\ldots,h_n\rangle}$. The best monolithic BFWS planner in [Lipovetzky and Geffner, 2017] is BFWS(f_5), which uses $w = w_{\langle \#g,\#r\rangle}$, where $\#g(s)$ counts how many atomic goals in G are not true in s, and $\#r(s)$ is a subtler counter that has to do with potential subgoals that are not explicit in G. More precisely, when a state s is generated that achieves more goals than its parent s_p, i.e., $\#g(s) < \#g(s_p)$, a set of atoms $R(s)$ is computed such that, for any descendant s' of s reached from s through states that do not achieve more goals than s, $\#r(s')$ is the number of atoms in $R(s)$ that are made true *at some point* in the way from s to s'. For example, if $R(s)$ is defined as a set of landmarks, then $\#r(s')$ would count the number of those landmarks in $R(s)$ achieved in the way from s to s', even if they do not hold in s' anymore. In BFWS(f_5), $R(s)$ is not defined as a set of landmarks but as the set of atoms determined by a *relaxed plan* $\pi(s)$ computed from s [Hoffmann and Nebel, 2001]; i.e., $R(s)$ contains the preconditions and positive effects of the actions in $\pi(s)$. Ties in BFWS(f_5) are broken using the $\#g$ counter first (nodes with minimum $\#g$ preferred), and the accumulated cost to the node, second (cheaper nodes preferred). The algorithm BFWS(f_5) solves as many problems of the 2014 Int. Planning Competition (IPC) as the best planners LAMA, Mercury [Katz and Hoffmann, 2014], and Jasper [Xie *et al.*, 2014]. The general algorithm BFWS(R) below is BFWS(f_5) but with different sets $R(s)$.

6 Simulation Planning with BFWS(R)

The BFWS(f_5) algorithm is a state-of-the-art method for classical planning, yet it cannot be applied to simulations. This is because the computation of the sets of atoms $R(s)$ used in the definition of the $\#r$ counter relies on the delete-relaxation of the problem, which is not available from simulations. In order to have an algorithm that can plan with the black box functions $A(s)$ and $f(a, s)$ (action costs are assumed to be 1), we generalize the BFWS(f_5) procedure into a *family* of search algorithms, called BFWS(R), that differ from BFWS(f_5) in the way that the sets of atoms $R(s)$ are defined and computed. Thus, BFWS(R) is a best-first search algorithm with a primary evaluation function given by novelty measures $w_{\langle \#r,\#g\rangle}$, breaking ties using the $\#g$ counter and accumulated costs. The BFWS(f_5) algorithm is BFWS(R) with $R(s)$ defined as the set of atoms in a relaxed plan computed from the state s, which we will denote as $R_X(s)$. Other methods for defining and computing $R(s)$ are explored below, most of which do not require declarative action descriptions. Moreover, we focus only on methods that define and compute the set R *once* from the initial state s_0 and then fix $R(s)$ to $R(s_0)$ for any other state s where the set R is required. This is because the computation of such sets, while polynomial, can be expensive.

The intuition behind the BFWS(R) search schema, where R is a fixed set of atoms precomputed from the initial state, is that R represents a set of "potential subgoals" that, along with the given goals G, partitions the search nodes into classes associated with different "subproblems"; namely, the set of nodes that satisfy the same subset of goals from G and have reached the same set of subgoals from R. In each subproblem, the aim is to reach another goal or another potential subgoal that can eventually lead to another goal, ideally, by expanding novelty-1 nodes only (of which there is a linear number). Yet, since the number of such subproblems is exponential in the sizes of G and R, subproblems with the same *number* of goals and subgoals $\#g$ and $\#r$ are merged together. This is indeed what the novelty measure $w_{\langle \#r,\#g\rangle}$ used in BFWS(R) does.[1] In particular, a novelty measure of 1 for a state s means that s is the first state in the search that makes some atom p true, among the states s' generated so far that belong to the same "subproblem" as s, namely, that have the same $\#g$ and $\#r$ counts. The result is that the total number of subproblems is given by $|G| \times |R|$, and hence the maximum number of states that can have novelty k is $|G| \times |R| \times |F|^k$, where $|F|$ is the number of problem atoms.

The choices of the sets R of atoms in the general algorithm BFWS(R) that we consider are all *fixed*, in the sense that each is computed *once* from the initial state s_0 as a *polynomial form of preprocessing* from the simulation, so that for any state s where the set $R(s)$ is required, $R(s) = R(s_0)$. The different sets R that we consider are:

- R_0 is the empty set. For this set, $\#r(s) = 0$ for all s,
- R_A is the set of all atoms; i.e., $R_A = F$,
- $R[k]$ is the set of atoms that are true in the states that are reachable from s_0 by running IW(k), $k \in \{1, 2\}$,
- R_G is a goal-oriented version of the $R[k]$ sets obtained from the plans for the individual goals computed by IW(1) and IW(2) from s_0, if any,
- R^*_G is a variant of R_G that skips the IW(2) computation when the number of ground actions is too large, and
- R_X is the union of the preconditions and positive effects of the actions in a *relaxed plan* computed from s_0. This set *cannot* be computed from simulations and is included only as a baseline.

The exact definition of R_G is as follows. The procedure IW(1) is run from s_0; if that single invocation of IW(1) finds

[1] A related discussion can be found in [Lipovetzky, 2014].

	PDDL Planners												Simulation Planner			
	FF*				LAMA-11				BFWS(f_5)				BFWS(R^*_G)			
	C	L	T	Exp	C	L	T	Exp	C	L	T	Exp	C	L	T	Exp
Barman (20)	11	195	1K	5M	19	219	20	42K	**20**	174	21	107K	9	226	369	1M
Caved. (20)	**7**	23	71	1M	**7**	23	117	1M	**7**	24	0	8K	**7**	24	3	44K
Childs.† (20)	**7**	65	391	278K	5	69	3	2K	2	50	372	406K	5	57	176	59K
CityCar (20)	11	39	3	11K	5	36	631	1M	5	29	153	222K	**19**	30	57	27K
Floort.† (20)	**2**	38	5	169K	**2**	39	21	246K	**2**	42	5	73K	0	-	-	-
GED† (20)	0	-	-	-	**20**	135	4	3K	18	126	29	17K	**20**	133	10	8K
Hiking (20)	**20**	58	3	26K	18	54	284	36K	16	51	150	176K	7	64	246	174K
Maint.† (20)	10	116	3	16K	0	-	-	-	**16**	86	38	86	**16**	85	94	1K
Openst.† (20)	0	-	-	-	**20**	892	31	892	**20**	840	262	65K	14	780	1K	123K
Parking (20)	13	97	433	18K	**20**	105	114	2K	**20**	105	205	3K	**20**	106	397	44K
Tetris (20)	10	61	412	76K	10	60	615	59K	11	70	152	9K	**20**	61	158	5K
Thought. (20)	10	72	24	190K	16	76	2	673	17	72	3	5K	**20**	70	19	8.0K
Transp. (20)	4	325	147	15K	12	233	58	6K	**20**	240	6	39K	**20**	234	141	43K
Visitall† (20)	0	-	-	-	**20**	4K	212	21K	**20**	3K	52	4K	**20**	3K	10	3K
Elevat.† (20)	0	-	-	-	**20**	229	145	19K	**20**	247	20	35K	18	334	516	51K
Nomyst. (20)	9	29	234	1M	11	28	0	929	**14**	30	3	43K	13	30	63	76K
Pegsol (20)	**20**	31	4	97K	**20**	33	2	18K	**20**	29	109	785K	**20**	29	5	123K
Scanal. (20)	**20**	50	19	595	**20**	42	25	508	**20**	40	5	342	**20**	40	6	429
Sokob. (20)	18	231	34	487K	**19**	240	122	817K	15	195	122	2M	14	178	322	5M
All (380)	*172*	*0.94*	*0.60*	*0.59*	*264*	*0.92*	*0.51*	*0.35*	***283***	*0.87*	*0.27*	*0.35*	*282*	*0.89*	*0.48*	*0.45*

Table 1: Performance of PDDL Planners vs. Best Simulation Planner. Coverage (C), avg. plan length (L), avg. runtime in sec. (T), and avg. number of expanded nodes (Exp). Averages computed over instances solved by all planners, except in domains marked with †, where there are less than three commonly-solved instances. Times are total and include preprocessing. Best coverage in bold. Last row shows aggregated coverage and, for L, T and Exp, the avg. of *normalized values* computed for each planner over all domains, except those marked with †, where an L, T, and Exp value x is normalized by dividing it by the maximum value of that attribute over all planners (i.e., lower is better).

plans that reach each one of the individual problem goals,[2] then R_G is set to the collection of atoms made true by such plans. If, on the contrary, IW(1) does not return a plan for some goal, then IW(2) is run from s_0. If IW(2) finds plans for each of the problem goals, R_G is set to the collection of atoms made true by such plans. Last, if neither IW(1) nor IW(2) reach each of the problem goals, R_G is set to R_A; i.e., the collection of all atoms. This definition takes advantage of the fact that the *width* of many standard domains when the goals are atomic is often 1 or 2, meaning that IW(1) or IW(2) will find plans for them in low polynomial time [Lipovetzky and Geffner, 2012]. The requirement that IW(1) and IW(2) reach *all* of the goals in order to consider the atoms appearing in state trajectories reaching a goal is there just to keep things simple. Indeed, one could consider the atoms in such trajectories even if, say, 10% of the goals are not reached. Also for simplicity, when each of the problem goals is reached by IW(1) or IW(2) through more than one plan, we collect in R_G the atoms made true by only the first plan found for each goal, discarding all the others, which in principle could also yield useful information.

The definition of R_G is reminiscent of the use of relaxed plans in PDDL planning. Indeed, the union of the plans found by IW(k) that reach each of the individual problem goals is a plan for a problem relaxation which is tighter than the standard delete-relaxation: whereas the relaxed plan for a goal $G_1 \wedge \ldots \wedge G_n$ in the delete-relaxation is the union of *relaxed* plans for each of the G_i goals, the relaxed plans computed by IW(k) are made up of *actual* plans for each G_i. The downside of this is that IW(k) will not deliver any individual plan if the width of the atomic goals G_i is higher than k, and that even IW(3), while polynomial, can be expensive [Lipovetzky and Geffner, 2012].

The set R^*_G is defined exactly as the set R_G except for problems involving too many ground actions ($> 40,000$), where running IW(2) becomes too expensive. For such problems, R^*_G is set to R_G when the IW(1) run reaches all of the problem goals, but when not, the IW(2) computation is skipped and R^*_G is set directly to the collection of all atoms R_A.

Notice that the first three R options, R_0, R_A and $R[k]$, are independent of the problem goal, while the last three, R_G, R^*_G, and R_X, are all goal-oriented, with R_X being the only option that assumes knowledge of action preconditions and effects. For the other R sets, BFWS(R) is a simulation-based planning method.

7 Experimental Results

The performance of the simulation-based planning algorithm BFWS(R) for different R's is shown in Tables 1 and 2. The first table compares the top-performing R, R^*_G, against two state-of-the-art PDDL planners, LAMA, and BFWS(f_5), and a version FF* of the FF planner [Hoffmann and Nebel, 2001] that is supported in the Fast Downward planner, where FF's heuristic is used to drive a best-first search with two

[2] Notice that finding plans that reach each of the goals individually is different than finding plans that reach all goals *jointly*.

	BFWS(R_X)				BFWS(R_0)				BFWS(R_A)				BFWS(R[1])				BFWS(R_G)			
	C	L	T	Exp	C	L	T	Exp	C	L	T	Exp	C	L	T	Exp	C	L	T	Exp
Barman†	6	161	759	7M	0	-	-	-	8	245	75	319K	**16**	271	109	342K	9	237	391	1M
Caved.	**7**	23	6	265K	**7**	23	8	141K	**7**	26	6	79K	**7**	23	6	91K	**7**	24	3	44K
Childs.†	0	-	-	-	0	-	-	-	7	54	364	141K	**8**	56	345	196K	5	57	181	59K
CityCar	4	28	86	184K	15	26	37	35K	5	28	253	488K	5	26	176	176K	**18**	27	57	28K
Floort.†	**1**	42	578	14M	0	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-
GED	**20**	148	39	269K	**20**	122	17	13K	**20**	126	5	8K	**20**	127	9	8K	**20**	133	28	8K
Hiking†	**16**	51	84	116K	2	42	12	142K	6	70	316	209K	3	44	1K	4M	7	76	525	241K
Maint.	14	83	293	582	**16**	83	122	1K	**16**	83	104	1K	**16**	83	99	1K	**16**	83	109	1K
Openst.†	**20**	839	112	63K	0	-	-	-	11	769	1K	124K	2	822	2K	140K	11	778	2K	120K
Parking	**20**	87	7	2K	**20**	102	441	51K	**20**	110	451	46K	**20**	110	480	46K	16	109	1K	46K
Tetris	17	93	145	261K	**20**	76	419	17K	**20**	78	230	10K	**20**	94	316	17K	**20**	84	307	11K
Thought.	17	84	8	172K	15	95	14	8K	**20**	87	71	69K	**20**	89	392	251K	**20**	86	187	46K
Transp.	14	248	21	63K	7	233	664	368K	17	224	51	39K	**20**	231	63	41K	7	290	613	59K
Visitall	**20**	3K	5	4K	**20**	3K	9	3K	**20**	3K	10	3K	**20**	3K	12	3K	**20**	3K	22	3K
Elevat.	18	175	19	27K	16	178	314	126K	15	282	148	46K	12	670	372	119K	**20**	261	348	30K
Nomyst.	**19**	24	0	4K	6	24	7	100K	10	26	14	124K	12	25	1	8K	13	24	3	16K
Pegsol	**20**	29	4	81K	**20**	30	3	100K	**20**	30	4	111K	**20**	30	4	111K	**20**	29	4	123K
Scanal.	**20**	40	1	358	**20**	40	6	432	**20**	40	6	430	**20**	40	5	430	**20**	40	7	429
Sokob.	**15**	186	106	3M	13	178	422	11M	12	184	183	4M	13	182	164	3M	13	191	233	4M
All	***268***	*0.89*	*0.36*	*0.56*	*217*	*0.87*	*0.62*	*0.66*	*254*	*0.91*	*0.53*	*0.58*	*254*	*0.95*	*0.56*	*0.56*	*262*	*0.92*	*0.68*	*0.44*

Table 2: Performance of BFWS(R) algorithms for different R sets. These are all simulation-based algorithms except for BFWS(R_X), that makes use of action structure and is included as a baseline. See caption of Table 1 for an explanation of the different entries.

queues, one restricted to the nodes obtained with helpful actions only [Helmert, 2006]. FF is a second-generation heuristic search planner that uses helpful actions, while LAMA is a third-generation heuristic search planner that uses a landmark heuristic as well. Together, FF and LAMA have been the top-performing classical planners for the last 15 years. The original version of FF could not be used, as many instances include cost information. For the experiments, however, actions costs are taken to be 1 so that plan cost is equal to plan length. The second table evaluates BFWS(R) for the different choices of R, including the baseline option, $R = R_X$, that relies on PDDL encodings for computing a relaxed plan once from the initial state.

Benchmarked problems include all instances from the last planning competition (IPC 2014), along with all instances from IPC 2011 domains that did not appear in IPC 2014, with the exception of *Parcprinter*, *Tidybot* and *Woodworking*, which produced parsing errors. There are thus a total of 19 domains, with 20 instances each, for a total of 380 instances. All planners and configurations in both tables have been run on AMD Opteron 6378@2.4Ghz CPUs with CPU-time and memory cutoffs of 1h and 16GB respectively. Tables report the standard measures: coverage (number of instances solved), and average plan lengths, (total) runtimes, and number of node expansions. Average times are in seconds rounded to the nearest integer. All averages are over the instances solved by all planners except when there are less than three such instances. Implementation details of the BFWS planners are given in the next Section; LAMA-11 and FF* are run using the latest version of Fast Downward [Helmert, 2006] available at the time of writing.

Table 1 shows that the best simulation-based BFWS(R) planner, BFWS(R^*_G), performs extremely well in spite of not using any information about action structure, when compared to the state-of-the-art PDDL planners. Indeed, while LAMA solves 264 of the 380 instances (69.5%), BFWS(R^*_G) solves 282 (74.2%). The PDDL planner BFWS(f_5) does only slightly better, solving 283 (74.5%). FF*, on the other hand, solves 172 instances (45.3%), which illustrates that the considered problems are challenging. The planners, however, do very differently in the different domains. For example, in the domains Barman, Hiking, and Openstacks, LAMA solves 19, 18, and 20 instances respectively, while BFWS(R^*_G) solves 9, 7, and 14. On the other hand, in domains such as CityCar, Maintenance, and Tetris, LAMA solves 5, 0, and 10 instances, while BFWS(R^*_G) solves 19, 16, and 20. Surprisingly, these numbers are better than those of BFWS(f_5) that solves 5, 16, and 11 instances, suggesting that relaxed plans are not helping that much in such domains. This seems to be confirmed by the other PDDL planner, R_X, shown in Table 2, that solves 4, 14, and 17 of the instances. In terms of average plan lengths and times, there is no clear pattern, although BFWS(f_5) is usually fastest, and along with LAMA-11, is the one that expands fewer of nodes. The averages of normalized plan lengths, times, and number of expanded nodes, displayed at the bottom of both tables, show that the simulation planner BFWS(R^*_G) is only a bit slower than LAMA, produces plans of similar quality and, somewhat surprisingly, does not expand too many more nodes on average, in spite of the strong exploratory nature of BFWS algorithms.

Table 2 evaluates BFWS(R) for different alternatives of R. While none of them is better than R^*_G in terms of coverage, they all do better than FF* and some approach the performance of LAMA. Except for R_X, they are all simulated algorithms that do not exploit action descriptions. In comparison with the simulation planners, the relaxed plan computed by

R_X appears to provide useful guidance in domains such as Hiking, Openstacks, and Nomystery, where simulation planners solve much less instances. In fact, in Nomystery R_X solves many more instances than LAMA and BFWS(f_5), 19 vs. 11 and 14 respectively, meaning that the computation of all the other relaxed plans does not pay off. Since R^*_G is R_G except in instances that have more than 40,000 ground actions, it is only on those instances where the performance of BFWS(R^*_G) and BFWS(R_G) differ. These instances are mainly in Parking and Transport where the latter solves 16 and 7 instances, while the former solves them all. The empty R set, R_0, is the weakest R, with 217 instances solved, well behind the other options and LAMA, although well ahead of FF*, which solves 172 instances.

The domains where simulated BFWS(R) algorithms struggle are of two types: problems where the atomic goals have a *high width*, and problems with *tightly constrained* plans where goals and subgoals are *not easy to serialize.* Examples of the former class of domains include Barman, Hiking, and Openstacks. Examples of the latter include Cavediving, Childsnack, and Floortile. This last class of domains, like any tight *constraint satisfaction problem* encoded as a planning problem, are hard also for heuristic PDDL planners like LAMA and BFWS(f_5), which nevertheless perform much better in the first class of problems due to the exploitation of action structure in the form of relaxed plans.

8 Planner

The BFWS(R) algorithms are all implemented in the Functional STRIPS planner `FS` [Francès and Geffner, 2015], on top of the `LAPKT` toolkit (`http://lapkt.org`) in an extension that allows the free use of procedures for providing the denotation of fixed function and predicate symbols appearing in goals and action preconditions and effects. This free use of procedures greatly expands the modeling capabilities of the language, but is not easy to accommodate when heuristics are supposed to be derived automatically. Nevertheless, when `FS` runs in *simulation mode* no attempt at deriving heuristics is made (the whole action structure being invisible to the planner), and thus fixed symbols with procedural denotations pose no particular challenge.

The use of simulation mode does however affect the complexity of computing novelty measures. In propositional mode, testing if the state generated by an action has novelty 1 can be done in constant time by checking whether each one of the (bounded number of) atoms added by the action is new. In simulation mode, this test is linear in the number of variables, as there is no information about action effects, and the value of all variables in the resulting state needs to be checked. In general, while novelty-i tests in propositional mode are exponential in $i-1$, in simulation mode they are exponential in i. For this reason, novelty measures greater than 1 are determined lazily; namely, only when no node with novelty 1 remains in the OPEN list of the search. Moreover, following [Lipovetzky and Geffner, 2017], only novelty-1 and novelty-2 measures are computed, and even the latter are skipped in the BFWS(R) search when the number of atoms is too large. The selected set R is computed in a preprocessing step.

```
action move(x1: location)
  prec alive(p) ∧ @valid_move(loc(p), x1)
  effs loc(p) := x1
       ¬pellet_at(x1)
       pellet_at(x1) ⟶ C := C + 1
       ∀g ∈ ghost [loc(g) := @move_G(loc(g), x1)]

goal alive(p), C = K
init alive(p), C = 0, pellet_at(l1), ...
def alive(p: pacman) ≡ ∀g : ghost loc(g) ≠ loc(p)
```

Figure 1: Fragment of Pacman encoding in Functional STRIPS where goal is to eat K food pellets. Functional (predicate) symbols $@f$ denote (Boolean) functions specified by external procedures.

9 New Possibilities for Modeling and Control

We have seen how a move from purely declarative planning languages to simulation languages that freely combine declarative representations with procedures does not entail a significant performance loss, as one can plan effectively by exploiting state and goal structure while ignoring action structure. We now illustrate what can be gained from this move by considering two other aspects: modeling and control knowledge.

9.1 Modeling

Many classical problems that require reaching a goal by applying deterministic actions from a known initial state cannot be modeled easily using the standard planners, sometimes because of limitations of planners, that do not always provide good support to complex language constructs [Ivankovic and Haslum, 2015], sometimes because of limitations of planning languages. Challenging domains include, for instance, so-called "pseudo-adversarial" problems involving adversaries that follow a known, deterministic strategy that is nonetheless difficult to express propositionally, and problems like Pong, where physical actions have complex ramifications, like a ball bouncing against walls. Many of these limitations, however, can be overcome *in one shot* through the combination of expressive modeling languages and black box planning algorithms. A modeling language like Functional STRIPS [Geffner, 2000] offers the best of the declarative and procedural worlds; namely, a declarative, first-order logical language for modeling, where the denotation of fixed function and predicate symbols can be given extensionally, through sets of atoms, or intensionally, by means of procedures. This distinction makes no difference to simulated planning algorithms like BFWS(R) that get to see only the black box action transition and applicability functions.

Fig. 1 shows a fragment of a possible Functional STRIPS encoding of Pacman. In Pacman, the agent has to collect a number of food pellets (dots) by moving around in a maze, while avoiding a number of deadly ghosts that chase him. Functional (predicate) symbols $@f$ denote (Boolean) functions specified by external procedures. In this case, the maze is implicitly encoded by the Boolean function *@valid_move*, and the deterministic strategy followed by the ghosts (move always to the adjacent grid position that is closest to the pacman according to the Manhattan-distance) is encoded by

the $@move_G$ procedure, where ghosts are assumed to know where the pacman is moving. For simplicity, "power pellets" are omitted. We have run different versions of BFWS(R) in these and other domains that are difficult or impossible to tackle with PDDL planners. These include a single-agent version of the *Pong* videogame, where the agent controls a paddle and aims at hitting a ball with billiard-like dynamics a certain number of times; *Trapping* [Ivankovic and Haslum, 2015], where the agent has to trap an opponent (a cat) that is always looking for the nearest exit; *Helping*, in which the agent guides the cat to an exit by turning lights on that the cat will follow if sufficiently close, and *Pursuit*, a predators-and-prey game. All problem encodings and results can be found at `https://github.com/aig-upf/2017-planning-with-simulators`. Interestingly, the GVG-AI competition problems encoded in VGDL, an elegant language for encoding grid-problems that is part declarative and part procedural [Schaul, 2013], should be easy to translate into Functional STRIPS, once dynamic object creation and non-determinism are excluded. Non-deterministic and probabilistic extensions of Functional STRIPS were implemented early in the `GPT` planner, that deals with MDPs and POMDPs [Bonet and Geffner, 2001a].

9.2 Control Knowledge: Features and BFWS(F)

Features offer a way to express domain-dependent knowledge in the context of width-based search algorithms like IW and BFWS. A feature ϕ is a Boolean function of the state which can be used as an extra atom in the computation of novelty. For example, in blocks-world problems, a Boolean feature $\phi_{x,y}$ can be defined for each goal atom $on(x, y)$, that is true in a state s when $clear(x)$ and $clear(y)$ are both true in s. By adding such a feature, BFWS(R_0), for example, decrements the number of nodes expanded to reach the goal by an average factor of 3. Such a reduction happens because some relevant states that are reached in the search with novelty 2, will have novelty 1 once the extra atoms are considered. This is indeed what conjunctive features do: they promote or give priority to states that achieve certain conjunctions of atoms. Yet features can be arbitrary Boolean functions of the state, not just conjunctions, and an interesting generalization can be obtained by combining *features* and explicit *priorities.*

Let a *ranked feature list* be a list $F = \langle F_1, \ldots, F_M \rangle$ where F_i is a set of Boolean features with priority i. Ranked feature lists provide an *abstraction* and *generalization* of the algorithms IW and BFWS, where the notion of novelty is decoupled from the problem atoms and the size of conjunctions. For this, let the F*-novelty* of a state s in the search, $w_F(s)$, be the minimum i such that for some feature ϕ in F_i, ϕ is true in s for the first time in the search, with the convention that $w_F(s)$ is $M + 1$ when there is no such feature.

The IW(1) algorithm can be seen as a breadth-first search where states with novelty $w_F(s) > 1$ are pruned, where $F = \langle F_1 \rangle$ and F_1 is given by the features $\phi_{\langle p \rangle}$, one for each problem atom, such that $\phi_{\langle p \rangle}(s)$ is true iff p is true in s. Similarly, IW(2) is a breadth-first search where states with novelty $w_F(s) > 2$ are pruned, where $F = \langle F_1, F_2 \rangle$, and F_2 contains a feature $\phi_{\langle p,q \rangle}$ for each pair of different atoms p and q, and $\phi_{\langle p,q \rangle}(s)$ is true iff p and q are both true in s.

More interestingly, BFWS(R) for $R = R_0$ (empty R) is a plain best-first algorithm guided by a novelty function w_F where the features are $F = \langle F_1, F_2 \rangle$, and F_1 and F_2 contain the features $\phi_{\langle p,i \rangle}$ for each atom p, and the features $\phi_{\langle p,q,i \rangle}$ for each pair p, q, where $1 \leq i \leq |G|$. The feature $\phi_{\langle p,i \rangle}$ is true in a state s when p is true in s and $\#g(s) = i$, while $\phi_{\langle p,q,i \rangle}$ is true in s when both p and q are true in s, and $\#g(s) = i$. Finally, the *general* BFWS(R) algorithm can be *approximated* when the features in F_1 and F_2 are defined as $\phi_{\langle p,i,j \rangle}$ and $\phi_{\langle p,q,i,j \rangle}$ where in both cases the new j component, $1 \leq j \leq |R|$, tests whether the number of atoms in R that are true in s, $\#r'(s)$, is equal to j. This is an approximation because the $\#r'$ counter is state-dependent, while the actual counter $\#r$ used in BFWS(R) is path-dependent (counts number of atoms from R reached in the way to s).

The algorithms IW(F) and BFWS(F) refer to the IW and BFWS algorithms where novelties are computed according to the ranked feature list F. The use of these lists not only provides a uniform way for understanding and programming width-search based algorithms, but also opens up the possibility of encoding control knowledge by playing with features and priorities, which could potentially be inferred from training data using machine learning algorithms.

10 Conclusions

We have developed a new class of black box planning algorithms that approach the performance of the best classical planners over the existing PDDL benchmarks. The black box or simulated algorithms have access to the structure of states (the variables), the initial state, and the set of goals, but have no access to the structure of actions; namely, action preconditions and effects. This suggests that the computational role of declarative planning languages such as PDDL may be overrated. Declarative languages, however, remain important from the point of view of modeling. First-order logical planning languages such as Functional STRIPS appear to provide the best of both worlds by allowing fixed function and predicate symbols to denote actual procedures. While this expressive power is not compatible with mainstream methods, it can be exploited by the methods proposed here. Effective black box planning methods can indeed produce a radical change in the scope and use of planners, and in the ways in which planning problems are modeled.

Acknowledgments

G. Francès is supported by the M. de Maeztu Programme (MDM-2015-0502) and H. Geffner by grant TIN2015-67959-P, both from MINECO, Spain. M. Ramirez and N. Lipovetzky have been partially funded by the Australian DST Group.

References

[Bellemare *et al.*, 2013] Marc Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 47:253–279, 2013.

[Bonet and Geffner, 2001a] Blai Bonet and Hector Geffner. GPT: A tool for planning with uncertainty and partial information. In *Proc. IJCAI Workshop on Planning with Uncertainty and Partial Information*, 2001.

[Bonet and Geffner, 2001b] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[Bylander, 1994] Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[Edelkamp and Kissmann, 2009] Stefan Edelkamp and Peter Kissmann. Optimal symbolic planning with action costs and preferences. In *Proc. IJCAI*, pages 1690–1695, 2009.

[Francès and Geffner, 2015] Guillem Francès and Hector Geffner. Modeling and computation in planning: Better heuristics for more expressive languages. In *Proc. ICAPS*, pages 70–78, 2015.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A concise introduction to models and methods for automated planning*. Morgan & Claypool Publishers, 2013.

[Geffner and Geffner, 2015] Tomás Geffner and Hector Geffner. Width-based planning for general video-game playing. In *Proc. AIIDE*, pages 23–29, 2015.

[Geffner, 2000] Hector Geffner. Functional STRIPS. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 187–205. Kluwer, 2000.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[Hoffmann and Nebel, 2001] Joerg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Hoffmann *et al.*, 2004] Joerg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

[Ivankovic and Haslum, 2015] Franc Ivankovic and Patrik Haslum. Optimal planning with axioms. In *Proc. IJCAI*, pages 1580–1586, 2015.

[Jinnai and Fukunaga, 2017] Yuu Jinnai and Alex Fukunaga. Learning to prune dominated action sequences in online black-box planning. In *Proc. AAAI*, pages 839–845, 2017.

[Johnson *et al.*, 2016] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo platform for AI experimentation. In *Proc. IJCAI*, pages 4246–4247, 2016.

[Katz and Hoffmann, 2014] Michael Katz and Joerg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *Proc. 8th Int'l Planning Competition*, 2014.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*, pages 1194–1201, 1996.

[Lipovetzky and Geffner, 2012] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *Proc. ECAI*, pages 540–545, 2012.

[Lipovetzky and Geffner, 2017] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proc. AAAI*, 2017.

[Lipovetzky *et al.*, 2015] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: Results on the atari video games. In *Proc. IJCAI*, pages 1610–1616, 2015.

[Lipovetzky, 2014] Nir Lipovetzky. *Structure and Inference in Classical Planning*. AI Access, 2014.

[McDermott, 1999] Drew McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[McDermott, 2000] Drew McDermott. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2):35–56, 2000.

[Newell and Simon, 1963] Allen Newell and Herbert Simon. GPS: a program that simulates human thought. In *Computers and Thought*, pages 279–293. McGraw Hill, 1963.

[Nilsson and Fikes, 1971] Nils Nilsson and Richard Fikes. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

[Nilsson, 1980] Nils Nilsson. *Principles of Artificial Intelligence*. Tioga Press, 1980.

[OpenAI, 2016] OpenAI. Universe software platform. `https://universe.openai.com/`, 2016.

[Perez-Liebana *et al.*, 2016] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M. Lucas, and Tom Schaul. General video game AI: Competition, challenges and opportunities. In *Proc. AAAI*, 2016.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:122–177, 2010.

[Rintanen, 2012] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[Schaul, 2013] Tom Schaul. A video game description language for model-based or interactive learning. In *Proc. IEEE-CIG*, pages 1–8, 2013.

[Seligman *et al.*, 2016] Martin Seligman, Peter Railton, Chandra Sripada, and Roy Baumeister. *Homo prospectus*. Oxford University Press, 2016.

[Shleyfman *et al.*, 2016] Alexander Shleyfman, Alexander Tuisov, and Carmel Domshlak. Blind search for atari-like online planning revisited. In *Proc. IJCAI*, pages 3251–3257, 2016.

[Soemers *et al.*, 2016] Dennis Soemers, Chiara Sironi, Torsten Schuster, and Mark Winands. Enhancements for real-time Monte-Carlo tree search in general video game playing. In *Proc. IEEE-CIG*, pages 436–443, 2016.

[Tate, 1977] Austin Tate. Generating project networks. In *Proc. IJCAI*, pages 888–893, 1977.

[Xie *et al.*, 2014] Fan Xie, Martin Müller, and Robert Holte. Jasper: the art of exploration in greedy best first search. In *Proc. 8th Int'l Planning Competition*, 2014.