Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Best-First Width Search: Exploration and Exploitation in Classical Planning

Nir Lipovetzky
The University of Melbourne
Melbourne, Australia
nir.lipovetzky@unimelb.edu.au

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
hector.geffner@upf.edu

Abstract

It has been shown recently that the performance of greedy best-first search (GBFS) for computing plans that are not necessarily optimal can be improved by adding forms of exploration when reaching heuristic plateaus: from random walks to local GBFS searches. In this work, we address this problem but using structural exploration methods resulting from the ideas of width-based search. Width-based methods seek novel states, are not goal oriented, and their power has been shown recently in the Atari and GVG-AI video-games. We show first that width-based exploration in GBFS is more effective than GBFS with local GBFS search (GBFS-LS), and then proceed to formulate a simple and general computational framework where standard goal-oriented search (exploitation) and width-based search (structural exploration) are combined to yield a search scheme, best-first width search, that is better than both and which results in classical planning algorithms that outperform the state-of-the-art planners.

Introduction

It is well known that action selection in reinforcement learning must balance exploration and exploitation for delivering optimal behavior (Sutton and Barto 1998). The same is true in probabilistic and non-deterministic planning when non-admissible heuristics are used, as then non-greedy actions must be considered as well (Bonet and Geffner 2012). The exploitation and exploration tradeoff is also at the heart of Monte-Carlo Tree Search (MTCS) algorithms like UCT (Kocsis and Szepesvári 2006), which are based on optimal methods developed for arm selection in multi-armed bandit problems (Auer, Cesa-Bianchi, and Fischer 2002).

In classical planning, exploration is not needed for optimality, as greedy best-first search (GBFS) and, indeed, any best-first algorithm (BFS), delivers optimal solutions when used in anytime mode; i.e., not stopping until completion (Hansen and Zhou 2007). GBFS is the complete search method of choice in planners such as FF, FD, and LAMA (Hoffmann and Nebel 2001; Helmert 2006; Richter and Westphal 2010). The problem of GBFS, however, is that it often runs into large plateaus where the search gets lost as no states with better heuristic values are found. Recently it has been shown that the performance of GBFS for computing plans that are not necessarily optimal can be improved by adding forms of exploration to get out of these plateaus (Xie, Nakhost, and Müller 2012; Xie et al. 2014; Xie, Müller, and Holte 2014).

In this work, we address the problem of heuristic plateaus in greedy best-first search but from a different perspective: rather than adding exploration to a greedy goal-directed search, we look at a very effective class of structural exploration methods that have been developed recently, and see how to extend them with a goal-directed search. The two perspectives, however, are closely related, and we will look into this relation too.

The exploration methods that we use are based on the ideas of width-based search (Lipovetzky and Geffner 2012). Width-based search algorithms were developed in the setting of classical planning to show that instances of many existing domains can be solved in low polynomial time when they feature atomic goals. The ideas have also been used to yield state-of-art results in classical planning over the standard instances (Lipovetzky and Geffner 2012), and more recently in the Atari games (Lipovetzky, Ramirez, and Geffner 2015; Shleyfman, Tuisov, and Domshlak 2016), and those of the General Video-Game AI competition (Geffner and Geffner 2015). Width-based methods are pure exploration methods that are not goal-oriented, and our goal is to integrate them with heuristic search methods to get the best of both. For this, we show first that GBFS with width-based exploration is more effective than GBFS with local GBFS search (Xie, Müller, and Holte 2014), and formulate then a simple but general computational framework where standard goal-oriented search (exploitation) and width-based search (exploration) are combined to yield a search scheme, that we call **best-first width-search**, that is better than both and results in classical planning algorithms that outperform the state-of-the-art planners.

The paper is organized as follows. We provide first some background on classical planning, and heuristic and width-based search, and look then at GBFS with width-based exploration, best-first width search, and ways to combine different best-first width searches, presenting in all cases relevant experimental results.

Background

The classical model for planning $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$ is made up of a finite set of states S, an initial state s_0, a set of goal states S_G, and actions $a \in A(s)$ that deterministically map one state s into another $s' = f(a, s)$, where $A(s)$ is the set of actions applicable in s. The solution to a classical planning model is a sequence of actions $a_0, \ldots, a_m$, called a plan, that generates a state sequence $s_0, s_1, \ldots, s_{m+1}$ such that $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{m+1} \in S_G$.

A classical planning problem P defines a classical state model $\mathcal{S}(P)$ in compact form through a set of variables in a planning language such as STRIPS. In the absence of explicit cost information, it is assumed that action costs are all 1 so that the cost of a plan is given by its length, and the optimal plans are the shortest ones. Planners that seek optimal plans are called optimal planners. We focus on satisficing planners which are aimed at computing good-quality plans fast. While optimal planners are compared in terms of time and coverage, the empirical evaluation of satisficing planners takes plan quality into account as well.

Best-First Search

Most computational approaches to satisficing planning seek a plan for P by searching for a path connecting the initial state with a goal state in the directed graph associated with the state model $\mathcal{S}(P)$ (Geffner and Bonet 2013). For guiding this search they use heuristics derived automatically from P, including the additive, relaxed planning graph, and landmark heuristics, that will be denoted as h_{add}, h_{ff}, and h_L respectively (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter, Helmert, and Westphal 2008). All these heuristics are based on a suitable problem simplification, that in STRIPS is called the delete-relaxation as it consists of the original problem but with the action delete lists set to empty. A delete-free STRIPS problem can be solved (non-optimally) in low polynomial time, and the additive and RPG heuristics encode the cost of solutions to such a relaxation from the seed state. The landmark heuristic, on the other hand, just counts the number of landmarks to be achieved. The landmarks represent explicit or implicit atomic subgoals of the problem, usually computed from the delete-relaxation as well (Hoffmann, Porteous, and Sebastia 2004).

In best-first search (BFS), a path in a graph is sought by sequentially expanding the best-node in the OPEN list (the search frontier) according to an evaluation function f (smaller values preferred). The process starts with a single node in OPEN representing the initial state, and terminates when the selected node represents a goal state. Greedy best-first search (GBFS) is a best-first search where the evaluation function f is given by the heuristic. The classical A* search is a BFS with a function f that adds up the heuristic h and the accumulated cost g (Pearl 1983; Edelkamp and Schroedl 2011). The plans returned by A* are provably optimal when the heuristic h doesn't overestimate costs. The plans found with GBFS are not optimal but are computed faster. In principle, any BFS can be used to compute optimal plans if used in anytime mode, i.e., if not terminated until the OPEN list gets empty while preserving the best plan found (Hansen and Zhou 2007)

While the first generation of heuristic-search planners was based on forms of BFS using one heuristic, more recent planners incorporate a number of enhancements, including helpful actions (Hoffmann and Nebel 2001), delayed evaluation, and multiple queues (Helmert 2006). Delayed evaluation is useful in problems with large branching factors. Helpful actions refer to applicable actions that are relevant to the relaxed plan computed as part of the RPG heuristic. Multiple queues allow the combination of heuristics without aggregating them into a single function. LAMA, for example, performs a BFS with 4 queues: two of the queues are ordered by h_{ff} (or h_{add}), and two by h_L. In addition, one queue for each heuristic is for nodes that result from helpful actions. An alternative way to combine heuristics $h_1, \ldots, h_n$ is *lexicographically:* preferring nodes that minimize h_1, and in case of ties, nodes that minimize h_2, and so on. We will denote the resulting heuristic or preferences as $\langle h_1, \ldots, h_n \rangle$, and will use them as the language for integrating goal-directed preferences (heuristics) with exploration-based preferences (width).

Width-Based Search

Width-based search algorithms operate over states that assign a value x to a finite number of variables X over finite and discrete domains. The simplest such algorithm is IW(1), which is a plain breadth-first search where newly generated states that do not make an atom $X = x$ true for the first time in the search are pruned. The algorithm IW(2) is similar except that a state s is pruned when there are no atoms $X = x$ and $Y = y$ such that the *pair* of atoms $\langle X = x, Y = y \rangle$ is true in s and false in all the states generated before s. More generally, the algorithm IW(k) is a normal breadth-first except that newly generated states s are pruned when their "novelty" is greater than k, where the *novelty* of s is i iff there is a tuple t of i atoms such that s is the first state in the search that makes all the atoms in t true, and no tuple of smaller size has this property (Lipovetzky and Geffner 2012).

From a theoretical point of view, IW(k) can solve *arbitrary* instances of many of the standard benchmark domains in low polynomial time, provided that the *goal is a single atom* (Lipovetzky and Geffner 2012). Such domains can be shown to have a small and bounded *width* w that does not depend on the instance size, which implies that they can be solved (optimally) by running IW(k) with $k = w$. Moreover, IW(k) runs in time and space that are exponential in k and not in the number of problem variables that grows with the instance size.

From a practical point of view, the algorithm Iterated Width (IW) that calls the procedures IW(1), IW(2), ... sequentially until finding a solution, has been used to solve the standard benchmark instances featuring *multiple (conjunctive) atomic goals*. For this, *Serialized IW* (SIW) calls IW sequentially: first for achieving one goal, then from the resulting state for achieving two goals, and so on. While SIW is a blind search procedure that is incomplete (it can get trapped into dead-ends), it performs much better than a

GBFS guided by the additive or RPG heuristics, computing in general better plans, in less time (Lipovetzky and Geffner 2012).

Lipovetzky and Geffner developed also a planner BFS(f) that made use of landmarks, heuristics, helpful actions and delayed evaluation, in combination with novelty-based preferences, whose performance matched the performance of LAMA (Lipovetzky and Geffner 2012). Our purpose in this work is to explore the synergies between heuristic and width-based search more systematically. By BFS(w), we will refer to a BFS search purely guided by novelty measures, breaking ties by accumulated cost g. It is complete like IW but more practical as it avoids the repeated work done in successive invocations of IW(1), IW(2), etc.

BFS with Width-Based Exploration

One of the problems of GBFS is the presence of *heuristic plateaus* where a large number of iterations of GBFS does not succeed in generating states with a lower heuristic value (Hoffmann 2005). One way to deal with this problem is through the use of multiple queues ordered by different heuristics as in LAMA, but even then large plateaus are bound to exist. Two methods that have been explored recently for dealing with heuristic plateaus are random walks (Xie, Nakhost, and Müller 2012), and local GBFS or GBFS-LS (Xie, Müller, and Holte 2014). In GBFS-LS, a local GBFS guided by the same heuristic is triggered when the global GBFS fails to improve the value of the heuristic after a STALL-SIZE number of iterations. The local GBFS is like the global GBFS except that it operates on a local queue, initialized with a selected node from the plateau. This local search finishes when a state is found with a lower heuristic value, or when LS-SIZE nodes have been expanded. At that point, the states in the local queue are moved to the global queue, whether the process succeeded or not. The local GBFS is triggered at most MAX-LOCAL-TRY times for any given plateau, resulting in a GBFS-LS search that improves the performance of the plain GBFS on many domains (Xie, Müller, and Holte 2014).

The local search in GBFS-LS, however, is very much blind, as the heuristic is of limited use for getting out of plateaus. Indeed, due to the FIFO strategy used, the local search tends to be a breadth-first search over states that have the same heuristic value. This prompts the first question that we address in this work: wouldn't this local blind exploration work better if carried out by a structural method like IW or BFS(w)?

We have evaluated this question experimentally using the same settings and parameters as those reported for GBFS-LS; namely, STALL-SIZE=1000, LS-SIZE=1000, and MAX-LOCAL-TRY=100. We just use BFS(w) for the local search instead of GBFS. We call the resulting algorithm GBFS-W to distinguish it from GBFS-LS.

Table 1 shows the results of the baseline GBFS in comparison with GBFS-LS and GBFS-W. The first four columns in the table use $h = h_{add}$, while the following four columns use $h = h_{\text{ff}}$. The domains and instances are from the 2014 planning competition (Vallati et al. 2015). For $h = h_{add}$, the number of instances solved (out of a total of 280) goes from 47 for the baseline GBFS, to 57 for GBFS-LS, and 78 for GBFS-W. For $h = h_{\text{ff}}$ the numbers are 55, 79, and 88. Thus, while GBFS-LS improves the baseline GBFS, GBFS-W improves GBFS-LS quite consistently. Table 1 shows other methods for combining goal-directed and exploratory search that yield better results and which we consider next.

In the implementation of GBFS-W as in other algorithms below, a simplification is made: novelty measures $w(s)$ are computed with 2 or 3 level precision only, meaning in the first case, that $w(s)$ is determined to be 1 or greater than 1, and in the second, that $w(s)$ is determined to be 1, 2, or greater than 2. The reason for this simplification is that determining that $w(s)$ is i is exponential in $i-1$, as all new tuples of size up to i may have to be considered. By default, novelty measures $w(s)$ are computed with 2-level precision only; namely, $w(s)$ is determined to be 1 or greater than 1. This includes their use in GBFS-W. This is informative enough even in problems where the width of individual goals is 2. The reason is that the novelty measures are used in GBFS-W and the other algorithms below for reducing the value of the heuristic and not for achieving the goals as in IW. In a a few cases, as indicated, we will appeal to novelty measures computed with 3-level precision as well.

Best-First Width-Search

The notion of novelty plays a secondary role in GBFS-W: just to get the GBFS search going when it gets stuck. There are however other ways for combining the goal-directed and the exploratory search. We focus next on their combination in the context of a *best-first search with lexicographic preferences*, where preferences arise from one or more goal-directed heuristics and novelty measures. We call this family of search algorithms *best-first width-search (BFWS)*. As we will see, in the best-performing BFWS variants, the novelty measures will appear first in the lexicographic ordering, thus playing the primary role in the search, with the goal-directed heuristics being used for tie breaking only.

GBFS-W can be approximated by BFWS with the (lexicographic) evaluation function $f = \langle h, w \rangle$ where h is the heuristic function and w the novelty measure function. For this evaluation function, the preferred states in OPEN are the ones with lowest h value, and among those, the ones that are most novel (i.e., smallest novelty measure). For this, however, we need to define and compute novelty in a slightly different way, taking the heuristic h, and more generally, multiple functions h_j, into account. A similar idea is used in the BFS(f) planner (Lipovetzky and Geffner 2012):

Definition 1 *The novelty $w(s)$ of a newly generated state s* given the functions $h_1, \ldots, h_m$ *is i iff there is a tuple (conjunction) of i atoms and no smaller tuple, that is true in s and false in all states s' generated before s* with the same function values*, i.e., with $h_1(s') = h_1(s)$, ..., and $h_m(s') = h_m(s)$.*

We write $w(s)$ as $w_{h_1,\ldots,h_m}(s)$ when we want to make explicit the use of the functions h_j in the definition of $w(s)$. According to this definition, a new state s has novelty $w_{h_1,\ldots,h_m}(s) = 1$, for example, if there is an atom p

that is true in s but false in all the states s' generated before s such that $h_j(s') = h_j(s)$ for all $1 \leq j \leq m$.

Provided with this definition, BFWS with evaluation function $f_1 = \langle h, w \rangle$, denoted as BFWS($f_1$), selects from OPEN the states s with lowest $h(s)$ value, and among those, the ones with smallest $w(s)$ value, where $w = w_h$. For efficiency, as discussed before, we don't compute the value $w(s)$ exactly. Instead, except when stated otherwise, we just determine whether $w(s)$ is 1 or greater than 1. As a result, a state s will not be preferred to a state s' when the two states have the same heuristic values and both states have novelty measures greater than 1.

BFWS(f_1) provides an approximation of a GBFS driven by h, with a local BFS(w) search for escaping plateaus; namely the GBFS-W algorithm above. It turns out however that this BFWS variant is not as good as GBFS-W. The reason is that when BFWS(f_1) reaches an h-plateau, the novelty-based exploration can give preference to at most $|F|$ states, where F stands for the set of atoms in the problem, as there can be at most $|F|$ states with novelty 1 for the same h value. In GBFS-W, on the other hand, there are at most $|F|$ preferred states in each local search, but multiple (MAX-LOCAL-TRY) local searches are triggered from the same plateau.

Remarkably, however, BFWS with evaluation function $f_2 = \langle w, h \rangle$, where the preference order between heuristics and novelties are reversed, performs much better than both BFWS(f_1) with $f_1 = \langle h, w \rangle$ and GBFS-W. The preferred states in $f_2 = \langle w, h \rangle$ are not picked among the ones with lowest h but among those with lowest novelty measure $w(s) = w_h(s)$, with the heuristic h being used as a tie breaker. As a result, BFWS(f_2) is *not greedy* and may expand nodes s' that do not have the min heuristic value in OPEN. The number of such non-greedy expansions however cannot exceed $|F| \times h_M$, where h_M is the maximum possible value of the heuristic (e.g., for $h = h_{\text{ff}}$, $h_M \leq |A|$, where A is the set of actions in the problem). The reason for this is that only states s with novelty 1 will be preferred to states s' with a smaller heuristic value and there cannot be more than $|F| \times h_M$ such states.

Table 1 shows the results of these and other algorithms. The table is organized into four sets of columns. The first set of columns is for plain GBFS, GBFS-LS, GBFS-W, and BFWS(f_2) for $f_2 = \langle w, h \rangle$, in all cases using $h = h_{add}$. The second set of columns is for the same algorithms but with $h = h_{\text{ff}}$. For h_{add}, the number of problems solved by the algorithms is 47, 57, 78, and 100 respectively; while for h_{ff}, it is 55, 79, 88, and 104. For both heuristics, the local search LS helps, but the local BFS(w) exploration in GBFS-W helps even more. Moreover, in both cases, the best results are achieved by BFWS(f_2) that is not greedy as it uses the novelty measures as the primary evaluation function in $f_2 = \langle w, h \rangle$, breaking ties with the heuristic h. Average times and plan qualities are shown in the bottom of the table. The first two sets of columns show that among the eight planners considered so far, BFWS(f_2) with $h = h_{\text{ff}}$ has the best coverage, and also the best average times and plan lengths.

The third set of columns in Table 1 shows the performance of two other planners. The first is a GBFS planner with evaluation function $f_3 = \langle h_L, h_{\text{ff}} \rangle$, where h_L is the landmark heuristic. This configuration results in 99 problems being solved. In this configuration, the landmark heuristic can be thought as dividing the problems into subproblems which are then solved with the h_{ff} heuristic. The second planner is BFWS(f_4) where the evaluation function f_4 is obtained from f_3 by just pushing in a novelty preference as the main function; namely $f_4 = \langle w, h_L, h_{\text{ff}} \rangle$ with $w = w_{h_L, h_{\text{ff}}}$. Interestingly, this small change increases the number of problems solved from 99 to 149.

The last column in Table 1 represents a different BFWS planner, which is the one that solves the highest number of problems: 192. As a reference, to be shown in a second table, LAMA solves 171. The evaluation function for this BFWS planner is $f_5 = \langle w, \#g \rangle$ where $\#g(s)$ tracks the number of top problem goals that are not true in s. The novelty measure $w(s)$ is computed given both this counter $\#g$ and a second counter $\#r(s)$, i.e., $w = w_{\#g, \#r}$. The second counter tracks the number of atoms in the last relaxed plan computed in the way to s that have been made true. Relaxed plans are computed *only* for states that decrease the $\#g$ count in relation to their parent, and for the initial state. The set of atoms F_π in a relaxed plan π are those that appear as preconditions or positive effects of actions a in π. If the last relaxed plan π in the way to s was computed in a state s', $\#r(s)$ stands for the number of atoms in F_π that are true in *some* state s'' between s' and s, including these two states (Lipovetzky and Geffner 2014).

In BFWS(f_5), novelty measures $w = w_{\#g, \#r}$ are computed with a 3-value precision (namely, $w(s)$ is 1, 2, or greater than 2), taking advantage that the two counters $\#g$ and $\#r$ are computationally cheap. The most preferred states s according to f_5 are the novelty 1 states that appear closest to the goal as measured by the number of top goals achieved. The total number of novelty 1 states is again polynomially bounded; indeed since $\#r(s) \leq |F|$, this number cannot exceed $|F|^2 \times |G|$, where F and G stand for the set of problem atoms and goals respectively. Similarly, the total number of novelty 2 states is bounded by $|F|^3 \times |G|$.

Notice that BFWS(f_5) with $f_5 = \langle w_{\#g, \#r}, \#g \rangle$ manages to outperform LAMA without appealing to many of the techniques that have been found essential for performance in recent years; namely, helpful actions, landmarks, delayed evaluation, and multiple queues or searches. The planner BFS(f) in (Lipovetzky and Geffner 2012), that combines width-based exploration with most of these techniques, solves more problems than LAMA, 177 vs. 171, but less than BFWS(f_5), as shown in Table 2.

Dual BFWS

The experiments above illustrate that it is better to use goal-heuristics to break ties in a novelty-driven BFS search, than using novelty-measures or other exploration forms to break ties in a greedy heuristic-driven search. We'll provide some intuitions for this below, and focus now on improving performance further by considering a dual BFWS algorithm. In this Dual BFWS planner, like in FF, a slow but incomplete search, the planner front-end, is followed if not successful,

	GBFS $\langle h_{add} \rangle$	GBFS-LS $\langle h_{add} \rangle$	GBFS-W $\langle h_{add} \rangle$	BFWS $\langle w, h_{add} \rangle$	GBFS $\langle h_{\text{ff}} \rangle$	GBFS-LS $\langle h_{\text{ff}} \rangle$	GBFS-W $\langle h_{\text{ff}} \rangle$	BFWS $\langle w, h_{\text{ff}} \rangle$	GBFS $f_3 = \langle h_L, h_{\text{ff}} \rangle$	BFWS $f_4 = \langle w, h_L, h_{\text{ff}} \rangle$	BFWS $f_5 = \langle w_{\langle \#g, \#r \rangle}, \#g \rangle$
Barman (20)	0	0	0	**16**	0	0	0	**10**	0	**15**	20
CaveDiving (20)	5	**6** (1.09)	**6 (0.22)**	**6** (1.22)	6	**6** (1.33)	**6** (1.34)	5 **(0.36)**	7	7 (1.08)	7
Childsnack (20)	0	0	0	0	0	0	0	0	3	**9 (0.33)**	2
CityCar (20)	0	3	**6**	**6**	0	0	4	**5**	1	**9** (1.50)	5
Floortile (20)	2	2 (0.99)	2 (1.00)	2 **(0.49)**	2	2 (1.18)	2 (1.18)	2 **(0.12)**	2	2 **(0.09)**	2
GED (20)	0	2	10	**16**	16	15 (0.60)	13 (0.47)	**18 (0.20)**	20	20 **(0.23)**	20
Hiking (20)	8	7 **(0.21)**	7 **(0.21)**	8 (1.73)	2	6 (0.61)	7 **(0.36)**	**9** (0.88)	2	**9 (0.42)**	8
Maintenance (20)	**16**	**16** (1.00)	**16** (1.00)	**16** (1.00)	11	**16 (0.21)**	**16 (0.21)**	11 (1.00)	**16**	**16** (1.00)	16
Openstacks (20)	0	0	0	0	5	0	0	5	4	4 (1.00)	20
Parking (20)	0	0	0	0	0	0	0	0	1	**2 (0.85)**	20
Tetris (20)	1	7 **(0.39)**	**9 (0.39)**	3 (2.43)	1	5 **(0.10)**	**9** (0.14)	1 (3.77)	9	**13 (0.07)**	15
Thoughtful (20)	13	6 (0.72)	9 (0.52)	**20 (0.30)**	12	9 (0.39)	11 (0.49)	**18 (0.27)**	12	**15 (0.09)**	17
Transport (20)	2	3 (0.26)	5 (0.28)	**6 (0.25)**	0	0	0	0	2	**8 (0.58)**	20
Visitall (20)	0	5	**8**	1	0	**20**	**20**	**20**	20	20 (1.00)	20
Total Coverage (280)	47	57	78	**100**	55	79	88	**104**	99	**149**	192
Average Time	138.43	119.62	**77.16**	98.50	82.85	98.56	95.28	**30.02**	275.84	**189.46**	228.36
Average Quality	**34.63**	72.23	47.43	34.79	30.67	55.04	68.83	**30.16**	**313.59**	314.86	362.33

Table 1: Coverage over IPC-8 benchmarks of GBFS, GBFS-LS, GBFS-W, and BFWS with different evaluation functions. First four columns use $h = h_{add}$, and second four $h = h_{\text{ff}}$. Heuristic and evaluation functions shown for each column. GBFS-W and GBFS-LS use the same parameters as Xie et al. Reduction factor in terms of generated states with respect to GBFS is reported in parenthesis, e.g. 0.07 implies the algorithm generated 7% of the states generated by GBFS. Average times in seconds and plan lengths shown for each of the four sets of columns, over problems solved by all planners in the corresponding set of columns. Averages for last column are higher as they include all problems solved by a single planner. Best coverage overall shown in red. Best coverage for each of the three sets of columns shown in bold. Algorithms implemented in LAPKT. Experiments performed on 2.40GHz Intel Processor; time and memory outs after 30 min or 8GB.

by a slower and complete search, the planner back-end. Dual BFWS turns out to outperform not only BFWS(f_5), and hence LAMA, but all the planners that participated in the competition, including Jasper, Mercury, and the winner Ibacop portfolio (Xie, Müller, and Holte ; Katz and Hoffmann 2014; Cenamor, De La Rosa, and Fernández 2014).

As a front end of Dual-BFWS, we use the BFWS(f_5) planner above with $f_5 = \langle w, \#g \rangle$ and $w = w_{\#g,\#r}$, but with *one difference* that makes the algorithm *incomplete:* states s with a novelty $w(s) > 1$ are pruned. Indeed, this incomplete algorithm runs in *polynomial* time and expands $|F|^2 \times |G|$ states in the worst case, which is the maximum number of states s that can have novelty $w(s) = 1$. This incomplete search, by itself, does not solve as many problems as the complete BFWS(f_5) planner (151 vs. 192), but it succeeds or fails very fast. Interestingly, FF's front-end algorithm EHC solves 71 problems only.

As the back-end of Dual-BFWS, we use an extension of BFWS(f_4) with $f_4 = \langle w, h_L, h_{\text{ff}} \rangle$, where the landmark and FF heuristics are used as tie breakers, in that order, for a novelty measure $w = w_{h_L,h_{\text{ff}}}$. The extension introduces delayed evaluation and a distinction between helpful and non-helpful actions. Actually, the evaluation function used is $f_6 = \langle w, help, h_L, w', h_{\text{ff}} \rangle$ where $help$ is a function that is 1 or 2 according to whether the action leading to the state is helpful or not, and w' is a second novelty measure. While w is computed given the two heuristics h_L and h_{ff}, w' is computed given h_{ff} only; i.e. $w' = w'_{h_{\text{ff}}}$. Also in both cases, the novelty measures are computed with a 3-level precision; thus, they can be 1, 2, or greater than 2.

Table 2 compares Dual-BFWS with the 2014 IPC planners LAMA, Jasper, Mercury, BFS(f), and the Ibacop2 portfolio. For reference the best BFWS planner from Table 1, BFWS(f_5), is also included. LAMA and Mercury do not solve as many problems as BFWS(f_5), but Jasper and Ibacop2 solve 193 and 198 problems each. The Dual-BFWS algorithm, that performs two searches in a row, solves 225 problems, with 151 of them being solved by the first incomplete search. The curves showing the number of problems solved as a function of time are displayed in Fig. 1. Dual-BWFS solves more problems than pruned BFWS(f_5) even for short time windows; those are problems where the incomplete search fails quickly that are then solved quickly by the following complete BFWS(f_6) search.

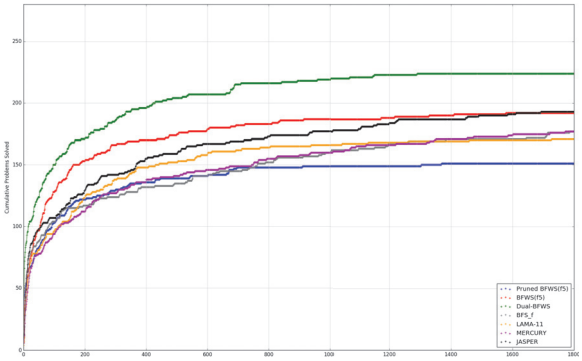

Figure 1: Coverage as a function of time for different planners. Top line is for Dual-BFWS. Bottom line is for its front end given by the pruned and polynomial BFWS(f_5) search.

	BFWS(f_5)		BFS(f)		LAMA-11		Mercury		Jasper		IBACOP2	Dual-BFWS	
	S	Q	S	Q	S	Q	S	Q	S	Q	S	S	Q
Barman (20)	**20**	178.83	**20**	160.89	19	209.89	19	250.83	**20**	297.00	**20**	**20**	**160.83**
CaveDiving (20)	7	27.29	**8**	23.57	7	**23.00**	7	**23.00**	**8**	24.86	7	**8**	23.29
Childsnack (20)	2	0.00	10	0.00	0	0.00	6	0.00	0	0.00	**20**	(1) 10	0.00
CityCar (20)	5	**22.67**	**20**	26.33	3	28.67	5	38.33	5	68.67	9	(1) **20**	26.33
Floortile (20)	2	39.50	3	40.00	2	**38.50**	2	40.50	2	45.00	**20**	2	41.00
GED (20)	**20**	114.00	13	127.69	**20**	116.85	**20**	**88.77**	**20**	122.62	**20**	**(20) 20**	119.92
Hiking (20)	8	45.80	12	46.40	16	56.80	12	79.20	**20**	99.20	**20**	11	**45.00**
Maintenance (20)	16	96.67	**17**	**85.67**	7	128.67	10	121.17	11	129.17	**17**	**(17) 17**	**85.67**
Openstacks (20)	**20**	663.83	6	680.50	**20**	684.67	18	667.33	**20**	683.67	6	**(20) 20**	**661.50**
Parking (20)	**20**	90.62	8	114.12	**20**	**79.38**	13	114.25	**20**	123.75	7	**(20) 20**	88.38
Tetris (20)	15	51.33	4	50.00	8	**43.33**	13	43.67	14	82.67	5	**(17) 17**	52.33
Thoughtful (20)	17	**78.45**	**20**	92.73	15	87.91	12	85.27	18	91.91	19	(15) **20**	80.91
Transport (20)	**20**	312.18	16	339.27	14	302.91	**20**	**232.82**	15	320.27	13	**(20) 20**	312.36
Visitall (20)	**20**	2916.85	**20**	**2859.70**	**20**	3628.60	**20**	2882.40	**20**	3834.35	15	**(20) 20**	2916.85
Overall (280)	192	331 (109s)	177	332 (291s)	171	388 (111s)	177	333 (151s)	193	423 (108s)	198	(151) **225**	**330 (49s)**

Table 2: Coverage (S) and Avg. Plan Quality (Q) of BFWS(f_5) and Dual-BFWS planners in relation to IPC-2014 planners and domains. Dual-BFWS involves two searches; number of problems solved by first incomplete search shown in parenthesis. Averages computed over problems solved by all planners except IBACOP2. Overall average time in seconds shown in parenthesis. Best coverage shown in red, best quality in bold. BFS(f), LAMA-11, Jasper, and Mercury are state-of-the-art algorithms. BFWS algorithms and BFS(f) implemented using LAPKT toolkit. Results for IBACOP2 portfolio taken from the IPC-8 report using an AMD 2.40GHz Processor. All experiments performed on 2.40GHz Intel Processor, with time and memory outs after 30 min and 8GB resp. as in last IPC.

Discussion

The reason that BFWS with evaluation function $f_2 = \langle w, h\rangle$ works better than with function $f_1 = \langle h, w\rangle$, where the order between the heuristic and the novelty function $w = w_h$ is swapped, is that the latter is too greedy: once a state s is generated with minimum heuristic value h_{min}, the search becomes focused on the successors of s with heuristic value equal to h_{min} or smaller, even if states with higher heuristic values need to be considered for leaving a plateau. The novelty function w in f_1 doesn't make the search less greedy, but just structures the search within the plateaus. On the other hand, the best first-search with evaluation function $f_2 = \langle w, h\rangle$ is not greedy, as it prefers a state s to a state s' in OPEN if $h(s) > h(s')$ and $w(s) < w(s')$. The result of this is that the search will not become focused on the first state that achieves a min heuristic value in OPEN. At the same time, smaller heuristic values are not ignored forever. First, heuristic values are used as tie breakers, and second and more importantly, heuristic values become the main drivers of the search when the OPEN list runs out of novelty 1 and 2 nodes, as all other nodes are then assumed to have the same novelty (actually, when $w(s)$ is computed with 2-level precision; only novelty 1 nodes are distinguished from the rest). Moreover, the total number of novelty 1 or novelty 2 nodes for $w = w_{h_1,h_2,\ldots,h_m}$ is polynomial for fixed m and functions h_i taking a polynomial number of values. So, while exploration plays the primary role in the search, the total amount of exploration remains polynomially bounded.

The way heuristics like h_L and h_{ff} are ordered, e.g., in BWFS with $f_5 = \langle w, h_L, h_{\text{ff}}\rangle$, takes advantage of the view of landmark heuristics as goal and subgoal serialization devices (Lipovetzky and Geffner 2011; 2012). In the ordering $f_4 = \langle h_L, h_{\text{ff}}\rangle$, the landmark heuristic tracks the number of subproblems that are yet unsolved, while the h_{ff} heuristic is used to solve them. GBFS(f_4) improves GBFS(h_{ff}) by the same ratio as the novelty exploration in BFWS(f_5) improves GBFS(f_4). The algorithm SIW, Serialized Iterated Width, also exploits goal serialization and novelty-based exploration, but its performance doesn't compete because it doesn't use heuristic estimators at all.

The Type-GBFS-LS algorithm used in Jasper (Xie, Müller, and Holte) uses a local search when the global search gets stuck, but uses Type-GBFS instead of GBFS for the local and global search levels. Type-GBFS (Xie et al. 2014) uses a dual multi-bucket data structure along with the standard queue sorted by the heuristic. Each bucket clusters together states with the same h_i values when more than one function is used. For example, h_1 can be the h_{ff} heuristic, while h_2 can the accummulated cost function g. Type-GBFS will then interleave expansions from the open queue and the buckets, randomly picking a bucket, and from that bucket, a node. These random selections result in non-greedy choices. BFWS using $f = \langle w, h\rangle$ achieves a similar behavior but without appealing to randomizations.

Conclusions

Heuristic search planning is the main computational approach in classical satisficing planning. The initial success of the approach rested on the derivation and use of heuristics for guiding the search for plans. Since then other ideas have been found crucial for performance and they belong to the palette of any planning user or researcher interested in performance and scalability. These include relaxed plans, helpful actions, landmark heuristics, delayed evaluation, and multiple search queues and architectures. In this work, we have tried to show that width-based exploration in the form

of simple novelty-based preferences or filters, provide an effective complement to goal-directed heuristics, open up new possibilities for structuring the search, and lead in fact to new state-of-the-art algorithms. One lesson of the work is that when lost in the search for the goal, it is useful not only to get away from the states that have been visited, but to move to states that are as different from those as much as possible, as revealed by the structure of the states, even if this means to move away from the goal. Best-first width search delivers this through the use of domain-independent novelty measures. We have shown that this combination of exploitation and structured exploration pays off computationally, yet this does not exclude that there may be other combinations that are more effective.

Acknowledgements. The authors thank M. Ramirez and G. Francès for useful comments. The work by N. Lipovetzky is partially supported by the Australian Research Council linkage grant LP11010015, H. Geffner is partially supported by grant TIN2015-67959-P, MEC, Spain.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2):235–256.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B., and Geffner, H. 2012. Action selection for MDPs: Anytime AO* vs. UCT. In *Proc. AAAI*, 1749–1755.

Cenamor, I.; De La Rosa, T.; and Fernández, F. 2014. IBACOP and IBACOP2 planner. In *Proc. IPC-8*.

Edelkamp, S., and Schroedl, S. 2011. *Heuristic search: theory and applications*. Elsevier.

Geffner, H., and Bonet, B. 2013. *A concise introduction to models and methods for automated planning*. Morgan & Claypool Publishers.

Geffner, T., and Geffner, H. 2015. Width-based planning for general video-game playing. In *Proc. AIIDE*.

Hansen, E., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research* 28:267–297.

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

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *Proc. of the 8th Int Planning Competition*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proc. ECML*, 282–293. Springer.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proc. ICAPS*, 154–161.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proc. ECAI*, 540–545.

Lipovetzky, N., and Geffner, H. 2014. Width-based algorithms for classical planning: New results. In *Proc. ECAI*, 88–90.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical planning with simulators: Results on the atari video games. In *Proc. IJCAI-2015*.

Pearl, J. 1983. *Heuristics*. Addison Wesley.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/. Accessed: 2016-09-15.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:122–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI*, 975–982.

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind search for atari-like online planning revisited. In *Proc. IJCAI*.

Sutton, R., and Barto, A. 1998. *Introduction to Reinforcement Learning*. MIT Press.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proc. AAAI*, 2395–2402.

Xie, F.; Müller, M.; and Holte, R. Jasper: the art of exploration in greedy best first search. In *Proc. IPC-8*.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding local exploration to greedy best-first search in satisficing planning. In *Proc. AAAI*, 2388–2394.

Xie, F.; Nakhost, H.; and Müller, M. 2012. Planning via random walk-driven local search. In *Proc. ICAPS*.