Width and Serialization of Classical Planning Problems

Nir Lipovetzky[1] and **Hector Geffner**[2]

Abstract. We introduce a width parameter that bounds the complexity of classical planning problems and domains, along with a simple but effective blind-search procedure that runs in time that is exponential in the problem width. We show that many benchmark domains have a bounded and small width provided that *goals* are restricted to *single atoms*, and hence that such problems are provably solvable in low polynomial time. We then focus on the practical value of these ideas over the existing benchmarks which feature conjunctive goals. We show that the blind-search procedure can be used for both serializing the goal into subgoals and for solving the resulting problems, resulting in a 'blind' planner that competes well with a best-first search planner guided by state-of-the-art heuristics. In addition, ideas like helpful actions and landmarks can be integrated as well, producing a planner with state-of-the-art performance.

1 INTRODUCTION

Various approaches have been developed for explaining the gap between the complexity of planning [5], and ability of current planners to solve most existing benchmarks in a few seconds [12, 19]. *Tractable planning* has been devoted to the identification of planning fragments that due to syntactic or structural restrictions can be solved in polynomial time; fragments that include for example problems with single atom preconditions and goals, among others [5, 2]. *Factored planning* has appealed instead to mappings of planning problems into Constraint Satisfaction Problems, and the notion of *width* over CSPs [1, 4]. The width of a CSP measures the number of variables that have to be collapsed to ensure that the graph underlying the CSP becomes a tree [8, 7]. The complexity of a CSP is exponential in the problem width. A notion of width for classical planning using a form of Hamming distance was introduced in [6], where the distance is set to the number of problem variables whose value needs to be changed in order to increase the number of achieved goals. These proposals, however, do not appear to explain the apparent simplicity of the standard domains.

A related thread of research has aimed at understanding the performance of modern heuristic search planners by analyzing the characteristics of the optimal delete-relaxation heuristic h^+ that planners approximate for guiding the search for plans [14, 11]. For instance, the lack of local minima for h^+ implies that the search for plans (and hence the global minimum of h^+) can be achieved by local search, and this local search is tractable when the distance to the states that decrement h^+ is bounded by a constant. This type of analysis has shed light on the characteristics of existing domains where heuristic search planning is easy, although it doesn't address explicitly the conditions under which the heuristic h^+ is easy to compute, nor whether it's the use of this heuristic that makes these domains easy.

[1] DTIC Universitat Pompeu Fabra, Spain, email: nir.lipovetzky@upf.edu
[2] ICREA & UPF, Spain, email: hector.geffner@upf.edu

The aim of this paper is to explore a new width notion for planning that can be useful both theoretically and practically. More precisely, the contributions of the paper are:

1. a new width notion for planning problems and domains;
2. a proof that many of the existing domains have a bounded and low width when goals are restricted to single atoms;
3. a simple, blind-search planning algorithm (*IW*) that runs in time exponential in the problem width;
4. a blind-search planner that uses *IW* for serializing a problem into subproblems and for solving the subproblems, which is competitive with a best-first search planner using state-of-the-art heuristics;
5. a state-of-the-art planner that integrates new ideas and old.

The organization of the paper follows this structure, preceded by a review of basic notions in planning.

2 PLANNING

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 the state s. The solution to a classical planning model is a sequence of actions $a_0, \ldots, a_m$ 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 model in compact form through a set of variables. We assume a Strips language $P = \langle F, I, O, G\rangle$, where F is the set of boolean variables, atoms, or fluents, I is the set of atoms characterizing the initial state, O is the set of actions, and G is the set of goal atoms. All definitions below extend easily to other planning languages provided that states are valuations over a set of variables.

We assume that *action costs* are all 1, so that plan cost is plan length, and the *optimal* plans are the *shortest* ones. We write $P(t)$ to denote the planning problem that is like P but with goal t. The cost of the goal t in P is the cost of an optimal plan for $P(t)$.

3 WIDTH

A state s is reachable from the initial state $s_0 = I$ in P, if there is a state trajectory $s_0, s_1, \ldots, s_n$, such that s_{i+1} is a successor of s_i for some action a_i, and $s = s_n$. Since this reachability relation is exponential in the number of atoms, we define a different reachability relation over *tuples (conjunctions) of atoms* t of bounded size. The key question is when a tuple t' can be regarded as a 'successor' of a tuple t in P. If this is taken to represent the presence of an action a in P such that the *regression* of t through a is in t', the reachability relation on tuples ends up being too weak. A stronger reachability

relation on tuples can be obtained if we assume that both t and t' are achieved *optimally*. We will define indeed that t' is a successor of t if *every optimal plan* for t can be extended into an *optimal plan* for t' by just adding one action. In this way, the 'side-effects' of the optimal plans for t can be used to achieve t' optimally from t. This is formalized below in terms of *tuple graphs*, where T^i stands for the collection of tuples from P with size no greater than a given positive integer i:

Definition 1 *For $P = \langle F, I, O, G\rangle$, $\mathcal{G}^i$ is the* graph *with vertices from T^i defined inductively as follows:*

1. *t is a root vertex in $\mathcal{G}^i$ iff t is true in I,*
2. *$t \rightarrow t'$ is a directed edge in $\mathcal{G}^i$ iff t is in $\mathcal{G}^i$ and for every optimal plan π for $P(t)$ there is an action $a \in O$ such that π followed by a is an optimal plan for $P(t')$.*

In other words, the presence of the tuple t' of at most i atoms in the graph $\mathcal{G}^i$ indicates that either t' is true in I or that there is another tuple t of at most i atoms in $\mathcal{G}^i$ such that *all* the optimal plans π for t yield optimal plans for t', once a suitable action a is appended to π.

The graph $\mathcal{G}^i$ is acyclic because an edge $t \rightarrow t'$ implies that the optimal cost for achieving t' is the optimal cost of achieving t plus 1. Since we are associating plan cost with plan length, this means also that a tuple at depth k in the graph has optimal cost k.

Let us now say that a goal formula G_1 *implies* goal formula G_2 *in a problem* P, if all the *optimal plans* for G_1 are also *optimal plans* for G_2. Notice that this is not the standard logical implication that requires the formula $G_1 \supset G_2$ to be true in all reachable states, and hence, an invariant. For example, the goal $G_1 = \{hold(b)\}$ implies the goal $G_2 = \{clear(a)\}$ in *Blocks World* if $on(b, a)$ is true initially, yet $hold(b) \supset clear(a)$ is not an invariant.

Provided with this notion of implication, we define the *width* of a planning problem P and, more generally, the *width* of an arbitrary goal formula ϕ relative to P as follows:

Definition 2 *For a formula ϕ over the fluents in P that is not true in the initial situation I, the* width *of ϕ relative to P is the min w such that $\mathcal{G}^w$ contains a tuple that implies ϕ. If ϕ is true in I, its width is 0.*

Definition 3 *The* width *of a planning problem P, $w(P)$, is the* width *of its goal G relative to P.*

As in the case of graphical models, the width of a problem give us a bound on the complexity of solving the problem:

Theorem 4 *If $w(P) = i$, P can be solved optimally in time that is exponential in i.*

Of course, the crucial question is whether there are interesting planning problems that have a bounded, and hopefully small width. The answer is yes. Indeed, *most domain benchmarks appear to have a small width independent of the size of the problems, provided that the goal G is restricted to a single atom*. Of course, this result doesn't settle the complexity of the existing benchmarks where goals are *not* single atoms, yet as far as we know, it's the first formal result that places the complexity of these benchmarks squarely on the *goal structure*, and not on the *domain structure*. That is, if a domain has a low width when goals are single atoms, then when a domain instance is not easy, it can only be due to conjunctive goals.

We state the result about the width of a few domains. For most other benchmark domains, the same result seems to hold, but we have not carried out the proofs. Later on, however, we will report experiments that bear on this issue.

Theorem 5 *The domains Blocks, Logistics, and n-puzzle have a* bounded width *independent of the problem size and initial situation, provided that the goals are restricted to* single atoms.

It turns out indeed that for single atom goals, the width of Blocks, Logistics and n-puzzle is at most 2. We omit the proofs for lack of space, but for illustration purposes we prove $w(G) = 1$ for any goal $G = ontable(b)$ in Blocks.

Clearly, if G is true in the initial situation, the tuple G will belong to the graph $\mathcal{G}^i$ for $i = 1$. Thus, assume that G is not true initially, and let $b_1, \ldots, b_{n-1}$ be the blocks on top of b, starting from the top, and let $b = b_n$. We can then just show that the path

$$clear(b_1), hold(b_1), ontable(b_1), hold(b_2), \ldots, ontable(b_n)$$

makes it into $\mathcal{G}^1$. For a path $t_0, t_1, \ldots, t_n$ to be in $\mathcal{G}^1$ when t_0 is true in the initial situation, we just need to show that any optimal plan for t_i can be extended by means of a single action into an optimal plan for t_{i+1}, $i = 0, \ldots, n-1$. This is trivial in this case, as the optimal plans for $hold(b_i)$ can always be extended with the action $putdown(b_i)$ into optimal plans for $ontable(b_i)$, while the *optimal plans* for $ontable(b_i)$ from the above initial situation can all be extended with the action $unstack(b_{i+1}, b_{i+2})$ into *optimal plans* for $hold(b_{i+1})$. It's important to notice that without the restriction to *optimal plans* this reasoning would not get through.

4 BASIC ALGORITHM: PRUNED BREADTH-FIRST

We turn now to the planning algorithm that achieves the complexity bounds expressed by Theorem 4. The algorithm, called *Iterated Width* search or *IW*, consists of a sequence of calls *IW*(i) for $i = 0, 1, 2, \ldots$ over a problem P until the problem is solved. Each iteration *IW*(i) is an i-width search that is complete for problems whose width is bounded by i and has complexity is $O(n^i)$, where n is the number of problem variables. If P is solvable and its width is w, *IW* will solve P in at most $w + 1$ iterations with a complexity $O(n^w)$. *IW*(i) is *a plain forward-state breadth-first search* with just one change: right after a state s is generated, the state is pruned if it doesn't pass a simple *novelty test* that depends on i.

Definition 6 *A newly generated state s* produces a new tuple of atoms t *iff s is the first state generated in the search that makes t true. The size of the smallest new tuple of atoms produced by s is called the* novelty *of s. When s does not generate a new tuple, it's novelty is set to $n + 1$ where n is the number of problem variables.*

In other words, if s is the first state generated in all the search that makes an atom p true, its novelty is 1. If s does not generate a new atom but generates a new pair (p, q), its novelty is 2, and so on. Likewise, if s does not generate a new tuple at all because the same state has been generated before, then its novelty is set to $n+1$. The higher the novelty measure, the less novel the state. The iterations *IW*(i) are plain *breadth-first searches* that treat newly generated states with novelty measure greater than i as if they were 'duplicate' states:

Definition 7 *IW(i) is a breadth-first search that prunes newly generated states when their novelty measure is greater than i.*

Table 1. Effective width of single goal instances obtained from existing benchmarks by splitting problems with N atomic goals into N problems with single goals. I is number of resulting instances. The other columns show percentage of instances with effective width 1, 2, or greater.

Domain	I	$w_e = 1$	$w_e = 2$	$w_e > 2$
8puzzle	400	55%	45%	0%
Barman	232	9%	0%	91%
Blocks World	598	26%	74%	0%
Cybersecure	86	65%	0%	35%
Depots	189	11%	66%	23%
Driver	259	45%	55%	0%
Elevators	510	0%	100%	0%
Ferry	650	36%	64%	0%
Floortile	538	96%	4%	0%
Freecell	76	8%	92%	0%
Grid	19	5%	84%	11%
Gripper	1275	0%	100%	0%
Logistics	249	18%	82%	0%
Miconic	650	0%	100%	0%
Mprime	43	5%	95%	0%
Mystery	30	7%	93%	0%
NoMystery	210	0%	100%	0%
OpenStacks	630	0%	0%	100%
OpenStacksIPC6	1230	5%	16%	79%
ParcPrinter	975	85%	15%	0%
Parking	540	77%	23%	0%
Pegsol	964	92%	8%	0%
Pipes-NonTan	259	44%	56%	0%
Pipes-Tan	369	59%	37%	3%
PSRsmall	316	92%	0%	8%
Rovers	488	47%	53%	0%
Satellite	308	11%	89%	0%
Scanalyzer	624	100%	0%	0%
Sokoban	153	37%	36%	27%
Storage	240	100%	0%	0%
Tidybot	84	12%	39%	49%
Tpp	315	0%	92%	8%
Transport	330	0%	100%	0%
Trucks	345	0%	100%	0%
Visitall	21859	100%	0%	0%
Woodworking	1659	100%	0%	0%
Zeno	219	21%	79%	0%
Summary	37921	37.0%	51.3%	11.7%

Notice that *IW*(n), when n is the number of atoms in the problem, just prunes truly duplicate states and it is therefore complete. On the other hand, *IW*(i) for lower i values prunes many states and is not. Indeed, the number of states *not* pruned in *IW*(1) is $O(n)$ and similarly, the number of states not pruned in *IW*(i) is $O(n^i)$. Likewise, since the novelty of a state is never 0, *IW*(0) prunes all the children states of the initial state s_0, and thus *IW*(0) solves P iff the goal is true in the initial situation. The resulting planning algorithm *IW* is just a series of i-width searches *IW*(i), for increasing values of i:

Definition 8 *Iterated Width (IW) calls IW*(i) *sequentially for* $i = 0, 1, 2, \ldots$ *until the problem is solved or* i *exceeds the number of problem variables.*

Iterated Width (*IW*) is thus a *blind-search algorithm* similar to Iterative Deepening (*ID*) except for two differences. First, each iteration is a pruned *depth-first* search in *ID*, and a pruned *breadth-first* search in *IW*. Second, each iteration increases pruning depth in *ID*, and pruning width or novelty in *IW*.

From the considerations above it is straightforward to show that *IW* like *ID* is *sound* and *complete*. On the other hand, while *IW*(w) is *optimal* for a problem P of width w, *IW* is not necessarily so. The reason is that *IW* may solve P in an iteration *IW*(i) for i smaller than w.[3]

[3] As an illustration, the goal G of width 2 is achieved non-optimally by *IW*(1) when $I = \{p_1, q_1\}$ and the actions are $a_i : p_i \rightarrow p_{i+1}$ and $b_i : q_i \rightarrow q_{i+1}$ for $i = 1, \ldots, 5$, along with $b : p_6 \rightarrow G$ and $c : p_3, q_3 \rightarrow G$. Indeed, *IW*$(2)$ achieves G optimally at cost 5 using the action c, yet this action is never applied in *IW*(1), where states that result from applying the actions a_i when q_j is true for $j > 1$ are pruned, and states that result from applying the actions b_i when p_j is true for $j > 1$ are pruned too. As a result, *IW*(1) prunes the states with pairs such as (p_3, q_2) and (p_2, q_3), and does not generate states with the pair (p_3, q_3), which are required for reaching G optimally. *IW*(1) however reaches G at the non-optimal cost 7 using the action b.

Table 2. Blind-search algorithm *IW* compared with two other blind-search algorithms: Iterative Deepening (*ID*) and Breadth-First Search (*BrFS*). Numbers report coverage over benchmark domains with single atomic goals. Also included for comparison the figure for heuristic Greedy Best First Search (*GBFS*) with h_{add}.

# Instances	*IW*	*ID*	*BrFS*	*GBFS* + h_{add}
37921	34627	9010	8762	34849

Nonetheless the completeness and optimality of *IW*(w) for problems with width w provides the right complexity bound for *IW*:

Theorem 9 *For solvable problems* P*, the time and space complexity of IW are exponential in* $w(P)$*.*

It's important to realize that this bound is achieved without knowing the actual width of P. This follows from the result below, whose proof we omit for lack of space:

Theorem 10 *For a solvable problem* P *with width* w*, IW*(w) *solves* P *optimally in time exponential in* w*.*

The algorithm *IW*(w) is guaranteed to solve P if $w(P) = w$, yet as discussed above, the algorithm *IW* does not assume that this width is known and thus makes the *IW*(i) calls in order starting from $i = 0$. We refer to the min value of i for which *IW*(i) solves P as the *effective width* of P, $w_e(P)$, which is never higher than the real width $w(P)$.

The effective width $w_e(P)$ provides an approximation of the actual width $w(P)$. While proving formally that most benchmark domains have bounded width *for single atom goals* is tedious, we have run the algorithm *IW* to compute the *effective width* of such goals. The results are shown in Table 1. We tested domains from previous IPCs. For each instance with N goal atoms, we created N instances with a single goal, and run *IW* over each one of them. The total number of instances is 37921. For each domain we show the total number of single goal instances, and the percentage of instances that have effective widths w_e equal to 1, 2, or greater than 2. The last row in the table shows the average percentage over all domains: 37% with $w_e = 1$, 51% with $w_e = 2$, and less than 12% with $w_e > 2$. *That is, on average, less than* 12% *of the instances have effective width greater than* 2. Actually, in most domains *all* the instances have effective width at most 2, and in four domains, all the instances have effective width 1. The instances with a majority of atomic goals with an effective width greater than 2 are from the domains Barman, Openstacks, and Tidybot (the first and last from the 2011 IPC).

Iterated Width (*IW*) is a complete blind-search algorithm like Iterative Deepening (*ID*) and Breadth-First Search (*BrFS*). We have also tested the three algorithms over the set of 37921 single goal instances above. The results are shown in Table 2 . With memory and time limits of 2GB and 2h, *ID* and *BrFS* solve less than 25% of the instances, while *IW* solves more than 90%, which is almost as many as a Greedy Best First Search guided by the additive heuristic (also shown in the Table). The result suggests that *IW* manages to exploit the low width of these problems much better than the other blind-search algorithms. We will see below that a simple extension suffices to make *IW* competitive with a *heuristic* planner over the standard benchmark instances that feature *joint goals*.

5 SERIALIZATION

The fact that single goal atoms can be achieved quite effectively in most benchmarks domains by a pruned breadth-first search that does not look at the goal in any way, suggests that the complexity of benchmarks comes from conjunctive goals. Indeed, this has been the intuition in the field of planning since its beginnings where goal decomposition was deemed as a crucial and characteristic technique. The analysis above formalizes this intuition by showing that the effective width of single atom goals in existing benchmarks is low. This old intuition also suggests that the power of planners that can handle single goals efficiently can be exploited for conjunctive goals through some form of decomposition.

Serialized Iterated Width is a search algorithm that uses the iterated width searches both for constructing a serialization of the problem $P = \langle F, I, O, G\rangle$ and for solving the resulting subproblems. While *IW* is a sequence of i-width searches *IW*(i), $i = 0, 1, \ldots$ over the same problem P, *SIW* is a sequence of *IW* calls over $|G|$ subproblems P_k, $k = 1, \ldots, |G|$. The definition of *SIW* takes advantage of the fact that *IW* is a blind-search procedure that doesn't need to know the goals of the problem in advance; it just needs to recognize them in order to stop. Thanks to this feature *IW* is used both for decomposing P into the sequence of subproblems P_k and for solving each one of them. The plan for P is the concatenation of the plans obtained for the subproblems.

Definition 11 *Serialized Iterated Width (SIW) over* $P = \langle F, I, O, G\rangle$ *consists of a sequence of calls to IW over the problems* $P_k = \langle F, I_k, O, G_k\rangle$, $k = 1, \ldots, |G|$, *where*

1. $I_1 = I$,
2. G_k *is the first* consistent *set of atoms achieved from* I_k *such that* $G_{k-1} \subset G_k \subseteq G$ *and* $|G_k| = k$; $G_0 = \emptyset$
3. I_{k+1} *represents the state where* G_k *is achieved,* $1 < k < |G|$.

In other words, the k-th subcall of *SIW* stops when *IW* generates a state s_k that consistently achieves k goals from G: those achieved in the previous subcall and a new goal from G. The same is required from the next subcall that starts at s_k. The state s_k *consistently* achieves $G_k \subseteq G$ if s_k achieves G_k, and G_k does not need to be undone in order to achieve G. This last condition is checked by testing whether $h_{max}(s_k) = \infty$ is true in P once the actions that delete atoms from G_k are excluded [3]. Notice that *SIW* does not use heuristic estimators to the goal, and does not even know what goal G_k is when *IW* is invoked on subproblem P_k: it finds this out when *IW* generates a set of atoms G' such that $G_{k-1} \subset G' \subseteq G$ and $|G'| = k$. It then sets G_k to G'. This is how *SIW* manages to use *IW* for both constructing the serialization and solving the subproblems.

The *SIW* algorithm is sound and the solution to P can be obtained by concatenating the solutions to the problems $P_1, \ldots, P_m$, where $m = |G|$. Like *IW*, however, *SIW* does not guarantee optimality. Likewise, while the *IW* algorithm is complete, *SIW* is not. The reason is that the subgoal mechanism implicit in *SIW* commits to intermediate states from which the goal may not be reachable. Of course, if there are no dead-ends in the problem, *SIW* is complete.

We have compared experimentally the blind-search algorithm *SIW* to a baseline heuristic search planner using a Greedy Best First Search (GBFS) and the additive heuristic [3]. Neither planner is state-of-the-art, as neither uses key techniques such as helpful actions or landmarks [12, 19]. Still, the comparison shows that the non-goal oriented form of pruning in *IW* and the simple form of decomposition in *SIW* are quite powerful; as powerful indeed, as the best heuristic estimators.

Table 3. Blind-Search *SIW* vs. Heuristic GBFS over real benchmarks (with joint goals). *I* is number of instances, *S* is number of solved instances, *Q* is average plan length, *T* is average time in seconds. *M/A* w_e stand for max and avg effective width per domain. Shown in bold are the numbers *S*, *Q*, or *T* that one planner improves over the other by more than 10%.

		Serialized *IW* (*SIW*)				GBFS + h_{add}		
Domain	I	S	Q	T	M/A w_e	S	Q	T
8puzzle	50	50	**42.34**	0.64	4/1.75	50	55.94	**0.07**
Barman	20	1	–	–	–	–	–	–
Blocks World	50	50	**48.32**	5.05	3/1.22	50	122.96	**3.50**
Cybersecure	30	–	–	–	–	–	–	–
Depots	22	**21**	**34.55**	**22.32**	3/1.74	11	104.55	121.24
Driver	20	**16**	28.21	2.76	3/1.31	14	**26.86**	**0.30**
Elevators	30	**27**	**55.00**	**13.90**	2/2.00	16	101.50	210.50
Ferry	50	50	**27.40**	**0.02**	2/1.98	50	32.88	0.03
Floortile	20	–	–	–	–	–	–	–
Freecell	20	**19**	**47.50**	**7.53**	2/1.62	17	62.88	68.25
Grid	5	**5**	**36.00**	**22.66**	3/2.12	3	195.67	320.65
Gripper	50	50	101.00	3.03	2/2.00	50	**99.04**	**0.36**
Logistics	28	28	**54.25**	2.61	2/2.00	28	56.25	**0.33**
Miconic	50	50	**42.44**	0.08	2/2.00	50	42.72	**0.01**
Mprime	35	27	**6.65**	**84.80**	2/2.00	28	17.92	204.76
Mystery	30	27	**6.47**	42.89	2/1.19	28	7.60	**15.44**
NoMystery	20	–	–	–	–	**6**	–	–
OpenStacks	30	**13**	**105.23**	**0.53**	3/1.80	7	112.42	6.49
OpenStacksIPC6	30	26	**29.43**	108.27	4/1.48	**30**	32.14	**23.86**
ParcPrinter	30	9	16.00	0.06	3/1.28	**30**	**15.67**	**0.01**
Parking	20	17	**39.50**	**38.84**	2/1.14	2	68.00	686.72
Pegsol	30	6	**16.00**	1.71	4/1.09	**30**	16.17	**0.06**
Pipes-NonTan	50	**45**	**26.36**	**3.23**	3/1.62	25	113.84	68.42
Pipes-Tan	50	**35**	**26.00**	205.21	3/1.63	14	33.57	**134.21**
PSRsmall	50	25	**13.79**	28.37	4/2.27	**44**	18.04	**4.99**
Rovers	40	**27**	**38.47**	**108.59**	2/1.39	20	67.63	148.34
Satellite	20	19	38.63	216.69	2/1.29	20	**34.11**	**8.44**
Scanalyzer	30	26	**26.81**	**33.96**	2/1.16	28	28.50	129.42
Sokoban	30	3	**80.67**	**7.83**	3/2.58	**23**	166.67	14.30
Storage	30	**25**	**12.62**	**0.06**	2/1.48	16	29.56	8.52
Tidybot	20	7	**42.00**	532.27	3/1.81	**16**	70.29	**184.77**
Tpp	30	24	**82.95**	**68.32**	3/2.03	23	116.45	199.51
Transport	30	**21**	**54.53**	94.61	2/2.00	17	70.82	**70.05**
Trucks	30	2	**31.00**	**4.58**	2/2.00	**8**	34.50	14.08
Visitall	20	**19**	**199.00**	**0.91**	1/1.00	3	2485.00	174.87
Woodworking	30	**30**	**21.50**	**6.26**	2/1.07	12	42.50	81.02
Zeno	20	19	**34.89**	166.84	2/1.83	20	35.11	**101.06**
Summary	1150	819	**44.4**	**55.01**	2.5/1.6	789	137.0	91.05

SIW and GBFS are both written in C++ and use Metric-FF as an *ADL* to *Propositional STRIPS* compiler [10]. The experiments were conducted on a dual-processor running at 2.33 GHz and 2 GB of RAM. Processes time or memory out after 30 minutes or 2 GB. The results are summarized in Table 3 . Out of 1150 instances, *SIW* solves 30 problems more, it's usually faster, and produces shorter solutions.

Table 3 shows also the highest and average effective widths of the subproblems that result from the serializations generated by *SIW*. The maximal effective width is 4, which occurs in two domains: *8puzzle* and *PSRsmall*. On average, however, the effective width is between 1 and 2, except for four domains with effective widths between 2 and 3: Sokoban (2.58), PSRsmall (2.27), Grid (2.12), and TPP (2.03).

6 STATE-OF-THE-ART PERFORMANCE

While the blind-search *SIW* procedure competes well with a greedy best-first search planner using the additive heuristic, neither planner is state-of-the-art. Since state-of-the-art performance is important in classical planning, we show next that it is possible to deliver such performance by integrating the idea of novelty that arises from width considerations, with known techniques such as helpful actions, landmarks, and heuristics. For this we switch to a *plain forward-search best-first search planner* guided by an evaluation function $f(n)$ over the nodes n given by

$$f(n) = novelha(n) \tag{1}$$

where $novelha(n)$ is a measure that combines novelty and helpful actions, as defined below. In addition, ties are broken lexicographi-

Table 4. BFS(f) vs. LAMA, FF, and PROBE. *I* is number of instances, *S* is number of solved instances, *Q* is average plan length, *T* is average time in seconds. *T* and *Q* averages computed over problems solved by all planners except FF, excluded because of the large gap in coverage. Numbers in bold show performance that is at least 10% better than the other planners.

		BFS(f)			PROBE			LAMA'11			FF		
Domain	I	S	Q	T	S	Q	T	S	Q	T	S	Q	T
8puzzle	50	50	**45.45**	0.20	50	61.45	**0.09**	49	58.24	0.18	49	52.61	0.03
Barman	20	20	174.45	281.28	20	169.30	12.93	20	203.85	**8.39**	–	–	–
Blocks World	50	50	54.24	2.40	50	**43.88**	**0.23**	50	88.92	0.41	44	39.36	66.67
Cybersecure	30	28	39.23	70.14	24	52.85	69.22	30	37.54	576.69	4	29.50	0.73
Depots	22	22	49.68	56.93	22	44.95	**5.46**	21	61.95	46.66	22	51.82	32.72
Driver	20	18	48.06	57.37	20	60.17	1.05	20	46.22	**0.94**	16	25.00	14.52
Elevators	30	30	129.13	93.88	30	107.97	26.66	30	**96.40**	**4.69**	30	85.73	1.00
Ferry	50	50	32.94	0.03	50	29.34	**0.02**	50	28.18	0.31	50	27.68	0.02
Floortile	20	**7**	43.50	**29.52**	5	45.25	71.33	5	49.75	95.54	5	44.20	134.29
Freecell	20	20	64.39	**13.00**	20	62.44	41.26	19	68.94	27.34	20	64.00	22.95
Grid	5	5	70.60	7.70	5	**58.00**	9.64	5	70.60	**4.84**	5	61.00	0.27
Gripper	50	50	101.00	0.37	50	101.00	**0.06**	50	**76.00**	0.36	50	76.00	0.03
Logistics	28	28	56.71	0.12	28	55.36	**0.09**	28	**43.32**	0.35	28	41.43	0.03
Miconic	50	50	34.46	0.01	50	44.80	**0.01**	50	**30.84**	0.28	50	30.38	0.03
Mprime	35	35	10.74	19.75	35	14.37	28.72	35	**9.09**	**10.98**	34	9.53	14.82
Mystery	30	27	7.07	**0.92**	25	7.71	1.08	22	7.29	1.70	18	6.61	0.24
NoMystery	20	**19**	24.33	**1.09**	5	25.17	5.47	11	24.67	2.66	4	19.75	0.23
OpenStacks	30	29	141.40	129.05	30	137.90	64.55	30	142.93	**3.49**	30	155.67	6.86
OpenStacksIPC6	30	30	125.89	40.19	30	134.14	48.89	30	130.18	**4.91**	30	136.17	0.38
ParcPrinter	30	27	35.92	6.48	28	36.40	0.26	30	37.72	0.28	30	42.73	0.06
Parking	20	17	90.46	577.30	17	146.08	693.12	**19**	87.23	**363.89**	3	88.33	945.86
Pegsol	30	30	24.20	**1.17**	30	25.17	8.60	30	25.90	2.76	30	25.50	7.61
Pipes-NonTan	50	47	**39.09**	35.97	45	46.73	**3.18**	44	57.59	11.10	35	34.34	12.77
Pipes-Tan	50	40	**40.48**	254.62	43	55.40	102.29	41	48.60	**58.44**	20	31.45	87.96
PSRsmall	50	48	22.15	2.62	50	21.40	**0.08**	50	**18.31**	0.36	42	16.71	63.05
Rovers	40	40	105.08	44.19	40	109.97	24.19	40	108.28	**17.90**	40	100.47	31.78
Satellite	20	20	36.05	1.26	20	37.05	0.84	20	40.75	0.78	20	37.75	0.10
Scanalyzer	30	27	29.37	7.40	28	25.15	**5.59**	28	27.52	8.14	30	31.87	70.74
Sokoban	30	23	220.57	125.12	25	233.48	39.63	**28**	213.00	58.24	26	213.38	26.61
Storage	30	20	20.94	4.34	21	**14.56**	**0.07**	18	24.33	8.15	18	16.28	39.17
Tidybot	20	18	62.94	198.22	**19**	**53.50**	35.33	16	62.31	113.00	15	63.20	9.78
Tpp	30	30	112.33	36.51	30	155.63	58.98	30	119.13	**18.12**	28	122.29	53.23
Transport	30	30	107.70	55.04	30	137.17	44.72	30	108.03	94.11	29	117.41	167.10
Trucks	30	15	26.50	8.59	8	26.75	113.54	16	24.12	**0.53**	11	27.09	3.84
Visitall	20	20	**947.67**	84.67	19	1185.67	308.42	20	1285.56	77.80	6	450.67	38.22
Woodworking	30	30	41.13	19.12	30	41.13	15.93	30	51.57	**12.45**	17	32.35	0.22
Zeno	20	20	37.70	77.56	20	44.90	6.18	20	35.80	**4.28**	20	30.60	0.17
Summary	1150	1070	87.93	63.36	1052	98.71	49.94	1065	98.67	44.35	909	67.75	51.50

cally by two other measures: first, $usg(n)$, that counts the number of subgoals not yet achieved up to n, and second, $h_{add}(n)$, that is the additive heuristic.

The subgoals are the problem landmarks [13] derived using a standard polynomial algorithm over the delete-relaxation [20, 15]. The count $usg(n)$ is similar to the landmark heuristic in LAMA [18], simplified a bit: we use only atomic landmarks (no disjunctions), sound orderings, and count a top goal p as achieved when goals q that must be established before q have been achieved [16].

The $novelha(n)$ measure combines the novelty of n and whether the action leading to n is helpful or not [12]. The novelty of n is defined as the size of the smallest tuple t of atoms that is true in n and false in *all previously generated nodes* n' *in the search with the same number of unachieved goals* $usg(n') = usg(n)$. Basically, nodes n and n' in the search with different number of unachieved goals, $usg(n) \neq usg(n')$, are treated as being about different subproblems, and are not compared for determining their novelty. The novelty of a node $novel(n)$ is computed approximately, being set to 3 when it's neither 1 nor 2. Similarly, if $help(n)$ is set to 1 or 2 according to whether the action leading to n was helpful or not, then $novelha(n)$ is set to a number between 1 and 6 defined as

$$novelha(n) = 2[novel(n) - 1] + help(n) \qquad (2)$$

That is, $novelha(n)$ is 1 if the novelty of n is 1 and the action leading to n is helpful, 2 if the novelty is 1 and the action is not helpful, 3 if the novelty is 2 and the action is helpful, and so on. Basically, novel states (lower $novel(n)$ measure) are preferred to less novel states, and helpful actions are preferred to non-helpful, with the former criterion carrying more weight. Once again, the criterion is simple and follows from performance considerations.

We call the resulting best-first search planner, BFS(f), and compare it with three state-of-the-art planners: FF, LAMA, and PROBE [12, 18, 16].[4] Like LAMA, BFS(f) uses delayed evaluation, a technique that is useful for problems with large branching factors [17].

Table 4 compares the four planners over the 1150 instances. In terms of coverage, BFS(f) solves 5 more problems than LAMA, 18 more than PROBE and 161 more than FF. Significant differences in coverage occur in *Sokoban*, *Parking*, *NoMystery* and *Floortile* where either LAMA or BFS(f) solve 10% more instances than the second best planner. The largest difference is in *NoMystery* where BFS(f) solves 19 instances while LAMA solves 11.

Time and plan quality averages are computed over the instances that are solved by BFS(f), LAMA and PROBE. FF is excluded from these averages because of the large gap in coverage. LAMA and PROBE are the fastest in 16 domains each, and BFS(f) in 5. On the other hand, BFS(f) finds shorter plans in 15 domains, PROBE in 13, and LAMA in 10. The largest differences between BFS(f) and the other planners are in *8puzzle*, *Parking*, *Pipesworld Tankage*, *Pipesworld Non Tankage*, and *VisitAll*.

The results show that the performance of BFS(f) is at the level of the best planners. The question that we address next is what's the contribution of the four different ideas combined in the evaluation function $f(n)$ and in the tie-breakers; namely, the additive heuristic $h_{add}(n)$, the landmark count $usg(n)$, the novelty measure $novel(n)$, and the helpful action distinction $help(n)$. The last two terms are the ones that determine the evaluation function $f(n)$ in (1) through the formula (2).

Table 5 shows the result of a simple ablation study. The first row shows the results for the planner BFS(f) as described above, while

[4] FF is FF2.3, while PROBE and LAMA are from the 2011 IPC.

Table 5. Ablation study of BFS(f) when some features are excluded. Delayed evaluation excluded from second and following rows, in addition to feature shown. Columns show number of instances (I), number of instances solved (S), % solved (%S), and average plan lengths (Q) and times in seconds (T).

	I	S	% S	Q	T
BFS(f)	1150	1070	93%	82.80	62.89
No Delayed Eval	1150	1020	89%	80.67	65.92
No Heuristic	1150	965	84%	100.47	32.43
No Helpful Actions	1150	964	84%	81.82	64.20
No Novelty	1150	902	78%	86.40	46.11

the following rows show results for the same planner with one or several features removed: first delayed evaluation, then the additive heuristic, helpful actions, and novelty. This is achieved by setting $help(n) = 0$, $h_{add}(n) = 0$, and $novel(n) = 1$ respectively in (1) and (2) for all n. As it can be seen from the table, the greatest drop in performance arises when the novelty term is dropped. In other words, the novelty measure is no less important in the BFS(f) planner than either the helpful action distinction or the heuristic. The most important term of all however is the $usg(n)$ that counts the number of unachieved goals, and whose effect is to 'serialize' the best-first search to the goal without giving up completeness (as *SIW*). Moreover, the definition of the novelty measure in (2) uses the $usg(n)$ count to delimit the set of previously generated states that are considered. Yet BFS(f') with the evaluation function $f' = usg(n)$, i.e., without *any* of the other features, solves just 776 instances. On the other hand, with the term $novelha(n)$ added, the number jumps to 965, surpassing FF that solves 909 problems. This is interesting as BFS(f') uses no heuristic estimator then.

7 DISCUSSION

We have introduced a width parameter that bounds the complexity of classical planning problems along with an iterative pruned breadth-first algorithm *IW* that runs in time exponential in the problem width. While most benchmark domains appear to have a bounded and small width provided that goals are restricted to single atoms, they have large widths for arbitrary joint goals. We have shown nonetheless that the algorithm derived for exploiting the structure of planning problems with low width, *IW*, also pays off over benchmarks with joint goals once the same algorithm is used for decomposing the problems into subproblems. Actually, the resulting blind-search algorithm *SIW* is competitive with a baseline planner based on a Greedy Best First Search and the additive heuristic, suggesting that the two ideas underlying *SIW*, novelty-based pruning and goal decomposition, are quite powerful. We have also shown that it is possible to integrate the notion of novelty derived from width considerations, with existing planning techniques for defining the evaluation function of a novel best-first search planner with state-of-the-art performance. Moreover, we have shown that the new technique contributes to the performance of this planner no less than helpful actions or heuristic estimators.

The notion of width is defined over graphs $\mathcal{G}^m$ whose vertices are tuples of at most m atoms. This suggests a relation between the width of a problem and the family of admissible h^m heuristics which are also defined over tuples of at most m atoms [9]. A conjecture that we considered is whether a width of w implies that h^w is equal to the optimal heuristic h^*. The conjecture however is false.[5] It turns out that for this correspondence to be true, an additional clause is needed in the definition of the heuristic h^m; namely, that $h^m(t)$ is no lower than $h^m(t')$ when t' is a tuple of at most m atoms that implies t in the sense defined in Section 3. Yet, checking this implication in general is intractable.

In this paper, we have only considered planning problems where actions have uniform costs. Some of the notions and algorithms developed in the paper, however, extend naturally to non-uniform costs provided that the breadth-first search in *IW* is replaced by a uniform-cost search (Dijkstra).

ACKNOWLEDGEMENTS

H. Geffner is partially supported by grants TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

REFERENCES

[1] E. Amir and B. Engelhardt, 'Factored planning.', in *Proc. IJCAI-03*, (2003).
[2] C. Bäckström, *Five years of tractable planning*, 19–33, IOS Press, 1996.
[3] B. Bonet and H. Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**, 5–33, (2001).
[4] R. I. Brafman and C. Domshlak, 'Factored planning: How, when, and when not.', in *Proc. AAAI-06*, (2006).
[5] T. Bylander, 'The computational complexity of STRIPS planning', *Artificial Intelligence*, **69**, 165–204, (1994).
[6] H. Chen and O. Giménez, 'Act local, think global: Width notions for tractable planning', in *Proc. ICAPS-07*, (2007).
[7] R. Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.
[8] E.C. Freuder, 'A sufficient condition for backtrack-free search', *J. ACM*, **29**, 24–32, (1982).
[9] P. Haslum and H. Geffner, 'Admissible heuristics for optimal planning', in *Proc. AIPS-00*, (2000).
[10] J. Hoffmann, 'The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables.', *JAIR*, **20**, 291–341, (2003).
[11] J. Hoffmann, 'Analyzing search topology without running any search: On the connection between causal graphs and h^+', *JAIR*, **41**, 155–229, (2011).
[12] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *JAIR*, **14**, 253–302, (2001).
[13] J. Hoffmann, J. Porteous, and L. Sebastia, 'Ordered landmarks in planning', *JAIR*, **22**, 215–278, (2004).
[14] Jörg Hoffmann, 'Where 'ignoring delete lists' works: Local search topology in planning benchmarks', *JAIR*, **24**, 685–758, (2005).
[15] E. Keyder, S. Richter, and M. Helmert, 'Sound and complete landmarks for and/or graphs', in *Proc. ECAI-10*, (2010).
[16] N. Lipovetzky and H. Geffner, 'Searching for plans with carefully designed probes', in *Proc. ICAPS-11*, (2011).
[17] S. Richter and M. Helmert, 'Preferred operators and deferred evaluation in satisficing planning', in *Proc. ICAPS-09*, (2009).
[18] S. Richter, M. Helmert, and M. Westphal, 'Landmarks revisited', in *Proc. AAAI-08*, (2008).
[19] S. Richter and M. Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *JAIR*, **39**, 122–177, (2010).
[20] Lin Zhu and Robert Givan, 'Landmark extraction via planning graph propagation', in *Proc. ICAPS-03 Doctoral Consortium*, (2003).

[5] A counterexample is due to Blai Bonet. Consider a problem P with initial situation $I = \{r\}$, goal $G = \{z\}$, and actions $a : r \rightarrow p, \neg r$, $b : r \rightarrow q, \neg r$, $c : r \rightarrow x$, $d : x \rightarrow p, q, y$, and $e : p, q \rightarrow z$. It can be shown that the problem has width $w = 1$, as the graph $\mathcal{G}^1$ contains the rooted path r, x, y, z. Yet, $h^*(P) = 3$ with optimal plan c, d, e, while the heuristic h^m for $m = 1$, is h_{max}, which for this problem is 2. Thus, $w(P) = w$ and yet $h^*(P) \neq h^w(P)$.