Width-based Algorithms for Classical Planning: New Results

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

Abstract. We have recently shown that classical planning problems can be characterized in terms of a width measure that is bounded and small for most planning benchmark domains when goals are restricted to single atoms. Two simple algorithms have been devised for exploiting this structure: Iterated Width (IW) for achieving atomic goals, that runs in time exponential in the problem width by performing a sequence of pruned breadth first searches, and Serialized IW (SIW) that uses IW in a greedy search for achieving conjunctive goals one goal at a time. While SIW does not use heuristic estimators of any sort, it manages to solve more problems than a Greedy BFS using a heuristic like h_{add}. Yet, it does not approach the performance of more recent planners like LAMA. In this short paper, we introduce two simple extension to IW and SIW that narrow the performance gap with state-of-the-art planners. The first involves changing the *greedy search* for achieving the goals one at a time, by a *depth-first search* that is able to backtrack. The second involves computing a relaxed plan once before going to the next subgoal for making the pruning in the *breadth-first procedure* less agressive, while keeping IW exponential in the width parameter. The empirical results are interesting as they follow from ideas that are very different from those used in current planners.

1 INTRODUCTION

The main approach for domain independent planning is based on heuristic search with heuristics derived automatically from problems [1]. To this, recent planners add other ideas like helpful actions, landmarks, and multiqueue best-first search for combining different heuristics [3, 2, 6]. From a different angle, we have recently shown that most of the benchmark domains are easy when the goals contain single atoms, and that otherwise, goals can be easily serialized [5]. More precisely, we showed that the former problems have a low width, and developed an algorithm, Iterative Width (IW) that runs in time that is exponential in the problem width by performing a sequence of pruned breadth first searches. This algorithm is used in the context of another algorithm, Serialized IW (SIW), that achieves conjunctive goals by using IW greedily for achieving one goal at a time. Surprisingly, the blind-search algorithm SIW which has no heuristic guidance of any sort, performs better than a greedy best-first search guided by delete-relaxation heuristics. SIW, however, does not perform as well as the most recent planners that incorporate other ideas as well.

In this short paper, we introduce two simple extensions to IW and SIW that narrow the performance gap between width-based algorithms and state-of-the-art planners.

[1] The University of Melbourne, Melbourne, Australia, email: first.lastname@unimelb.edu.au

[2] ICREA & Universitat Pompeu Fabra, Barcelona, Spain, email: first.lastname@upf.edu

2 PRELIMINARIES

We assume a STRIPS problem $P = \langle F, I, O, G\rangle$, where F is the set of atoms, I is the set of atoms characterizing the initial state, O is the set of actions, and G is the set of goal atoms.

The algorithm *Iterated Width* or *IW* consists of a sequence of calls *IW*(i) for $i = 0, 1, \ldots, |F|$ until the problem is solved. Each iteration *IW*(i) is a breadth-first search that prunes states that do not pass a *novelty* test; namely, for a state s in *IW*(i) *not* to be pruned there must be a tuple t of at most i atoms such that s is the first state generated in the search that makes t true. The time complexities of *IW*(i) and *IW* are $O(n^i)$ and $O(n^w)$ respectively where n is $|F|$ and w is the problem width. The width of existing domains is low for atomic goals, and indeed, 89% of the benchmarks can be solved by *IW*(2) when the goal is set to any of the atoms in the goal [5]. The width of the benchmark domains with conjunctive goals, however, is not low in general, yet such problems can be serialized.

The algorithm *Serialized Iterative width* or *SIW* uses *IW* for serializing a problem into subproblems and for solving the subproblems. Basically, SIW uses IW to achieve one atomic goal at a time, greedily, until all atomic goals are achieved jointly. In between, atomic goals may be undone, but after each invocation of IW, each of the previously achieved goals must hold. SIW will thus never call IW more than $|G|$ times where $|G|$ is the number of atomic goals. SIW compares surprisingly well to a baseline heuristic search planner based on greedy best-first search and the h_{add} heuristic [1], but doesn't approach the performance of the most recent planners. For this, a new planner $BFS(f)$ was introduced that integrates the *novelty* measure with helpful-actions, landmarks and delete-relaxation heuristics [5]. Here we aim to achieve similar results but with different and simpler ideas.

3 EXTENSIONS: *IW*$^+$ AND *DFS(i)*

The first extension, from *IW* to *IW*$^+$ involves a slight change in the pruning criterion in the breadth-first search *IW*(i) procedure. When the new procedure *IW*$^+(i)$ is called from a state s, a *relaxed plan* from s is computed, so that the states s' generated by *IW*$^+(i)$ keep a count of the number of atoms in the relaxed plan from s that have been achieved in the way to s'. These atoms are the ones made true by actions in the relaxed plan that do not hold in s. We say that state s' makes the *extended tuple* (t, m) true if s' makes the tuple t true and m is the the number of atoms in the relaxed plan from s that are made true by the sequence of actions leading to s' in *IW*$^+$. For the state s' in the breadth-first search underlying *IW*$^+(i)$ *not* to be

pruned, the new condition is that there must be a tuple t with at most i atoms, such that s' is the first state in the search that makes the *extended tuple* (t, m) true. An interesting property and motivation of the *IW*$^+$ algorithms is that all *delete-free* problems will be solved efficiently, as they will be solved by *IW*$^+(1)$. The same is not true of *IW*(1).

While the algorithm *IW* calls the algorithm *IW*(i) sequentially, *IW*$^+$ calls *IW*$^+(i)$ instead. The complexity of *IW*$^+$ is $O(n^{w+1})$ where w is the problem width, as in the worst case, the number of atoms in a relaxed plan can be equal to the number of fluents.

The second extension is a depth-first search algorithm that extends the serialization of conjunctive goals made greedily by *SIW* with the ability to backtrack. For this, the depth-first search algorithm *DFS(i)* uses the positive integer parameter i to indicate that invocations of the *IW*$^+$ procedure should involve a sequence of *IW*$^+(0)$, *IW*$^+(1), \ldots,$ *IW*$^+(i)$ calls that should fail and lead to a backtrack when *IW*$^+(i)$ fails to achieve one more atomic goal. *SIW*, on the other hand, will keep trying *IW*$(i+1)$, *IW*$(i+2)$, and so on, and will never backtrack.

DFS(i) can be understood as normal depth-first search where the actions in each node are sets of "macro actions" *IW*$^+(1), \ldots,$ *IW*$^+(i)$ performed in order, such the children resulting from the "macro actions" in *IW*$^+(k)$ applied in a state s are the states s' reachable by *IW*$^+(k)$ from s that achieve all the goal atoms in s plus one.

Notice that while *DFS(i)* computes a relaxed plan once for each *IW*$^+$ call, *DFS(i)* does not use the relaxed plan for computing heuristic estimates.

4 EXPERIMENTS

The algorithm *SIW*$^+$ is *SIW* with the *IW* procedure replaced by *IW*$^+$. Thus, *SIW*$^+$ incorporates the first extension only, while *DFS(i)* incorporates both extensions. In order to evaluate their performance, we include results for a baseline heuristic search planner made of a greedy best-first search (GBFS) driven by h_{add} [1], and state-of-the-art planners such as FF [3], PROBE [4], LAMA-11 [6] and BFS(f). All planners are run over the set of benchmarks from the last International Planning Competition on a 2.40GHz Intel Processor, with processing time or memory out of 30min and 2GB.

Table 1. Number of instances solved. Bold shows best performer.

	GBFS	SIW	SIW$^+$	FF	DFS$^+$	BFS(f)	PROBE	LAMA'11
Barman	0	0	17	0	**20**	**20**	**20**	**20**
Elevators	2	17	19	**20**	**20**	17	**20**	**20**
Floortile	3	0	0	2	0	**6**	5	5
NoMystery	6	0	0	5	4	**15**	6	10
Openstacks	14	5	**20**	**20**	**20**	16	14	**20**
Parcprinter	19	**20**	**20**	**20**	**20**	17	13	19
Parking	5	**20**	**20**	19	**20**	**20**	19	19
Pegsol	**20**	0	1	**20**	19	**20**	**20**	**20**
Scanalyzer	**18**	16	16	17	**18**	17	**18**	**18**
Sokoban	15	0	0	15	1	13	15	**17**
Tidybot	17	5	17	**19**	**19**	18	**19**	15
Transport	4	13	14	8	16	16	16	**17**
VisitAll	3	**20**	**20**	3	**20**	**20**	18	**20**
Woodworking	2	19	**20**	19	**20**	**20**	**20**	19
All	128	135	184	187	217	235	223	**239**

Interestingly, the results in Table 1 and 2 highlight the big gap in performance reached by just considering the first extension *IW*$^+$ in the greedy serialization *SIW*$^+$, reaching indeed the performance of FF that uses helpful actions in addition to heuristics and was the state-of-the-art planner until few years ago. When the second extension is considered, the performance compares well with current state-of-the-art planners that also incorporate multiple heuristics and landmarks. Indeed, as shown in Table 2, even by having a smaller coverage, *DFS(i)* has the *highest IPC score*[3] in terms of speed and doesn't perform bad at all in terms of the *quality IPC score*. *DFS(i)* backtracks mainly in Pegsol, NoMystery, and Sokoban: 31000, 21217, and 137 times on average.

Table 2. Time(T), and plan length (Q) of first solution as IPC scores. Bold shows best performer.

		GBFS	SIW	SIW$^+$	FF	DFS$^+$	BFS(f)	PROBE	LAMA'11
Barman	T	0	0	3.59	0	4.52	15.51	**16.56**	10.18
	Q	0	0	15.3	0	17.95	**19.19**	17.39	15.19
Elevators	T	0.01	1.23	11.33	11.78	15.62	2.41	6.16	**16.33**
	Q	1.14	15.24	10.83	**19.44**	10.93	12	14.62	18.24
Floortile	T	0.06	0	0	0.59	0	**3.26**	2.2	2.8
	Q	2.96	0	0	1.83	0	**5.75**	4.61	4.67
NoMystery	T	1.59	0	0	5	0.08	**12.04**	2.52	9.48
	Q	5.74	0	0	5	3.77	**14.57**	5.72	9.72
Openstacks	T	0.64	1.73	15.72	11.62	**19.88**	1.99	1.64	13.94
	Q	13.4	5	**19.83**	18.81	**19.83**	15.38	13.79	19.24
Parcprinter	T	16.06	**20**	**20**	**20**	**20**	7.85	11.74	19
	Q	18.9	19.01	19.01	**19.79**	19.01	16.01	12.88	18.31
Parking	T	0.07	**19.19**	18.82	2.57	18.5	2.35	2.15	5.06
	Q	2.53	**19.53**	19.36	10.94	19.42	10.72	6.27	10.68
Pegsol	T	19.01	0	1	**20**	7.3	19.5	16.86	19.23
	Q	17.89	0	1	17.93	17.1	**18.99**	18	17.92
Scanalyzer	T	5.15	**15.24**	14.32	13.27	14.97	9.91	13.75	10.71
	Q	14.81	13.48	13.4	15.82	14.88	14.04	**16.56**	14.86
Sokoban	T	2.2	0	0	9.47	0.3	6.68	**11.22**	10.08
	Q	9.66	0	0	**13.83**	0.93	10.58	10.29	13.23
Tidybot	T	1.66	3.04	6.88	**14.41**	8.44	2.32	8.02	2.81
	Q	13.04	4.86	14.77	14.43	15.81	15.31	**16.46**	12.54
Transport	T	0.09	3.11	13.67	0.18	**15.17**	4.91	5.84	9.23
	Q	2.66	11.79	12.27	6.04	13.38	14	11.14	**15.82**
VisitAll	T	0.14	17.37	12.81	1.88	12.96	**20**	5.68	6.46
	Q	0.25	19.5	19.5	1.58	19.5	**20**	14.39	14.69
Woodworking	T	1.01	7.92	6.05	**19**	7.18	2.16	2.74	3.44
	Q	2	17.47	17.88	16.97	17.81	19.86	**20**	15.51
All	T	47.69	88.84	124.2	129.78	**144.92**	110.89	107.07	139.96
	Q	104.98	125.88	163.15	162.41	190.91	**206.49**	182.12	200.62

5 CONCLUSIONS

We have shown how ideas based on width considerations result in a simple planner whose performance approaches the performance of the best current planners. The ideas are very different than those that can be found in current planners that usually rely on heuristic estimators, helpful actions, and landmarks, suggesting that they may deserve further consideration.

ACKNOWLEDGEMENTS

This research was co-funded by ARC linkage grant LP11010015, TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

REFERENCES

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

[2] Malte Helmert, 'The Fast Downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[3] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[4] Nir Lipovetzky and Héctor Geffner, 'Searching for plans with carefully designed probes', in *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pp. 154–161, (2011).

[5] Nir Lipovetzky and Héctor Geffner, 'Width and serialization of classical planning problems', in *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI 2012)*, pp. 540–545, (2012).

[6] Silvia Richter and Matthias Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *Journal of Artificial Intelligence Research*, **39**, 122–177, (2010).

[3] Score for each planner is the sum of T/T^* (Q/Q^*), where T (Q) is solution time (length) divided by the fastest (shortest) solution found T^* (Q^*).