Analyzing the Impact of Partial States on Duplicate Detection and Collision of Frontiers

Vidal Alcázar, Susana Fernández, Daniel Borrajo
Universidad Carlos III de Madrid
Av. Universidad, 30
28911 Leganés, Spain
valcazar@inf.uc3m.es;sfarregu@inf.uc3m.es;dborrajo@ia.uc3m.es

Abstract

Partial states are states in which the truth value of one or more propositions is unknown. Such states are usually generated in regression and represent not a single state but rather a set of states. Because of this, a new partial state can be a subset of another existent partial state, phenomenon known as subsumption of states. Subsumed states can be pruned as if they were a duplicate. However, regular duplicate detection methods cannot detect such cases. Furthermore, subsumption of states also occurs when forward and backward search algorithms are integrated into a bidirectional planner. In these cases, the forward frontier contains only complete states and the backward frontier will often contain partial states. In this work, we analyze the impact that subsumption of states has on search and propose methods for duplicate detection and detection of collision of frontiers.

Introduction

Automated planning consists of finding a sequence of actions, commonly known as plan, that reaches a set of goals from a given initial state. The most common approach nowadays is planning as heuristic search (Bonet and Geffner 2001), which can be done either advancing from the initial state towards a goal state (called forward search or progression) or from a goal state to the initial state (called backward search or regression). Due to the different nature of regression, backward search planners have several drawbacks, as recently pointed out in (Alcázar et al. 2013). Some of them have been thoroughly studied, like the generation of spurious states; however, it is still unclear how and by which amount subsumption of states, often cited as a problematic phenomenon, affects search.

A partial state s_1, that represents a set of complete states, subsumes another partial state s_2 if s_1 is a subset of s_2; or, equivalently, when the set of propositions of the subsuming state s_1 is a subset of the set of propositions of the subsumed state s_2. For example, if some state $s = \{p\}$ already exists and a new state $s' = \{p, q\}$ is generated, s subsumes s', as the set of states that s' represents is included in the set of states represented by s. Subsumed states can be labeled as duplicates and thus can be safely pruned if their g value is not lower than the g value of the subsuming state. Heuristic search planners store the closed list in a hash table to allow the detection of regular duplicates. However, subsumption of states is not detectable using a hash function. Therefore, backward search planners will expand subsumed states. This may lead to an exponential decrease in performance, so addressing this problem seems to be of critical importance.

The impact of subsumption of states is not limited to the duplicate detection case. Bidirectional search has the great advantage of allowing halting the search when the frontiers of both searches collide, which reduces the depth of the search tree and hence may decrease the search effort exponentially. However, detecting the collision of frontiers is equivalent to duplicate detection in regression. This is the main reason why bidirectional search, an otherwise attractive option *a priori* for satisficing planing, has not been implemented yet in explicit-state planners.

In this work we propose several alternatives to deal with subsumption of states and analyze their impact. First, *disambiguating* reduces the cases in which subsumption occurs by completing the partial states (Alcázar et al. 2013), so its impact will be studied. Second, we propose the use of Binary Decision Diagrams (BDDs) (Bryant 1986) to detect subsumption for both backward and bidirectional search. Third, we will implement front-to-front heuristics based on Backwards Generated Goals (Alcázar, Borrajo, and López 2010), able to detect collision of frontiers.

Background

In this section we will provide some definitions and concepts needed for subsequent sections.

Propositional Planning and Subsumption

A planning task is defined as a tuple *P=(S,A,I,G)*. *S* is the set of propositions, *A* is the set of grounded actions, $I \subseteq S$ is the initial state, and $G \subseteq S$ is the set of goal propositions. Each action $a \in A$ is defined as a triple *(pre(a), add(a), del(a))* (preconditions, add effects and delete effects) where *pre(a), add(a), del(a)* $\subseteq S$. Optimal planners return a provably least-cost plan; satisficing planners cannot prove optimality. Partial states are states in which at least one proposition $p \in S$ has an unknown value. Subsumed states can be pruned as if they were duplicates.

Definition 1 *Given two states $s_i, s_j \subseteq S$, s_i subsumes s_j ($s_i \sqsubseteq s_j$) if $s_i \subseteq s_j$ and $g(s_i) \leq g(s_j)$.*

Disambiguation of States

Disambiguation (Alcázar et al. 2013) is the process of reducing the valid values of unknown propositions given a set of known propositions $s \subseteq S$ and a set of constraints. Disambiguation consists of solving a Constraint Satisfaction Problem (CSP) with the unknown propositions as variables and state invariants like mutexes (Bonet and Geffner 2001) as constraints. Disambiguation detects spurious sets of propositions (when a proposition $p \in S$ cannot be true nor false) and deduces the value of unknown propositions, which allows completing partial states.

Binary Decision Diagrams

Decision Diagrams (Bryant 1986) are data structures inspired by the graphical representation of a logical function. A Binary Decision Diagram (BDD) is a directed acyclic graph with two terminal nodes (called *sinks*) labeled with *true* and *false*. All the internal nodes are labeled with a proposition $p \in S$ and have two outgoing edges that correspond to the cases in which p is *true* and *false*, respectively. For any assignment of the variables on a path from the root to a sink, the represented logical function will be evaluated to the value labeling the sink.

Backward Generated Goals

Backward Generated Goals (BGGs) are partial states generated from the goals used to reinforce a forward reachability heuristic (Alcázar, Borrajo, and López 2010). BGGs were originally computed using the last actions of the relaxed plan obtained from the FF heuristic (Hoffmann and Nebel 2001), although they can be generated with any regression technique. BGGs allow computing any reachability heuristic against an arbitrary number of intermediate sets of goals, although they introduce an overhead. When using BGGs the distance is computed against the BGG with the lowest h^{max}.

Subsumption of States and Duplicate Detection in Regression

Failing to prune subsumed states in regression may have a very negative impact. Imagine the following case: in a *Logistics* task the states s_0={*(at Truck A), (at Package A)*}, s_1={*(at Truck A), (in Package Truck)*} and s_2={*(at Truck B), (at Package A)*} have been visited. A new state that includes a new truck s_3={*(at Truck A), (at Package A), (at Truck2 A)*} is generated; in this case $s_0 \sqsubseteq s_3$, but a hash function is unable to detect it. Consequently, a search tree of exponential size whose root is s_3 may be explored, with a huge impact on performance. Furthermore, if the location of an arbitrary number of trucks n is unknown, a total of 2^n trees (each one of exponential size) may have to be explored in the worst case.

Several techniques can be employed to avoid this futile exploration. First, disambiguation allows completing partial states. Thus, it helps with cases in which, given two states $s, s' \subseteq S$, $s \sqsubseteq s'$ before disambiguation and $s = s'$ after disambiguation. Suppose a planner is using regression to solve a problem in *Blocksworld* and expands the partial state s={*(on A B)*}. If the actions *(stack A B)* and *(unstack A B)* are applied in sequence in regression, it would generate a new state s'={*(on A B), (clear A), (arm-empty)*}. A human will quickly realize that this sequence leads to the same state, but a planner will not realize that s subsumes s'. However, if disambiguation is done, then both s and s' become s, s'={*(on A B), (clear A), (arm-empty), (on-table B), ¬(clear B)*}, so a regular hash function would be able to prune s'. Disambiguation means solving a CSP per state, which in some domains may be costly. Nevertheless, it is possible to disambiguate the preconditions of the actions in A, which allows pruning spurious actions and obtaining more complete states when the actions in A are applied in regression. In fact, it may be the case that disambiguating G and the actions in A may be enough to obtain most of the benefits of disambiguating per state.

Another alternative is storing the states expanded in regression in both a regular hash table *and* a BDD, as BDDs detect subsumption efficiently. To illustrate this let us revisit the previous *Logistics* task. Figure 1 depicts the BDD that contains s_0, s_1 and s_2. The edges highlighted in red represent the path that corresponds to s_0. If s_3 is generated, checking the value of the propositions of s_3 in the BDD will lead to the *true* node following the highlighted path, which means that the BDD contains some state that subsumes s_3.

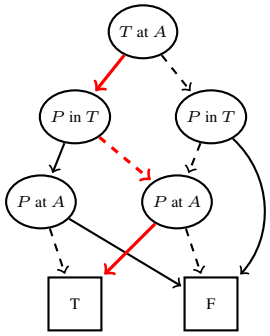

Figure 1: Using a BDD as the closed list of a *Logistics* task.

Here s_3 is encoded as the BDD b and intersected with the BDD $closed$ used to represent the closed list. If $b \cap closed = b$, then s_3 is subsumed. Note however that inserting and querying in a BDD is slower than in a hash table and that its complexity in size is exponential on the number of partial states (Edelkamp and Kissmann 2011), which may cause overhead. Also, BDDs do not store pointers to parent states, so the hash table is still necessary to trace back a path.

Forward Collision of Frontiers and Partial States

Bidirectional search seems well suited to satisficing planning. The main disadvantage of bidirectional search is that it must continue searching after the frontiers collide to prove

optimality (Kaindl and Kainz 1997) - unless an admissible front-to-front heuristic is used, which is often prohibitively expensive. In satisficing planning optimality is not required, so the search can be halted, and a solution returned, the first time a collision of frontiers is detected. The reason why this has not been done in satisficing planning yet is because detecting the collision of frontiers when partial states are present requires dealing with subsumption of states.

Here we propose two methods to detect collision of frontiers, the use of BDDs and employing BGGs to derive a front-to-front heuristic. Using BDDs is analogous to the case presented in the previous section; the closed list of the backward search is duplicated in a BDD and we check whether states generated forward are subsumed by the BDD. This is also possible when detecting backward collision of frontiers, although the operation would be $b \cap closed \neq \emptyset$. Note that since BDDs merge nodes to maintain the succinctness of the data structure, the exact subsuming state may not be obtained by following the path down the BDD and hence the hash table must be sequentially explored after detecting the collision to find the subsuming state.[1] More efficient alternatives to overcome this may be possible, *i.e.* as done in frontier search (Korf et al. 2005).

BGGs also allow detecting forward collision of frontiers. Any reachability heuristic enriched with BGGs returns 0 when evaluating a state subsumed by a BGG. In this work BGGs are created from the states *expanded* backwards instead of using the last actions of forward relaxed plans. As the BGG corresponds univocally to a state in the backward closed list, tracing back a path is straightforward.

A Case for Sparser Frontiers when Using BGGs

Like other front-to-front heuristics, using BGGs becomes increasingly expensive as the opposite frontier grows. As the overhead is proportional to the number of BGGs, keeping their number to a minimum may alleviate this. We propose removing BGGs at random when a given threshold is surpassed. Ideally, if only a subset of BGGs is kept, the farther they are from each other the better they will represent the explored search space. Removing them at random does not guarantee this, although it is likely that, if an area of the search space contains a high amount of BGGs, most of the removed BGGs will belong to that area. Another concern is that removing BGGs may render the h value of nodes obsolete. This is so for every front-to-front heuristic when the opposite frontier changes for any reason (*i.e.* when new nodes are generated), so it is for the most part unavoidable.

Another alternative is using sparse data structures. Random Planning Trees (RPT) (Alcázar, Veloso, and Borrajo 2011), an adaptation of Rapidly-Exploring Random Trees (Kuffner and LaValle 2000) for automated planning, keep a sparse tree instead of a dense closed list. They are generated expanding towards randomly sampled states and the goal alternatively. They use a local planner with a limit in the number of expanded states to create new nodes of the tree. The number of nodes of the tree is usually very small, so keeping a BGG per node is unlikely to cause a big overhead. Furthermore, RPTs tend to explore the search space evenly thanks to their sampling method, which impacts positively the accuracy of the BGG-derived heuristic.

Figure 2 shows how BGGs work. Both trees represent RPTs, although they can also be seen as regular search trees generated by best-first search algorithms. The waves generated from S_i represent the layers of the relaxed exploration of the heuristic. This shows how the heuristic computation is stopped when the first BGG is satisfied and how the distance to the BGG is computed by extracting a relaxed plan.

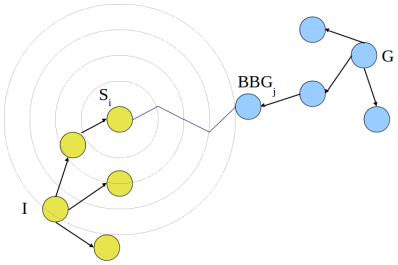

Figure 2: Computation and extraction of a relaxed plan from the forward RPT node S_i to the closest BGG BGG_j.

Experimentation

All the techniques were implemented on top of Fast Downward (Helmert 2006). Benchmarks, score and settings were the ones from the International Planning Competition of 2011. First we evaluate the impact of subsumption in regression. We used FDr using greedy best-first search and the cached FF heuristic (Alcázar et al. 2013). G and the preconditions of the actions in A are always disambiguated. Versions labeled with b use a BDD and with d do disambiguation per state. The results of Table 1 show that surprisingly BDDs only help in two domains, *Sokoban* and *Parking*, improving the coverage considerably in the former. In the rest of the domains BDDs yield a similar or worse result, losing 4 problems in *Visitall*. Disambiguation appears to be more useful, but its main benefit comes from detecting spurious states, which explains why it solves more problems in *Floortile* and *Parking*. FDr(bd) achieves the highest coverage, which suggests that avoiding pathological cases induced by subsumption and spurious states pays off despite the overhead introduced. Nevertheless, the impact of subsumption is minor in most domains as long as G and the preconditions of the actions are disambiguated.

Now we try 10 planners in the context of bidirectional search: FD, Fast Downward with lazy evaluation, the FF heuristic, preferred operators and spurious operator pruning; FDr(bd); biFD, a combination of FD and FDr(bd); biFD(c), biFD with collision of frontiers; BGG, biFDc + BGGs; BGG(s), BGG with a sparse frontier - when the number of BGGs is greater than 2k BGGs are removed until there are only 1k; RPT, a forward RPT with a limit of 100k nodes and p=0.5 that uses FD as local planner; RPTr, RPT but uses FDr; biRPT, combination of RPT and RPTr with a limit of 10k; and biRPT(bgg), biRPT + BGGs. All bidirectional

[1] This happens with forward collision of frontiers; in backward collision of frontiers the required state is encoded by $b \cap closed$.

Domain	FD	FDr	biFD	biFD(c)	BGG	BGG(s)	RPT	RPTr	biRPT	biRPT(bgg)
Barman	**19**	0	18	17	1	1	11	0	8	5
Elevators	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Floortile	7	**20**	**20**	**20**	**20**	**20**	9	18	**20**	**20**
Nomystery	10	6	10	10	9	10	10	4	11	**12**
Openstacks	**20**	0	19	19	2	8	**20**	5	13	18
Parcprinter	**12**	8	**12**	**12**	11	**12**	**12**	7	**12**	11
Parking	**20**	0	19	19	2	7	**20**	3	8	8
Pegsol	**20**	17	**20**	**20**	14	19	**20**	5	**20**	**20**
Scanalyzer	19	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Sokoban	**19**	11	**19**	**19**	12	16	17	5	18	11
Tidybot	**17**	2	15	15	14	14	**17**	4	2	9
Transport	**11**	5	**11**	**11**	10	**11**	**11**	5	10	6
Visitall	5	2	5	5	2	2	**10**	8	6	6
Woodworking	**20**	14	**20**	**20**	19	19	**20**	18	15	18
Total	219	129	**228**	227	156	179	217	122	183	184

Table 2: Problems solved in IPC-11.

	FDr(bd)		FDr(b)		FDr(d)		FDr	
Barman	0	0	0	0	0	0	0	0
Elevators	15.50	**20**	16.36	**20**	18.59	**20**	**19.17**	**20**
Floortile	14.79	**20**	13.21	18	**15.79**	**20**	13.84	18
Nomystery	4.90	**6**	4.07	**6**	4.54	**6**	**5.00**	**6**
Openstacks	0	0	0	0	0	0	0	0
Parcprinter	6.53	**8**	6.92	**8**	5.95	**8**	**8.00**	**8**
Parking	3.06	4	2.35	3	**3.73**	**5**	1.10	2
Pegsol	3.61	17	6.10	18	8.56	18	**18.00**	**18**
Scanalyzer	16.19	**20**	16.99	**20**	17.95	**20**	**19.42**	**20**
Sokoban	**10.67**	**11**	2.90	7	1.05	4	2.05	3
Tidybot	**1.98**	**2**	1.80	**2**	0.80	**2**	1.48	**2**
Transport	4.53	**5**	4.62	**5**	4.87	**5**	**4.97**	**5**
Visitall	0.30	2	0.30	2	2.38	4	**6.00**	**6**
Woodworking	4.97	**14**	13.56	**14**	4.97	**14**	**14.00**	**14**
Total	87	**129**	89	123	89	126	**113**	122

Table 1: Time score and coverage of FDr.

planners expand towards the direction that has used less time so far at each time. Results are shown in Table 2.

Several conclusions can be drawn from the experimentation. First, biFD obtains a higher coverage. However, detecting the collision does not offer any advantage, and in fact biFD(c) loses one problem - it times out when retrieving the subsuming state. This happens because in search spaces with symmetries and transpositions there is a high probability that the search trees for both directions are completely different. This is further the case when the density of solutions of the problem is high. Optimal search algorithms must explore all the states whose f value is lower than the optimal cost of the solution; however, suboptimal search algorithms commit strongly to subtrees in the search state and often do not collide with the opposite frontier until they are close to the goal. BGG, although guided towards the opposite frontier, is considerably less efficient than biFD due to the overhead induced by BGGs. A sparse frontier partially alleviates this problem, but the results are still not close to biFD's. RPTs are competitive with best-first search planners, but biRPT is not as efficient as biFD. biRPT(bgg) does not suffer from BGG's overhead and obtains a similar coverage to biRPT, although this varies greatly across domains.

Related Work

A recent work by Eyerich and Helmert (2013) identifies subsumption as an important problem when searching backwards and proposes the use of a partial-match trie (Rivest 1974). This is a similar idea to ours, but there are important differences. Tries allow retrieving subsuming states, so a path can be traced back immediately. However, tries have one leaf per state, which may be very memory demanding. Furthermore, queries in the trie are reported by the authors to be a bottleneck in terms of time. At every inner node of the trie both the corresponding edge and the *"don't care"* edge must be explored, which may lead to exploring a high number of paths. As a consequence, in some domains their planner runs out of time, a surprising fact given that planners that use abstraction heuristics typically exhaust memory long before time. As for BDDs, Table 1 shows that the overhead is not critical in the tested domains despite their exponential worst case. A direct comparison between partial-match data structures would shed light on this matter.

Conclusions

In this work an experimental evaluation of the impact of partial states has been done. Additionally, several new bidirectional planners have been implemented to test the viability of the proposed techniques. The impact of subsumption in regression varies from domain to domain, although it seems to be overstated (as long as G and the actions are disambiguated). Furthermore, bidirectional search does not seem to be better than combining two planners that search in opposite directions into a portfolio. These results apply to satisfacing planning; we leave a more extensive analysis of the described techniques in optimal planning as future work.

Acknowledgments

The authors would like to thank Malte Helmert and Álvaro Torralba for their insightful comments about the complexity of BDDs regarding partial states.

This work has been partially supported by the Spanish government through MICINN projects TIN2008-06701-C03-03 and TIN2011-27652-C03-02.

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence*.

Alcázar, V.; Borrajo, D.; and López, C. L. 2010. Using backwards generated goals for heuristic planning. In *International Conference on Automated Planning and Scheduling*, 2–9.

Alcázar, V.; Veloso, M. M.; and Borrajo, D. 2011. Adapting a Rapidly-Exploring Random Tree for automated planning. In *Symposium on Combinatorial Search*.

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

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Edelkamp, S., and Kissmann, P. 2011. On the complexity of BDDs for state space search: A case study in connect four. In *AAAI Conference on Artificial Intelligence*.

Eyerich, P., and Helmert, M. 2013. Stronger abstraction heuristics through perimeter search. In *International Conference on Automated Planning and Scheduling*.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research* 7:283–317.

Korf, R. E.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 995–1001.

Rivest, R. L. 1974. *Analysis of Associative Retrieval Algorithms.* Ph.D. Dissertation, Stanford, CA, USA.