KI - Künstliche Intelligenz (2018) 32:191–192
https://doi.org/10.1007/s13218-018-0531-2

SYSTEMS DESCRIPTION

Dynamic Programming on Tree Decompositions with D-FLAT

Michael Abseher[2] · Bernhard Bliem[1] · Markus Hecher[2] · Marius Moldovan[2] · Stefan Woltran[2]

Published online: 15 May 2018

Abstract
Many hard problems can be solved efficiently by dynamic programming algorithms that work on tree decompositions. In this paper, we present the D-FLAT system for rapid prototyping of such algorithms. Users can specify the algorithm for their problem using Answer Set Programming. We illustrate the framework by an example and briefly discuss its main features.

Keywords Tree decomposition · Answer set programming · Dynamic programming

1 Introduction

Graphs are ubiquitous in computing. Treewidth [7] is a parameter that, intuitively, measures how far a graph is from being a tree. This parameter proved to be very useful because many hard problems become tractable if instances are restricted to those whose treewidth is bounded by a constant [6]. Algorithms that exploit this usually employ dynamic programming on a tree decomposition of the instance. A tree decomposition is a tree obtained from a graph, where each node has a bag, which is a set of vertices of the original graph. A decomposition thus divides the instance into several parts. For many problems, we can then apply dynamic programming to compute tables containing partial solutions for the respective subproblems and finally combine them to obtain solutions for the whole problem (if possible).

The D-FLAT system [1, 4] is a tool for rapid prototyping of such algorithms. Users merely need to specify the computation at each node of the tree decomposition in a declarative way, and the system takes care of problem-independent parts like computing a decomposition and combining partial solutions. For formalizing the computations at the decomposition nodes, D-FLAT uses the logic programming language Answer Set Programming (ASP). ASP makes the specification of many combinatorial problems easy, and this often carries over to D-FLAT specifications.

✉ Bernhard Bliem
bernhard.bliem@helsinki.fi; bliem@dbai.tuwien.ac.at

1 University of Helsinki, Gustaf Hällströmin katu 2b, 00560 Helsinki, Finland

2 TU Wien, Favoritenstr. 9-11, 1040 Vienna, Austria

2 Dynamic Programming with D-FLAT

Given an ASP program specified by the user, D-FLAT first reads an input to the problem and generates an appropriate tree decomposition using heuristic methods. It then traverses the decomposition in post-order and executes the user-supplied program at each node (see Fig. 1) with the following information: the vertices in the current bag (`current`/1), those encountered for the first time (`introduced`/1), the names of children of the current decomposition node (`childNode`/1), the (globally unique) identifiers of rows from the tables of these child nodes along with the respective node names (`childRow`/2), and the contents of these child rows (`childItem`/2). Output predicate `item`/1 is used to store the content of the current table row, whereas `extend`/1 indicates the origins of this row by pointing to one row from each child table.

We illustrate problem solving in D-FLAT with an example. Consider the D-FLAT encoding from Listing 1, which implements a dynamic programming algorithm for finding independent sets (i.e., sets of vertices such that no two elements are adjacent to each other) in graphs given by facts over the predicate `edge`.

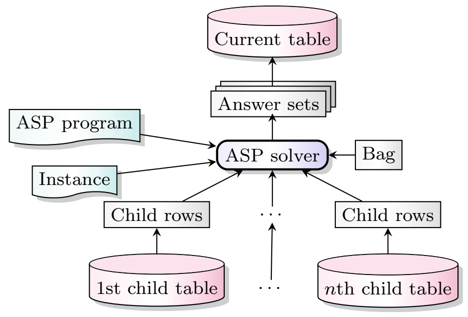

Fig. 1 Procedure at each tree decomposition node

```
1 { extend(R) : childRow(R,N) } 1 ←
  childNode(N).
← extend(R1), extend(R2), childItem(R1,X),
  not childItem(R2,X).
item(X) ← extend(R), childItem(R,X),
  current(X).
{ item(X) : introduced(X) }.
← edge(X,Y), item(X), item(Y).
```

Listing 1 Computing independent sets with D-FLAT

In line 1, we guess a row from each child table. By line 2, we never combine child rows that disagree on the status of a bag element. (For this we may assume that each decomposition node with more than one child has the same bag as all children.) If an extended partial solution contains an element that is still present in the current bag, line 3 makes sure that this element still appears in the current row. Line 4 guesses for each introduced bag element whether it is included in the partial solution, and we ensure in line 5 that the partial solution restricted to the current bag is independent.

2.1 Advanced Features

The algorithm formalized in Listing 1 computes a table at each tree decomposition node. However, many dynamic programming algorithms require more involved data structures than this "flat" table structure. Hence D-FLAT supports a generalization where we store a tree of tables at each decomposition node. It has been shown [5] that D-FLAT can thus solve all problems expressible in monadic second-order logic in linear time for instances of bounded treewidth.

D-FLAT also supports optimization problems and allows for anytime optimization; that is, the execution can be interrupted and the best solution found so far is reported [3]. This is realized via lazy evaluation, which essentially fills tables bit by bit. Hereby, D-FLAT does not simply perform one post-order traversal to compute all table rows but tries to compute as few as possible using backtracking in case the currently computed rows do not lead to a solution yet. Moreover, the system provides a facility for shifting arithmetic from ASP encodings into the framework, which often leads to smaller groundings. Finally, there is an extension of D-FLAT that allows for an easy and efficient handling of problems whose solutions are minimal or maximal w.r.t. set inclusion [2].

3 Sources

Source code and binary releases can be found at https://github.com/bbliem/dflat/. The project website is at http://dbai.tuwien.ac.at/proj/dflat/ and contains references to extensions and related software. For an exhaustive description of D-FLAT, we refer to [4].

Acknowledgements This work was supported by the Austrian Science Fund (FWF) projects P25607 and Y698.

References

1. Abseher M, Bliem B, Charwat G, Dusberger F, Hecher M, Woltran S (2014) The D-FLAT system for dynamic programming on tree decompositions. In: Proceedings of the JELIA 2014, LNCS, vol 8761. Springer, pp 558–572
2. Bliem B, Charwat G, Hecher M, Woltran S (2016) *D- FLAT*^{2}: subset minimization in dynamic programming on tree decompositions made easy. Fundam Inform 147(1):27–61
3. Bliem B, Kaufmann B, Schaub T, Woltran S (2016) ASP for anytime dynamic programming on tree decompositions. In: Proceedings of IJCAI 2016. AAAI Press, pp 979–986
4. Bliem B, Moldovan M, Woltran S (2017) The D-FLAT system: user manual. Technical report, DBAI-TR-2017-107, DBAI, TU Wien, Vienna, Austria
5. Bliem B, Pichler R, Woltran S (2017) Implementing Courcelle's theorem in a declarative framework for dynamic programming. J Logic Comput 27(4):1067–1094
6. Niedermeier R (2006) Invitation to fixed-parameter algorithms. In: Oxford lecture series in mathematics and its applications, vol 31. Oxford University Press
7. Robertson N, Seymour PD (1984) Graph minors. III. Planar treewidth. J Combin Theory Ser B 36(1):49–64