

# Using (hyper)graph decomposition for satisfiability decision

Oliver Kullmann

Computer Science Department  
University of Wales Swansea  
Swansea, SA2 8PP, UK

e-mail: [O.Kullmann@Swansea.ac.uk](mailto:O.Kullmann@Swansea.ac.uk)

<http://cs-svr1.swan.ac.uk/~csoliver/>

April 29, 2006

Discussing decomposable graph representations

Joint work with Marijn Heule  
(see *University of Wales Swansea Report Series  
CSR 2-2006*)

BCTCS 2006 (Swansea)

OK 04/2006

## Solving problems by graph representations

Consider a **SAT problem**  $F$  (it could also be a constraint satisfaction problem, or some other decision problem). We have to compute

$$\text{sat}(F) \in \{0, 1\}.$$

For that we want to exploit some **graph representation**  $G(F)$  (which can be some sort of graph or hypergraph).

There are many ways of doing so, but we want to focus on exploiting the most basic features of graphs, namely their **connectedness properties**:

- I If  $G(F)$  consists of several connected components, then we should be able to take advantage of this.
- II We should have efficient means to split a connected  $G(F)$  (in a meaningful way).

## Working on the components

Assume that  $G_1, \dots, G_m$  are the connected components of  $G(F)$ .

What to do with that information ?!

Our first basic assumption is, that we can efficiently associate “parts”  $F(G_i)$  to the components  $G_i$ , so that  $F(G_1), \dots, F(G_m)$  is a “partitioning” of  $F$ .

Important here: the parts  $F(G_i)$  of  $F$  do not overlap, so that we have the situation of *divide-and-conquer* (on the graph side(!)).

More specifically, for the SAT problems it is natural to require

$$\text{sat}(F) = \min_{i \in \{1, \dots, m\}} \text{sat}(F(G_i))$$

i.e.,  $F$  is satisfiable iff all parts  $F(G_i)$  are satisfiable.

So we are happy if  $m > 1$  — alas, in most cases we have  $m = 1$ .

## How to split graphs (and hypergraphs)

“Bist Du nicht willig, so gebrauche ich Gewalt.”

I.e., if  $G(F)$  does not split into components by itself, we must encourage it a bit. For connected graphs  $G$  there are two general possibilities:

- Find a **separating vertex set**  $N \subseteq V(G)$  such that  $G - N$  is disconnected.
- Find a **separating edge set**  $M \subseteq E(G)$  such that  $G \setminus M$  is disconnected.

Actually, there is a third possibility, namely a **mixed separating set** (possibly containing vertices and edges).

Reasonable (and interesting) polynomial-time algorithms exist for computing separating vertex sets and separating edge sets of minimum cardinality (where the latter task is somewhat easier).

## Finding separating sets

So we can search for a *single splitting* one at a time. Either using some “precise method” or using some of the many (heuristic) optimisation methods for finding hypergraph (hyperedge) cuts (for a given hypergraph or its dual).

Prototypical *global methods* for finding separating vertex sets in graphs:

- If  $G$  is planar then we can apply the planar separator theorem for  $G$  and all derived graphs, “typically” reducing the decision complexity  $2^n$  to  $2^{\sqrt{n}}$ .
- We can compute a tree decomposition  $(T, \chi)$  of  $G$ , by which we “typically” can achieve a decision complexity  $n^k$ , where  $k$  is the width of  $(T, \chi)$ .

Under special condition (requiring (much) more than just the decomposition property) we can also achieve FPT (fixed parameter tractability) in  $k$ .

## A general heuristic scheme $\mathcal{HS}$

Let us assume that we have either given now some separating vertex set  $N$  or some separating edge set  $M$  for  $G$ .

We assume that somehow we can efficiently find problem instances  $F_1, \dots, F_s$  such that

- out of a solution for the  $F_1, \dots, F_s$  we can compute (efficiently) a solution for  $F$ ;
- each  $G(F_i)$  embeds into  $G(F) - N$  or  $G(F) \setminus M$ .

So we get a recursive scheme for solving  $F$ .

Our first contribution now is a generic scheme  $\mathcal{HS}$  for comparing

different branchings  
 $(F_1, \dots, F_s)$  and  $(F'_1, \dots, F'_{s'})$ .

The main technique is the “ $\mathcal{T}$ -method” for estimating (branching) tree sizes.

## How to realise graph splits

So we have an overview on how to achieve and exploit splittings on the *graph side* — but how to actually realise such a split on the *problem side* (i.e., what are those  $F_1, \dots, F_s$ ) ?!

(One main point I want to raise in my talk is awareness of the many possibilities we have for exploiting graph decompositions — not just one.)

We either have to remove vertices or (hyper)edges. In the cases we consider they are either labelled with variables or with sets of sub-formulas of the original problem instance (considered as a kind of “logical formula”).

So we have to get rid off either

- a set  $V$  of variables
- or a set  $S$  of sub-problem-instances (for example a set of clauses).

## Intelligent splitting

The situation we are facing is: **Given** a problem instance  $F$  and either a set  $V$  of variables or a set  $S$  of sub-instances of  $F$ , where removal of the vertices/edges corresponding to  $V/S$  splits  $G(F)$  into components, **find** a way to eliminate  $V$  resp.  $S$  such that we can exploit the splitting of  $G(F)$ .

The most natural approach is to consider a set of partial assignments  $\varphi_1, \dots, \varphi_m$  such that

- $\varphi_1, \dots, \varphi_m$  is “complete” (every total assignment is covered);
- application of any  $\varphi_i$  to  $F$  removes  $V$  resp.  $S$ .

In our general situation we should make use of the **concrete problem**  $F$  (*not just*  $G(F)$ ), and

1. extend the  $\varphi_i$  to  $\varphi'_i$  by further satisfiability-equivalent assignments (regarding the whole  $F$ );
2. eliminate  $\varphi'_i$  which lead to inconsistencies (for  $F$ ).

## Splitting on variables or clauses

Given a set  $V$  of variables, the canonical choice for  $\varphi_1, \dots, \varphi_m$  is the the set of all  $2^{|V|}$  total assignments for  $V$  (and then applying the “afterburner”).

More problematic (and interesting(!)) is the case of a set  $S$  of sub-problem-instances. Here we can distinguish three levels of “globalicity” for the choice of  $\varphi_1, \dots, \varphi_m$ :

1. Choosing the set of all  $2^{|\text{var}(S)|}$  total assignments for the variables appearing in  $S$ .
2. Choosing a set of satisfying assignments for  $S$  which covers all satisfying assignments for  $S$  (this is exactly the problem of representing  $S$  as a DNF).
3. Running an “embedded SAT solver” on  $S$  in the context of  $F$ .

## Two decomposable graph representations

The only graph which seems to have been studied in this context is the **variable-interaction**  $\text{vig}(F)$  graph (“Gaifman graph”, “primal graph”, “interaction graph”):

1. the vertices are the variables;
2. two variables are connected if they occur together in some “constraint”.

It is trivial that  $F \mapsto \text{vig}(F)$  is a decomposable graph representation. Furthermore, in a certain sense *because*  $G(F)$  is such a rough picture of  $F$ , actually here the splitting approach can be refined to yield FPT in the treewidth (and related measures).

## A non-trivial decomposable graph representation: The resolution graph

A more accurate picture of a clause-set  $F$  is given by the **conflict graph**  $F \mapsto \text{cg}(F)$ :

1. the vertices are the clauses;
2. two clauses are connected if they clash in at least one variable.

Modulo elimination of pure variables (occurring in only one sign)  $\text{cg}(F)$  is connected iff  $\text{vig}(G)$  is connected. *However*,  $\text{cg}(F)$  offers “in principle” better splitting possibilities.

Finally the strongest representation (which is actually not completely trivial) is the **resolution graph modulo subsumption**  $\text{srg}(F)$ :

1. the vertices are the clauses;
2. two clauses are connected if they clash in *exactly* one variable, and their resolvent is not subsumed by an already existing clause.

## How to use the conflict graph and the resolution graph

The variable-interaction graph  $\text{vig}(F)$  comes with an important “superstructure”, the **variable hypergraph**  $\text{vhg}(F)$ :

1. the vertices are the variables;
2. the hyperedges are the variable sets of clauses.

The variable-interaction graph is the 2-section of the variable hypergraph, and thus the variable hypergraph  $\text{vhg}(F)$  induces a **clique partition** of  $\text{vig}(F)$ : This superstructure seems to be of (great) importance we efficiently finding splittings of  $\text{vig}(F)$ .

Now  $\text{cg}(F)$  also has a superstructure:  $F$  induces a **biclique partition** of  $\text{cg}(F)$ : Every variable corresponds to a complete bipartite subgraph of  $\text{cg}(F)$ .

(BTW, I glossed over the somehow important question of whether we need to use parallel edges or not; when splitting on variables we don't need them.)

## Conclusion

I believe that in the following directions we will find interesting mathematical structure, which also can be used for “really real” applications:

- Can we achieve FPT in the treewidth of the conflict graph or even of the resolution graph?
- Is there a general theory when to achieve FPT, given a decomposable graph decomposition?
- Study  $\mathcal{HS}$  and instantiations experimentally and theoretically.
- Develop a theory of graph decompositions based on biclique partitions.
- Find other interesting decomposable graph decompositions.

END