# Automated Search Tree Generation (2004; Gramm, Guo, Hüffner, Niedermeier)

Falk Hüffner*, Friedrich-Schiller-Universität Jena, theinf1.informatik.uni-jena.de/~hueffner

entry editor: Rolf Niedermeier

**INDEX TERMS:** NP-hard problems, graph modification, search tree algorithms, automated development and analysis of algorithms.

**SYNONYMS:** Automated proofs of upper bounds on the running time of splitting algorithms

## 1 PROBLEM DEFINITION

This problem is concerned with the automated development and analysis of search tree algorithms. Search tree algorithms are a popular way to find optimal solutions to NP-complete problems.[1] The idea is to recursively solve several smaller instances in such a way that at least one branch is a yes-instance iff the original instance is. Typically, this is done by trying all possibilities to contribute to a solution certificate for a small part of the input, yielding a small local modification of the instance in each branch.

For example, consider the NP-complete Cluster Editing problem: can a graph be transformed by adding or deleting up to $k$ edges into a *cluster graph*, that is, a disjoint union of cliques? To give a search tree algorithm for Cluster Editing, one can use the fact that cluster graphs are exactly the graphs that do not contain a $P_3$ (a path of 3 vertices) as induced subgraph. One can thus solve Cluster Editing by finding a $P_3$ and splitting into 3 branches: delete the first edge, delete the second edge, or add the missing edge. By the characterization, whenever no $P_3$ is found, one already has a cluster graph. The original instance has a solution with $k$ modifications iff at least one of the branches has a solution with $k-1$ modifications.

**Analysis** For NP-complete problems, the running time of a search tree algorithm depends up to a polynomial factor only on the size of the search tree, which depends on the number of branches and the reduction in size in each branch. If the algorithm solves a problem of size $s$ and calls itself recursively for problems of sizes $s-d_1, \ldots, s-d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of this recursion. It is known that the size of the search tree is then $O(\alpha^s)$, where the *branching number* $\alpha$ is the only positive real root of the *characteristic polynomial*

$$z^d - z^{d-d_1} - \cdots - z^{d-d_i}, \tag{1}$$

where $d = \max\{d_1, \ldots, d_i\}$. For the simple Cluster Editing search tree algorithm and the size measure $k$, the branching vector is $(1, 1, 1)$ and the branching number is 3, meaning that the running time is up to a polynomial factor $O(3^k)$.

*Partially supported by the Deutsche Forschungsge meinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

[1]For ease of presentation, only decision problems are considered; adaption to optimization problems is straightforward.

**Case Distinction** Often, one can obtain better running times by distinguishing a number of cases of instances, and giving a specialized branching for each case. The overall running time is then determined by the branching number of the worst case. Several publications obtain such algorithms by hand (e. g., a search tree of size $O(2.27^k)$ for Cluster Editing [4]); the topic of this work is how to automate this. That is, the problem is the following:

**Problem 1** (Fast Search Tree Algorithm).
Input*: An NP-hard problem $\mathcal{P}$ and a size measure $s(I)$ of an instance $I$ of $\mathcal{P}$ where instances $I$ with $s(I) = 0$ can be solved in polynomial time.*
Output*: A partition of the instance set of $\mathcal{P}$ into* cases*, and for each case a branching such that the maximum branching number over all branchings is as small as possible.*

Note that this problem definition is somewhat vague; in particular, to be useful, the case an instance belongs to must be recognizable quickly. It is also not clear whether an optimal search tree algorithm exists; conceivably, the branching number can be continuously reduced by increasingly complicated case distinctions.

## 2 KEY RESULTS

Gramm et al. [3] describe a method to obtain fast search tree algorithms for Cluster Editing and related problems, where the size measure is the number of editing operations $k$. To get a case distinction, simply a number of subgraphs is enumerated such that each instance is known to contain at least one of these subgraphs. It is next described how to obtain a branching for a particular case.

A standard way of systematically obtaining specialized branchings for instance cases is to use a combination of a *basic branching* and *data reduction rules*. A basic branching is a typically very simple branching; data reduction rules replace in polynomial time an instance with a smaller, solution-equivalent instance. Applying this to Cluster Editing first requires a small modification of the problem: one considers an *annotated* version, where an edge can be marked as *permanent* and a non-edge can be marked as *forbidden*. Any such annotated vertex pair cannot be edited anymore. For a pair of vertices, the basic branching then branches into two cases: permanent or forbidden (one of these options will require an editing operation). The reduction rules are: if two permanent edges are adjacent, the third edge of the triangle they induce must also be permanent; and if a permanent and a forbidden edge are adjacent, the third edge of the triangle they induce must be forbidden.

Figure 1 shows an example branching derived in this way.

Using a refined method of searching the space of all possible cases to distinguish and all branchings for a case, Gramm et al. [3] derive a number of search tree algorithms for graph modification problems.

## 3 APPLICATIONS

Gramm et al. [3] apply automated generation of search tree algorithms to several graph modification problems (see also Table 1). Further, Hüffner [5] demonstrates an application to Dominating Set on graphs with maximum degree 4, where the size measure is the size of the dominating set.

Fedin and Kulikov [2] examine variants of SAT; however, their framework is limited in that it only proves upper bounds for a fixed algorithm instead of generating algorithms.

Skjernaa [6] also presents results on variants of SAT. His framework does not require user-provided data reduction rules, but determines reductions automatically.

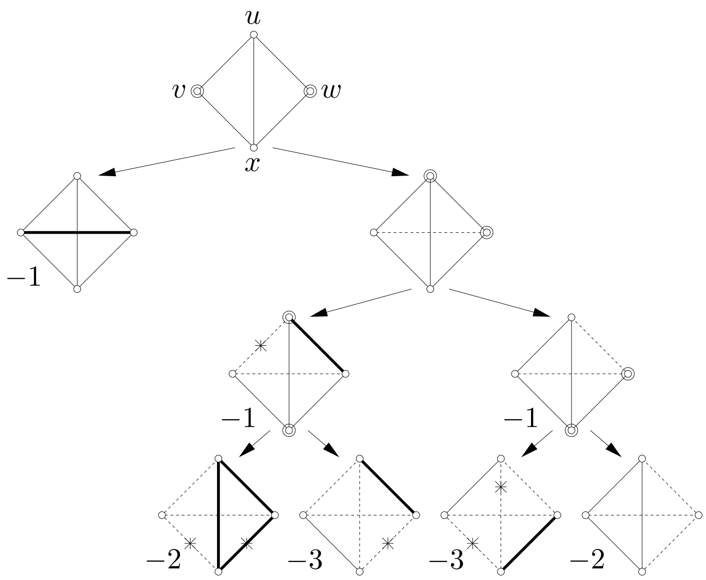


Figure 1: Branching for a CLUSTER EDITING case using only basic branching on vertex pairs (*double circles*), and applications of the reduction rules (*asterisks*). Permanent edges are marked *bold*, forbidden edges *dashed*. The *numbers* next to the subgraphs state the change of the problem size $k$. The branching vector is $(1, 2, 3, 3, 2)$, corresponding to a search tree size of $O(2.27^k)$.

| Problem | Trivial | Known | New |
|---|---|---|---|
| CLUSTER EDITING | 3 | 2.27 | 1.92 [3] |
| CLUSTER DELETION | 2 | 1.77 | 1.53 [3] |
| CLUSTER VERTEX DELETION | 3 | 2.27 | 2.26 [3] |
| BOUNDED DEGREE DOMINATING SET | 4 | | 3.71 [5] |
| X3SAT, size measure $m$ | 3 | 1.1939 | 1.1586 [6] |
| $(n, 3)$-MAXSAT, size measure $m$ | 2 | 1.341 | 1.2366 [2] |
| $(n, 3)$-MAXSAT, size measure $l$ | 2 | 1.1058 | 1.0983 [2] |

Table 1: Summary of search tree sizes where automation gave improvements. "Known" is the size of the best previously published "hand-made" search tree. For the satisfiability problems, $m$ is the number of clauses and $l$ is the length of the formula.

# 4 OPEN PROBLEMS

The analysis of search tree algorithms can be much improved by describing the "size" of an instance by more than one variable, resulting in multivariate recurrences [1]. It is open to introduce this technique into an automation framework.

It has frequently been reported that better running time bounds through a large number of cases to distinguish do not necessarily lead to a speedup, but in fact can slow a program down. A careful investigation of the tradeoffs involved and a corresponding adaption of the automation frameworks is an open task.

# 5 EXPERIMENTAL RESULTS

Gramm et al. [3] and Hüffner [5] report search tree sizes for several NP-complete problems. Further, Fedin and Kulikov [2] and Skjernaa [6] report on variants of satisfiability. Table 1 summarizes the results.

## 6 CROSS REFERENCES

Vertex Cover Search Trees (2001; Chen, Kanj, Jia)

## 7 RECOMMENDED READING


[1] D. Eppstein, *Quasiconvex analysis of backtracking algorithms*, in Proc. 15th SODA, ACM/SIAM, 2004, pp. 788–797.

[2] S. S. Fedin and A. S. Kulikov, *Automated proofs of upper bounds on the running time of splitting algorithms*, Journal of Mathematical Sciences, 134 (2006), pp. 2383–2391. Improved results at `http://logic.pdmi.ras.ru/~kulikov/autoproofs.html`.

[3] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier, *Automated generation of search tree algorithms for hard graph modification problems*, Algorithmica, 39 (2004), pp. 321–347.

[4] ——, *Graph-modeled data clustering: Exact algorithms for clique generation*, Theory of Computing Systems, 38 (2005), pp. 373–392.

[5] F. Hüffner, *Graph Modification Problems and Automated Search Tree Generation.* Diplomarbeit, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2003.

[6] B. Skjernaa, *Exact Algorithms for Variants of Satisfiability and Colouring Problems*, PhD thesis, University of Aarhus, Department of Computer Science, 2004.