Pushdown Reachability with Constant Treewidth

Krishnendu Chatterjee[a], Georg Osang[a]

[a]*IST Austria (Institute of Science and Technology Austria) Klosterneuburg, Austria*

Abstract

We consider the problem of reachability in pushdown graphs. We study the problem for pushdown graphs with constant treewidth. Even for pushdown graphs with treewidth 1, for the reachability problem we establish the following: (i) the problem is PTIME-complete, and (ii) any subcubic algorithm for the problem would contradict the k-clique conjecture and imply faster combinatorial algorithms for cliques in graphs.

Keywords: pushdown systems, reachability, constant tree-width graphs

1. Introduction

In this work we consider the problem of reachability in pushdown graphs with the restriction of constant treewidth. Our main results are negative results which shows that the restriction does not help to obtain better computational or algorithmic complexity. We first describe the notion of treewidth and its relevance.

Treewidth of graphs. A very well-known concept in graph theory is the notion of *treewidth* of a graph, which is a measure of how similar a graph is to a tree (a graph has treewidth 1 precisely if it is a tree) [20]. The treewidth of a graph is defined based on a *tree decomposition* of the graph [15]. Beyond the mathematical elegance of the treewidth property for graphs, there are many classes of graphs which arise in practice and have constant treewidth. The most important example is that the control flow graphs of goto-free programs for many programming languages are of constant treewidth [21], and it was also shown in [14] that typically all Java programs have constant treewidth. For many other applications see the surveys [4, 5]. The constant treewidth property of graphs has also played an important role in logic and verification; for example, MSO (Monadic Second Order logic) queries can be solved in polynomial time [11] (also in log-space [12]) for constant-treewidth graphs; parity games on graphs with constant treewidth can be solved in polynomial time [19]; and there exist faster algorithms for probabilistic models (like Markov decision processes) [8]. Moreover, recently it has been shown that the constant treewidth property is also useful for interprocedural analysis [7] and intraprocedural analysis of concurrent programs [6].

The relevant questions. First, the constant treewidth property has played a significant role for finite graphs. For example, for the basic reachability problem, while for general graphs the complexity is NL-complete (non-deterministic log-space complete), for constant treewidth graphs the problem is in DLOGSPACE (deterministic log-space) [12]. Second, the constant treewidth property has been exploited to obtain faster algorithms for several problems in program analysis, such as [7, 6]. Since the control-flow graphs of a wide class of programs have constant treewidth, a very relevant question is whether for general pushdown graphs faster algorithms or better complexity results can be established with the assumption of constant treewidth. More precisely, we consider the reachability problem for pushdown graphs. In general, the problem is (i) PTIME-complete and (ii) can be solved in cubic time[1]. Our goal is to investigate whether the constant treewidth restriction can improve the algorithmic or computational complexity.

Our results. We establish the following negative results.

1. We show even for pushdown graphs with treewidth 1 the reachability problem is PTIME-complete. Thus in contrast to finite graphs, where the constant treewidth restriction decreases the complexity from non-deterministic to deterministic logspace, for pushdown graphs, the constant treewidth restriction does not help to improve the computational complexity for pushdown graphs.
2. We show that for pushdown graphs with treewidth 1 the reachability problem is k-clique hard, in the sense that any combinatorial sub-cubic algorithm would contradict the k-clique conjecture (i.e., a sub-cubic algorithm would imply a faster combinatorial algorithm for finding k-cliques in graphs). Thus our results show that even for pushdown graphs with treewidth 1 obtaining better algorithms would require a major break-through.

2. Preliminaries

We shall start by introducing the above notions formally and the discuss relevant known results and conjectures.

2.1. Basic Definitions

Treewidth. Given a (directed or undirected) graph or a multi-graph $G = (V, E)$, a *tree-decomposition* is a tree $\mathrm{Tree}(G) = (V_T, E_T)$ whose vertices $B_i, 1 \leq i \leq m$ are subsets of V, called *bags*, such that the following holds:

1. $\bigcup_{B \in V_T} B = V$.
2. For every edge $(u, v) \in E$ there is a bag $B \in V_T$ with $u, v \in B$.

[1]Improvement by logarithmic factors is also possible [9]

3. For each $v \in V$, the set $\{B : v \in B\}$ of bags containing v is connected in $\mathrm{Tree}(G)$.

The width of a tree decomposition is the size of the largest bag minus 1. The *tree-width* of a graph G is the lowest width achieved by a tree decomposition of G.
Note that given a multi-graph G, a tree-decomposition of its simplification (i.e. all loops removed and multiple edges replaced with single edges) is still a valid tree-decomposition for G. We call a graph *path-like* if its simplification is a path. Furthermore note that the tree-width of any tree is 1, and consequently the tree-width of any path-like graph is also 1.

Context-free grammars. A context-free grammar (CFG) $\mathcal{G}$ is a tuple (V, Σ, P, S) where V is the set of non-terminal symbols, Σ is the set of terminal symbols (disjoint from V), $P \subset V \times (V \cup \Sigma)^*$ is the set of productions and $S \in V$ is the start symbol. If (N, u) is a production we also write $N \to u$. For two words w, w' over $V \cup \Sigma$ we write $w \Rightarrow w'$ if w' can be obtained by replacing one of its non-terminals N with a string u where $N \to u$ is a production. We write $w \Rightarrow^* w'$ if w' can be obtained from w by a sequence of such replacements and we say w' can be derived from w. The language generated by $\mathcal{G}$ is defined as the set $\{w : S \Rightarrow^* w\}$ of words derivable from the start symbol and is denoted as $L(\mathcal{G})$. The context-free membership problem is defined as follows:

Context-free membership problem (CFG membership).
Input: A context-free grammar $\mathcal{G} = (V, \Sigma, P, S)$ and a word $w \in \Sigma^*$.
Output: YES if $w \in L(\mathcal{G})$, NO otherwise.

Pushdown systems. A pushdown system (PDS) is a tuple $(Q, \Gamma, \Delta, q_s, \$)$ where Q is the set of states, $q_s \in Q$ is the start state, Γ is the stack alphabet, $\$ \in \Gamma$ is the initial stack symbol and $\Delta \subset (Q \times \Gamma) \times (Q \times \Gamma^*)$ is the transition relation. If $((q, \gamma), (q', w'))$ is a transition we also write $(q, \gamma) \to (q', w')$. The associated pushdown graph is the directed graph with vertex set Q and for each transition $(q, \gamma) \to (q', w')$ an edge from q to q'. When talking about the tree-width of a PDS or a PDS being path-like, we are referring to the associated pushdown graph. A configuration of a PDS is pair $\langle q, w \rangle$ of a state together with the stack content $w \in \Gamma^*$. Performing a transition $(q, \gamma) \to (q', w')$, when currently in state q with top of stack symbol γ means going into state q' and replacing the top of stack symbol with the word $w' \in \Gamma^*$.
We say a state q' is reachable from another state q if there is a series of transitions leading from $\langle q, \$ \rangle$ to some configuration $\langle q', w' \rangle$ for some arbitrary stack content w'. We say q' is same-context reachable from q if $w' = \$$. This allows us to define the pushdown reachability problem.

Pushdown reachability problem.
Input: A pushdown system $\mathcal{P} = (Q, \Gamma, \Delta, q_s, \$)$ and two states $q, q' \in Q$.
Output: YES if q' is reachable from q, NO otherwise.

A clique is a graph such that there is an edge between every pair of vertices. We will need the following two problems whose complexity is related to CFG membership:

k-clique problem (k-clique).
Input: A graph G and a positive integer k.
Output: YES if G contains a clique on k vertices as subgraph, NO otherwise.

Boolean matrix multiplication problem (BMM).
Input: Two square $n \times n$ matrices A and B with boolean entries.
Output: The product AB of the two matrices.

We define the boolean matrix multiplication exponent ω to be the smallest value such that BMM for two $n \times n$ matrices is in $O(n^{\omega+\varepsilon})$ for all $\varepsilon > 0$.

2.2. Known Results and Conjectures

It is known that $\omega < 2.376$ due to Coppersmith and Winograd [10]. While it is conjectured by some that $\omega = 2$, there have been only very minor improvements on the upper bound during the last 25 years. Furthermore these algorithms tend to have large hidden constants and in practice "combinatorial" algorithms [2] are more widely used. None of these run significantly faster than cubic however [3], putting into question whether any significant improvement can be made:

Conjecture 2.1 (Combinatorial Boolean Matrix Multiplication Conjecture)**.** *There is no combinatorial algorithm for BMM for two $n \times n$ matrices that runs in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$.*

A classical algorithm by Valiant [22] reduces CFG membership to BMM thus giving an algorithm for CFG membership in $O(n^{\omega})$. Lee introduced in [17] a stronger version of CFG parsing called c-parsing where the parser has to provide some additional data about parsing substrings of the input string. She showed that BMM can be reduced to c-parsing such that c-parsing in $O(gn^{3-\varepsilon})$ where g is the size of the CFG and n the length of the word to parse implies BMM in $O(n^{3-\varepsilon/3})$ (for two $n \times n$ matrices). A drawback of this reduction is that the grammar grows quadratically in n. Abboud, Backurs and Williams [1] provided

[2]The term "combinatorial" has no formal definition but see [3] for a brief discussion of the term.

a constant size CFG $\mathcal{G}$ and a scheme which for given k transforms a graph G on n nodes into a string of length $O(k^2 n^{k+1})$ such that this string is in the language generated by $\mathcal{G}$ if and only if G has a $3k$-clique. This implies the following theorem:

Theorem 2.2. *If CFG membership can be solved in $T(n)$ time, then the k-clique problem on any graph of n nodes can be solved in $O(T(n^{k/3+1}))$ time, for any constant $k \geq 3$ divisible by 3.*

The fastest known algorithm for k-clique runs in $O(n^{k\omega/3})$ for k divisible by 3 [18]. The fastest known combinatorial algorithm known for k-clique runs in $O(n^k / \log^k n)$ and therefore is only better than the naive algorithm by a polylog factor [23]. The lack of better algorithms, as mentioned in [1], motivates the following conjecture:

Conjecture 2.3 (k-clique Conjecture)**.** *Let G be a graph on n vertices and k an integer.*
There is no algorithm solving k-clique running in $O(n^{k\omega/3-\varepsilon})$ for $\varepsilon > 0$.
There is no combinatorial algorithm solving k-clique running in $O(n^{k-\varepsilon})$ for $\varepsilon > 0$.

Regarding the pushdown reachability problem, Finkel et al. showed that the set of reachable states in a PDS $\mathcal{P}$ from an initial state with a given initial stack content can be computed in $O(|\mathcal{P}|^3)$ [13]. So in particular the pushdown reachability problem is solvable in cubic time. Alur et al. introduced in [2] the notion of recursive state machines (RSM). They showed that every PDS can be transformed into an equivalent RSM of the same asymptotic size and vice versa, and they provide a cubic time algorithm to compute reachability in RSMs.

3. Results

We will give a reduction from CFG membership to pushdown reachability where the resulting pushdown system is path-like. This reduction can be done in DLOGSPACE or in linear-time.

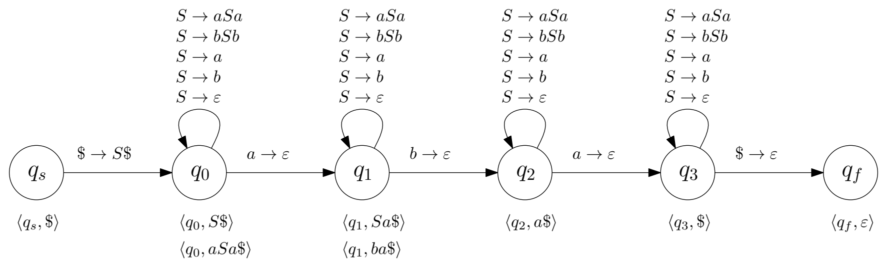
3.1. Reduction

The reduction is non-technical but has important implications about push-down reachability.

Theorem 3.1. *For a context-free grammar $\mathcal{G}$ with $|\mathcal{G}| = O(1)$ and a word w of length n, there is a path-like PDS $\mathcal{P}$ of size $|\mathcal{P}| = O(n)$ with start state q_s and a designated state q_f such that $q_s \to q_f$ if and only if $w \in L(\mathcal{G})$. $\mathcal{P}$ can be constructed in time $O(n)$ or alternatively with working memory $O(\log n)$.*

Proof. Let $\mathcal{G} = (V, \Sigma, P, S)$ be a context-free grammar. We shall henceforth assume that the size of $\mathcal{G}$ is constant. We have a word $w \in \Sigma^*$ of length n and we want to determine if $w \in L(\mathcal{G})$.

We will construct a path-like PDS $\mathcal{P}$ and show that there's a state q_f which is only reachable from the start state q_s if and only if $S \Rightarrow^* w$. Let $\mathcal{P} = (Q, \Gamma, \Delta, q_s, \$)$ where $Q = \{q_0, q_1, \ldots, q_n\} \cup \{q_s, q_f\}$ is the set of states, q_s is the start state, $\Gamma = \Sigma \cup V \cup \{\$\}$ is the stack alphabet and \$ is the initial stack symbol and $\Delta \subset (Q \times \Gamma) \times (Q \times \Gamma^*)$ is the transition relation that we'll define in the following. For each production $N \rightarrow \gamma$ from P where γ is a word over $V \cup \Sigma$ and for each state q_i, $0 \le i \le n$, we add a transition $(q_i, N) \rightarrow (q_i, \gamma)$. Furthermore, for each $0 \le i < n$ we add a transition $(q_i, w_i) \rightarrow (q_{i+1}, \varepsilon)$ where w_i is the ith character of w indexed from 0. Finally we add two transitions $(q_s, \$) \rightarrow (q_0, S\$)$ and $(q_n, \$) \rightarrow (q_f, \varepsilon)$. These transitions ensure that we can transition into the final state q_f if and only if only the \$ symbol is left on the stack. The size of the resulting PDS is $O(n)$ as $\mathcal{G}$ was of constant size, and its simplification is a path. See Figure 1 for an example.



$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow a$
$S \rightarrow b$
$S \rightarrow \varepsilon$
q_s
$\$ \rightarrow S\$$
q_0
$a \rightarrow \varepsilon$
q_1
$b \rightarrow \varepsilon$
q_2
$a \rightarrow \varepsilon$
q_3
$\$ \rightarrow \varepsilon$
q_f
$\langle q_s, \$\rangle$
$\langle q_0, S\$\rangle$
$\langle q_0, aSa\$\rangle$
$\langle q_1, Sa\$\rangle$
$\langle q_1, ba\$\rangle$
$\langle q_2, a\$\rangle$
$\langle q_3, \$\rangle$
$\langle q_f, \varepsilon\rangle$

Figure 1: Example of the reduction for the palindrome grammar $S \rightarrow aSa|bSb|a|b|\varepsilon$ with the word aba to test membership for. Transitions $(q, \gamma) \rightarrow (q', w)$ are depicted as edges from q to q' with label $\gamma \rightarrow w$. At the bottom is a sequence of configurations of the PDS leading to an accepting state, read top to bottom, left to right.

Claim 1: q_f is reachable from q_s if and only if $w \in L(\mathcal{G})$.
Assume q_f is reachable from q_s, and we have a valid sequence of transitions starting in q_s with only \$ on the stack and finishing in q_f. After the jth transition, when in state q_i for some $0 \le i \le n$, we shall call the word consisting of the first i characters of w with the current stack content appended to it and the final \$ sign removed the *input-stack word* and denote it as W_j. We shall first show by induction that $S \Rightarrow^* W_j$ for all $j \ge 1$. We have that $W_1 = S$ as after the first transition we are in q_0 and the stack only consists of S. Now assume that $S \Rightarrow^* W_j$. If we perfom a state-changing transition from some q_i to q_{i+1}, then w_i must be at the top of the stack and is removed during the transition. But as we advance one state, w_i is now the last of the first $i+1$ characters of w and thus $W_{j+1} = W_j$. So we also get that $S \Rightarrow^* W_{j+1}$. Similarly, if we have a transition remaining in the same state, then we're replacing the non-terminal symbol at the top of the stack with a string in accordance to one of the production rules of $\mathcal{G}$ while leaving the rest unchanged. Therefore we have $W_j \Rightarrow W_{j+1}$ and thus together with the induction hypothesis it follows that $S \Rightarrow^* W_{j+1}$. Now q_f is only reachable if we can reach q_n with only \$ on the stack. But then the input-stack word is w once we reach q_n, and thus $S \Rightarrow^* w$ and $w \in L(\mathcal{G})$.
Conversely if $w \in L(\mathcal{G})$, then there is a sequence of productions that yield

w starting from S. In particular, as $\mathcal{G}$ is context-free we can reorder these productions such that always the first non-terminal is being replaced. Following this sequence of productions with their corresponding same-state transitions in the PDS, and using the state-changing transitions whenever a terminal symbol is at the top of the stack, we can reach q_n in the PDS with nothing but the \$ symbol left on the stack, and thus also q_f is reachable.
Also note that q_f is reachable from q_s if and only if q_n is same-context reachable from q_s.
Claim 2: This reduction can be done alternatively in DLOGSPACE or linear time.
We only need one pointer to the input tape, and one counting variable whose range is $O(n)$, each of which takes $O(\log n)$ space.
Similarly, we can construct $\mathcal{P}$ by passing over w only once and for each character of w only do a constant number of operations as $|\mathcal{G}|$ is constant.

□

3.2. Consequences

We will outline two implications of this result.
Recall that pushdown reachability can be solved in cubic time, and thus is in P. It is known that CFG-membership is P-complete under DLOGSPACE-reductions [16]. As we have just shown that CFG membership can be reduced to pushdown reachability in DLOGSPACE, it follows that pushdown reachability is also P-complete and we get the following corollary:

Corollary 3.2. *Pushdown reachability is P-complete under DLOGSPACE-reductions, even if restricted to path-like PDSs.*

Similarly, as our reduction is in linear time, an algorithm for pushdown reachability in running in $T(n)$ time implies an algorithm for CFG membership in $O(T(n))$. Together with Theorem 2.2 we get that the k-clique problem on any graph of n nodes can be solved in $O(T(n^{k/3+1}))$ time, for any constant $k \geq 3$ divisible by 3. If $T(n) = O(n^{3-\delta})$ for some δ, this in particular would imply an algorithm for k-clique running in $O(n^{k+(3-\delta(k/3+1))}) = O(n^{k-\varepsilon})$ for some $\varepsilon > 0$ if k is chosen sufficiently large. Similarly, $T(n) = O(n^{\omega-\delta})$ would imply an algorithm for k-clique running in $O(n^{k\omega/3-\varepsilon})$ for some $\varepsilon > 0$ if k is chosen sufficiently large. With this observation we get the following result:

Corollary 3.3. *If the k-clique conjecture holds, then there is no algorithm for push-down reachability running in $O(n^{\omega-\varepsilon})$ for $\varepsilon > 0$, and no combinatorial algorithm for push-down reachability running in $O(n^{3-\varepsilon})$ for $\varepsilon > 0$, even if the problem is restricted to path-like PDSs.*

4. Conclusions

Unlike for a variety of other problems, our results imply that the treewidth restriction by itself cannot be exploited for faster reachability algorithms for pushdown systems. An interesting direction for future work would be to consider other restrictions, along with the treewidth restriction, that arise in program analysis and can be exploited to develop faster algorithms for pushdown systems.

5. Acknowledgements

The research was partially supported by the Austrian Science Fund (FWF): P23499-N23, S11407-N23 (RiSE/SHiNE), and an ERC Start Grant (279307: Graph Games).

6. References

[1] Amir Abboud, Arturs Backurs, and Virginia V. Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117, 2015.

[2] Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas Reps, and Mihalis Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, July 2005.

[3] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 745–754. IEEE, 2009.

[4] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 1993.

[5] Hans L. Bodlaender. Discovering treewidth. In *SOFSEM: Theory and Practice of Computer Science*, LNCS. Springer, 2005.

[6] Krishnendu Chatterjee, Amir K. Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Algorithms for algebraic path properties in concurrent systems of constant treewidth components. In *POPL*, pages 733–747, 2016.

[7] Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Andreas Pavlogiannis, and Prateesh Goyal. Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 97–109, New York, NY, USA, 2015. ACM.

[8] Krishnendu Chatterjee and Jakub Lacki. Faster algorithms for Markov decision processes with low treewidth. In *CAV*, LNCS. Springer, 2013.

[9] Swarat Chaudhuri. Subcubic algorithms for recursive state machines. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 159–169, New York, NY, USA, 2008. ACM.

[10] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. Computational algebraic complexity editorial.

[11] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 1990.

[12] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *FOCS*. IEEE Computer Society, 2010.

[13] Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). *Electronic Notes in Theoretical Computer Science*, 9:27 – 37, 1997. Infinity'97, Second International Workshop on Verification of Infinite State Systems.

[14] Jens Gustedt, Ole A. Mæhle, and Jan A. Telle. The treewidth of Java programs. In *Algorithm Engineering and Experiments*, LNCS. Springer, 2002.

[15] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 1976.

[16] Neil D. Jones and William T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105 – 117, 1976.

[17] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, January 2002.

[18] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[19] Jan Obdrzálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, LNCS. Springer, 2003.

[20] Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 1984.

[21] Mikkel Thorup. All Structured Programs Have Small Tree Width and Good Register Allocation. *Inf. Comput.*, 1998.

[22] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308 – 315, 1975.

[23] Virginia V. Williams. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4):254–257, 2009.