Graph and String Parameters: Connections Between Pathwidth, Cutwidth and the Locality Number

Katrin Casel
Hasso Plattner Institute, University of Potsdam, Germany
Katrin.Casel@hpi.de

Joel D. Day
Department of Computer Science, Loughborough University, UK
J.Day@lboro.ac.uk

Pamela Fleischmann
Department of Computer Science, Kiel University, Germany
fpa@informatik.uni-kiel.de

Tomasz Kociumaka
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
Institute of Informatics, University of Warsaw, Poland
kociumaka@mimuw.edu.pl

Florin Manea
Department of Computer Science, Kiel University, Germany
flm@informatik.uni-kiel.de

Markus L. Schmid
Trier University, Germany
mlschmid@mlschmid.de

Abstract

We investigate the locality number, a recently introduced structural parameter for strings (with applications in pattern matching with variables), and its connection to two important graph-parameters, cutwidth and pathwidth. These connections allow us to show that computing the locality number is NP-hard but fixed-parameter tractable (when the locality number or the alphabet size is treated as a parameter), and can be approximated with ratio $O(\sqrt{\log \mathsf{opt}} \log n)$. As a by-product, we also relate cutwidth via the locality number to pathwidth, which is of independent interest, since it improves the best currently known approximation algorithm for cutwidth. In addition to these main results, we also consider the possibility of greedy-based approximation algorithms for the locality number.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Mathematics of computing → Combinatorics on words; Theory of computation → Approximation algorithms analysis

Keywords and phrases Graph and String Parameters, NP-Completeness, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.109

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version A full version of the paper is available at `https://arxiv.org/abs/1902.10983`.

Funding *Tomasz Kociumaka*: Supported by ISF grants no. 824/17 and 1278/16 and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064). *Florin Manea*: Supported by the DFG grant MA 5725/2-1.

1 Introduction

Graphs, on the one hand, and strings, on the other, are two different types of data objects and they have certain particularities. Graphs seem to be more popular in fields like classical and parameterised algorithms and complexity (due to the fact that many natural graph problems are intractable), while fields like formal languages, pattern matching, verification or compression are more concerned with strings. Moreover, both the field of graph algorithms as well as string algorithms are well established and provide rich toolboxes of algorithmic techniques, but they differ in that the former is tailored to computationally hard problems (e.g., the approach of treewidth and related parameters), while the latter focuses on providing efficient data-structures for near-linear-time algorithms. Nevertheless, it is sometimes possible to bridge this divide, i.e., by "flattening" a graph into a sequential form, or by "inflating" a string into a graph, to make use of respective algorithmic techniques otherwise not applicable. This paradigm shift may provide the necessary leverage for new algorithmic approaches.

In this paper, we are concerned with certain structural parameters (and the problems of computing them) for graphs and strings: the *cutwidth* $\mathsf{cw}(G)$ of a graph G (i.e., the maximum number of "stacked" edges if the vertices of a graph are drawn on a straight line), the *pathwidth* $\mathsf{pw}(G)$ of a graph G (i.e., the minimum width of a tree decomposition the tree structure of which is a path), and the *locality number* $\mathsf{loc}(\alpha)$ of a string α (explained in more detail in the next paragraph). By Cutwidth, Pathwidth and Loc, we denote the corresponding decision problems and with the prefix Min, we refer to the minimisation variants. The two former graph-parameters are very classical. Pathwidth is a simple (yet still hard to compute) subvariant of treewidth, which measures how much a graph resembles a path. The problems Pathwidth and MinPathwidth are intensively studied (in terms of exact, parameterised and approximation algorithms) and have numerous applications (see the surveys and textbook [10, 34, 8]). Cutwidth is the best-known example of a whole class of so-called *graph layout problems* (see the survey [17, 39] for detailed information), which are studied since the 1970s and were originally motivated by questions of circuit layouts.

The locality number is rather new and we shall discuss it in more detail. A word is k-local if there exists an order of its symbols such that, if we *mark* the symbols in the respective order (which is called a *marking sequence*), at each stage there are at most k contiguous blocks of marked symbols in the word. This k is called the *marking number* of that marking sequence. The *locality number* of a word is the smallest k for which that word is k-local, or, in other words, the minimum marking number over all marking sequences. For example, the marking sequence $\sigma = (\mathtt{x}, \mathtt{y}, \mathtt{z})$ marks $\alpha = \mathtt{xyxyzxz}$ as follows (marked blocks are illustrated by overlines): $\mathtt{xyxyzxz}$, $\overline{\mathtt{x}}\mathtt{y}\overline{\mathtt{x}}\mathtt{yz}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxy}}\mathtt{z}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxyzxz}}$; thus, the marking number of σ is 3. In fact, all marking sequences for α have a marking number of 3, except $(\mathtt{y}, \mathtt{x}, \mathtt{z})$, for which it is 2: $\mathtt{x}\overline{\mathtt{y}}\mathtt{x}\overline{\mathtt{y}}\mathtt{zxz}$, $\overline{\mathtt{xyxy}}\mathtt{z}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxyzxz}}$. Thus, the locality number of α, denoted by $\mathsf{loc}(\alpha)$, is 2.

The locality number has applications in pattern matching with variables [14]. A *pattern* is a word that consists of *terminal symbols* (e.g., $\mathtt{a}, \mathtt{b}, \mathtt{c}$), treated as constants, and *variables* (e.g., $x_1, x_2, x_3, \ldots$). A pattern is mapped to a word by substituting the variables by strings of terminals. For example, $x_1 x_1 \mathtt{bab} x_2 x_2$ can be mapped to $\mathtt{acacbabcc}$ by the substitution $(x_1 \to \mathtt{ac}, x_2 \to \mathtt{c})$. Deciding whether a given pattern matches (i.e., can be mapped to) a given word is one of the most important problems that arise in the study of patterns with variables (note that the concept of patterns with variables arises in several different domains like combinatorics on words (word equations [30], unavoidable patterns [36]), pattern matching [1], language theory [2], learning theory [2, 19, 38, 42, 31, 22], database theory [7], as well as in practice, e.g., extended regular expressions with backreferences [26, 27, 44, 28], used in programming languages like Perl, Java, Python, etc.). Unfortunately, the *matching problem* is NP-complete [2] in general (it is also NP-complete for strongly restricted variants [23, 21] and also intractable in the parameterised setting [24]). As demonstrated in [43], for the matching problem a paradigm shift as sketched in the first paragraph above yields a very promising algorithmic approach. More precisely, any class of patterns with bounded treewidth (for suitable graph representations) can be matched in polynomial-time. However, computing (and therefore algorithmically exploiting) the treewidth of a pattern is difficult (see the discussion in [21, 43]), which motivates more direct string-parameters that bound the treewidth and are simple to compute (virtually all known structural parameters that lead to tractability [14, 21, 43, 45] are of this kind (the efficiently matchable classes investigated in [15] are one of the rare exceptions)). This also establishes an interesting connection between ad-hoc string parameters and the more general (and much better studied) graph parameter treewidth. The locality number is a simple parameter directly defined on strings, it bounds the treewidth and the corresponding marking sequences can be seen as instructions for a dynamic programming algorithm. However, compared to other "tractability-parameters", it seems to cover best the treewidth of a string, but whether it can be efficiently computed is unclear.

In this paper, we investigate the problem of computing the locality number and, by doing so, we establish an interesting connection to the graph parameters cutwidth and pathwidth with algorithmic implications for approximating cutwidth. In the following, we first discuss related results in more detail and then outline our respective contributions.

Known Results and Open Questions. For Loc, only exact exponential-time algorithms are known and whether it can be solved in polynomial-time, or whether it is at least fixed-parameter tractable is mentioned as open problems in [14]. Approximation algorithms have not yet been considered. Addressing these questions is the main purpose of this paper.

Pathwidth and Cutwidth are NP-complete, but fixed-parameter tractable with respect to parameter $\mathsf{pw}(G)$ or $\mathsf{cw}(G)$, respectively (even with "linear" fpt-time $g(k)\,\mathrm{O}(n)$ [9, 11, 47]). With respect to approximation, their minimisation variants have received a lot of attention, mainly because they yield (like many other graph parameters) general algorithmic approaches for numerous graph problems, i.e., a good linear arrangement or path-decomposition can often be used for a dynamic programming (or even divide and conquer) algorithm. More generally speaking, pathwidth and cutwidth are related to the more fundamental concepts of small balanced vertex or edge separators for graphs (i.e., a small set of vertices (or edges, respectively) that, if removed, divides the graph into two parts of roughly the same size. More precisely, $\mathsf{pw}(G)$ and $\mathsf{cw}(G)$ are upper bounds for the smallest balanced *vertex* separator of G and the smallest balanced *edge* separator of G, respectively (see [20] for further details and explanations of the algorithmic relevance of balanced separators). The best known approximation algorithms for MinPathwidth and MinCutwidth (with approximations ratios of $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$ and $\mathrm{O}(\log^2(n))$, respectively) follow from approximations of vertex separators (see [20]) and edge separators (see [35]), respectively.

Our Contributions. There are two natural approaches to represent a word α over alphabet Σ as a graph $G_\alpha = (V_\alpha, E_\alpha)$: (1) $V_\alpha = \{1, 2, \ldots, |\alpha|\}$ and the edges are somehow used to represent the actual symbols, or (2) $V_\alpha = \Sigma$ and the edges are somehow used to represent the positions of α. We present a reduction of type (2) such that $|E_\alpha| = \mathrm{O}(|\alpha|)$ and $\mathsf{cw}(G_\alpha) = 2\,\mathsf{loc}(\alpha)$, and a reduction of type (1) such that $|E_\alpha| = \mathrm{O}(|\alpha|^2)$ and $\mathsf{loc}(\alpha) \leq \mathsf{pw}(G_\alpha) \leq 2\,\mathsf{loc}(\alpha)$. Since these reductions are parameterised reductions and also allow transferring approximation results, we conclude that LOC is fixed-parameter tractable if parameterised by $|\Sigma|$ or by the locality number (answering the respective open problem from [14]), and also that there is a polynomial-time $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for MINLOC.

In addition, we also show a way to represent an arbitrary multi-graph $G = (V, E)$ by a word α_G over alphabet V, of length $|E|$ and with $\mathsf{cw}(G) = \mathsf{loc}(\alpha)$. This describes a Turing-reduction from CUTWIDTH to LOC which also allows to transfer approximation results between the minimisation variants. As a result, we can conclude that LOC is NP-complete (which solves the other open problem from [14]). Finally, by plugging together the reductions from MINCUTWIDTH to MINLOC and from MINLOC to MINPATHWIDTH, we obtain a reduction which transfers approximation results from MINPATHWIDTH to MINCUTWIDTH, which yields an $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for MINCUTWIDTH. This improves, to our knowledge for the first time since 1999, the best approximation for CUTWIDTH from [35].

To our knowledge, this connection between cutwidth and pathwidth has not yet been reported in the literature so far. This is rather surprising since CUTWIDTH and PATHWIDTH have been jointly investigated in the context of exact and approximation algorithms, especially in terms of balanced vertex and edge separators. More precisely, the approximation of pathwidth and cutwidth follows from the approximation of vertex and edge separators, respectively, and the approximation of vertex separators usually relies on edge separators: the edge separator approximation from [35] can be used as a black-box for vertex separator approximation, and the best vertex separator algorithm from [20] uses a technique for computing edge separators from [4] as component. Our improvement, on the other hand, is achieved by going in the opposite direction: we use pathwidth approximation (following from [20]) in order to improve the currently best cutwidth approximation (from [35]). This might be why the reduction from cutwidth to pathwidth has been overlooked in the literature. Another reason might be that this relation is less obvious on the graph level and becomes more apparent if linked via the string parameter of locality, as in our considerations. Nevertheless, since pathwidth and cutwidth are such crucial parameters for graph algorithms, we also translate our locality based reduction into one from graphs to graphs directly.

2 Preliminaries

Basic Definitions. The set of strings (or words) over an alphabet X is denoted by X^*, by $|\alpha|$ we denote the length of a word α, $\mathsf{alph}(\alpha)$ is the smallest alphabet X with $\alpha \in X^*$. A string β is called a *factor* of α if $\alpha = \alpha'\beta\alpha''$; if $\alpha' = \varepsilon$ or $\alpha'' = \varepsilon$, where ε is the empty string, β is a *prefix* or a *suffix*, respectively. For a position j, $1 \leq j \leq |\alpha|$, we refer to the symbol at position j of α by the expression $\alpha[j]$, and $\alpha[j..j'] = \alpha[j]\alpha[j+1]\dots\alpha[j']$, $1 \leq j \leq j' \leq |\alpha|$. For a word α and $x \in \mathsf{alph}(\alpha)$, let $\mathsf{ps}_x(\alpha) = \{i \mid 1 \leq i \leq |\alpha|, \alpha[i] = x\}$ be the set of all positions where x occurs in α. For a word α, let $\alpha^0 = \varepsilon$ and $\alpha^{i+1} = \alpha\alpha^i$ for $i \geq 0$.

Let α be a word and let $X = \mathsf{alph}(\alpha) = \{x_1, x_2, \dots, x_n\}$. A *marking sequence* is an enumeration, or ordering on the letters, and hence may be represented either as an ordered list of the letters or, equivalently, as a bijection $\sigma : \{1, 2, \dots, |X|\} \to X$. Given a word α and a marking sequence σ, the *marking number* $\pi_\sigma(\alpha)$ (of σ with respect to α) is the maximum number of marked blocks obtained while marking α according to σ. We say that α is k-local if and only if, for some marking sequence σ, we have $\pi_\sigma(\alpha) \leq k$, and the smallest k such that α is k-local is the *locality number* of α, denoted by $\mathsf{loc}(\alpha)$. A marking sequence σ with $\pi_\sigma(\alpha) = \mathsf{loc}(\alpha)$ is *optimal* (for α). For a marking sequence $\sigma = (x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(m)})$ and a word α, by *stage i of σ* we denote the word α with exactly positions $\bigcup_{j=1}^{i} \mathsf{ps}_{x_{\sigma(j)}}(\alpha)$ marked.

For a word α, the *condensed form of* α, denoted by $\mathsf{cond}(\alpha)$, is obtained by replacing every maximal factor x^k with $x \in \mathsf{alph}(\alpha)$ by x. For example, $\mathsf{cond}(x_1x_1x_2x_2x_2x_1x_2x_2) = x_1x_2x_1x_2$. A word α is *condensed* if $\alpha = \mathsf{cond}(\alpha)$.

▶ Remark 1. For a word α, we have $\mathsf{loc}(\mathsf{cond}(\alpha)) = \mathsf{loc}(\alpha)$ [14]. Hence, by computing $\mathsf{cond}(\alpha)$ in time $\mathrm{O}(|\alpha|)$, algorithms for computing the locality number (and the respective marking sequences) for *condensed* words extend to algorithms for general words.

Examples and Word Combinatorial Considerations. The structure of 1-local and 2-local words is characterised in [14]. The simplest 1-local words are repetitions x^k for some $k \geq 0$. Furthermore, if α is 1-local, then $y^\ell \alpha y^r$ is 1-local, where $y \notin \mathsf{alph}(\alpha), \ell, r \geq 0$. Marking sequences for 1-local words can be obtained by going from the "inner-most" letters to the "outer-most" ones. The English words *radar*, *refer*, *blender*, and *rotator* are all 1-local.

Generally, in order to have a high locality number, a word needs to contain many alternating occurrences of (at least) two letters. For instance, $(x_1x_2)^n$ is n-local. In general, one can show that if $\mathsf{loc}(w) = k$, then $\mathsf{loc}(w^i) \in \{ik - i + 1, ik\}$.

The well-known *Zimin words* [36] also have high locality numbers compared to their lengths. These words are important in the domain of avoidability, as it was shown that a terminal-free pattern is unavoidable (i.e., it occurs in every infinite word over a large enough finite alphabet) if and only if it occurs in a Zimin word. The Zimin words Z_i, for $i \in \mathbb{N}$, are inductively defined by $Z_1 = x_1$ and $Z_{i+1} = Z_i x_{i+1} Z_i$. Clearly, $|Z_i| = 2^i - 1$ for all $i \in \mathbb{N}$. Regarding the locality of Z_i, note that marking x_2 leads to 2^{i-2} marked blocks; further, marking x_1 first and then the remaining symbols in an arbitrary order only extends or joins marked blocks. Thus, we obtain a sequence with marking number 2^{i-2}. In fact, it can be shown that $\mathsf{loc}(Z_i) = \frac{|Z_i|+1}{4} = 2^{i-2}$ for $i \in \mathbb{N}_{\geq 2}$. Notice that both Zimin words and 1-local words have an obvious palindromic structure. However, in the Zimin words, the letters occur multiple times, but not in large blocks, while in 1-local words there are at most 2 blocks of each letter. One can show that if w is a palindrome, with $w = uau^R$ or $w = uu^R$, and $\mathsf{loc}(u) = k$, then $\mathsf{loc}(w) \in \{2k-1, 2k, 2k+1\}$ (u^R denotes the reversal of u).

The number of occurrences of a letter alone is not always a good indicator of the locality of a word. The German word *Einzelelement* (a basic component of a construction) has 5 occurrences of e, but is only 3-local, as witnessed by marking sequence (*l,m,e,i,n,z,t*). Nevertheless, a repetitive structure often leads to high locality. The Finnish word *tu-tustuttu* (perfect passive of *tutustua* – to meet) is nearly a repetition and 4-local, while *pneumonoultramicroscopicsilicovolcanoconiosis* is an (English) 8-local word, and *lentokone-suihkuturbiinimoottoriapumekaanikkoaliupseerioppilas* is a 10-local (Finnish) word.

Complexity and Approximation. We briefly summarise the fundamentals of parameterised complexity [25, 18] and approximation [5].

A *parameterised problem* is a decision problem with instances (x, k), where x is the actual input and $k \in \mathbb{N}$ is the *parameter*. A parameterised problem P is *fixed-parameter tractable* if there is an *fpt-algorithm* for it, i.e., one that solves P on input (x, k) in time $f(k) \cdot p(|x|)$ for a recursive function f and a polynomial p. We use the $\mathrm{O}^*(\cdot)$ notation which hides multiplicative factors polynomial in $|x|$.

A minimisation problem P is a triple (I, S, m), where I is the *set of instances*, S is a function that maps instances $x \in I$ to the *set of feasible solutions* for x, and m is the *objective function* that maps pairs (x, y) with $x \in I$ and $y \in S(x)$ to a positive rational number. For every $x \in I$, we denote $m^*(x) = \min\{m(x, y) \colon y \in S(x)\}$. For $x \in I$ and $y \in S(x)$, the value $R(x, y) = \frac{m(x,y)}{m^*(x)}$ is the *performance ratio* of y with respect to x. An algorithm $\mathcal{A}$ is an *approximation algorithm* for P with ratio $r : \mathbb{N} \to \mathbb{Q}$ (or an r-approximation algorithm, for short) if, for every $x \in I$, $\mathcal{A}(x) = y \in S(x)$, and $R(x, y) \leq r(|x|)$. We also let r be of the form $\mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ when the ratio r depends on $m^*(x)$ and $|x|$; in this case, we write $r(\mathsf{opt}, |x|)$. We further assume that the function r is monotonically non-decreasing. Unless stated otherwise, all approximation algorithms run in polynomial time with respect to $|x|$.

Pathwidth, Cutwidth and Problem Definitions. Let $G = (V, E)$ be a (multi)graph with the vertices $V = \{v_1, \ldots, v_n\}$. A *cut* of G is a partition (V_1, V_2) of V into two disjoint subsets V_1, V_2, $V_1 \cup V_2 = V$; the (multi)set of edges $\mathcal{C}(V_1, V_2) = \{\{x, y\} \in E \mid x \in V_1, y \in V_2\}$ is called the cut-set or the (multi)set of edges crossing the cut, while V_1 and V_2 are called the sides of the cut. The *size* of this cut is the number of crossing edges, i.e., $|\mathcal{C}(V_1, V_2)|$. A *linear arrangement* of the (multi)graph G is a sequence $(v_{j_1}, v_{j_2}, \ldots, v_{j_n})$, where $(j_1, j_2, \ldots, j_n)$ is a permutation of $(1, 2, \ldots, n)$. For a linear arrangement $L = (v_{j_1}, v_{j_2}, \ldots, v_{j_n})$, let $L(i) = \{v_{j_1}, v_{j_2}, \ldots, v_{j_i}\}$. For every i, $1 \leq i < n$, we consider the cut $(L(i), V \setminus L(i))$ of G, and denote the cut-set $\mathcal{C}_L(i) = \mathcal{C}(L(i), V \setminus L(i))$ (for technical reasons, we also set $\mathcal{C}_L(0) = \mathcal{C}_L(n) = \emptyset$). We define the *cutwidth* of L by $\mathsf{cw}(L) = \max\{|\mathcal{C}_L(i)| \mid 0 \leq i \leq n\}$. Finally, the cutwidth of G is the minimum over all cutwidths of linear arrangements of G, i.e., $\mathsf{cw}(G) = \min\{\mathsf{cw}(L) \mid L \text{ is a linear arrangement for } G\}$.

A path decomposition (see [11]) of a connected graph $G = (V, E)$ is a tree decomposition whose underlying tree is a path, i.e., a sequence $Q = (B_0, B_1, \ldots, B_m)$ (of *bags*) with $B_i \subseteq V$, $0 \leq i \leq m$, satisfying the following two properties:

- *Cover property*: for every $\{u, v\} \in E$, there is an index i, $0 \leq i \leq m$, with $\{u, v\} \subseteq B_i$.
- *Connectivity property*: for every $v \in V$, there exist indices i_v and j_v, $0 \leq i_v \leq j_v \leq m$, such that $\{j \mid v \in B_j\} = \{i \mid i_v \leq i \leq j_v\}$. In other words, the bags that contain v occur on consecutive positions in $(B_0, \ldots, B_m)$.

The *width* of a path decomposition Q is $\mathsf{w}(Q) = \max\{|B_i| \mid 0 \leq i \leq m\} - 1$, and the *pathwidth* of a graph G is $\mathsf{pw}(G) = \min\{\mathsf{w}(Q) \mid Q \text{ is a path decomposition of } G\}$. A path decomposition is *nice* if $B_0 = B_m = \emptyset$ and, for every i, $1 \leq i \leq m$, either $B_i = B_{i-1} \cup \{v\}$ or $B_i = B_{i-1} \setminus \{v\}$, for some $v \in V$.

It is convenient to treat a path decomposition Q as a scheme marking the vertices of the graph based on the order in which the bags occur in the bag sequence. More precisely, all vertices are initially marked as `open`. Then we process the bags one by one, as they occur in Q. When we process the first bag that contains a vertex v, then v becomes `active`. When we process the last bag that contains v, it becomes `closed`. The connectivity property enforces that vertices that are `closed` cannot be marked as `active` again, while the cover property enforces that adjacent vertices must be both `active` at some point. The width is the maximum number of vertices which are marked `active` at the same time minus one. If the path decomposition is nice, then whenever a bag is processed as described above, we change the marking of exactly one vertex.

We next formally define the computational problems of computing the parameters defined above. By Loc, Cutwidth and Pathwidth, we denote the problems to check for a given word α or graph G and integer $k \in \mathbb{N}$, whether $\mathsf{loc}(\alpha) \leq k$, $\mathsf{cw}(G) \leq k$, and $\mathsf{pw}(G) \leq k$, respectively. Note that since we can assume that $k \leq |\alpha|$ and $k \leq |G|$, whether k is given in binary or unary has no impact on the complexity. With the prefix Min, we refer to the minimisation variants. More precisely, MinLoc $= (I, S, m)$, where I is the set of words, $S(\alpha)$ is the set of all marking sequences for α and $m(\alpha, \sigma) = \pi_\sigma(\alpha)$ (note that $m^*(\alpha) = \mathsf{loc}(\alpha)$); MinCutwidth $= (I, S, m)$, where I are all multigraphs, $S(G)$ is the set of linear arrangements of G, and $m(G, L) = \mathsf{cw}(L)$ (note that $m^*(G) = \mathsf{cw}(G)$); finally, MinPathwidth $= (I, S, m)$, where I are all graphs, $S(G)$ is the set of path decompositions of G, and $m(G, Q) = \mathsf{w}(Q)$ (note that $m^*(G) = \mathsf{pw}(G)$).

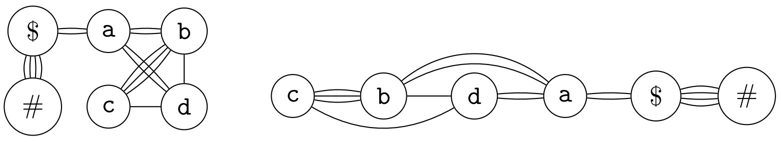

Figure 1 The graph $H_{\alpha,k}$ for $\alpha = \mathtt{abcbcdbada}$ and $k = 2$; an optimal linear arrangement of $H_{\alpha,k}$ with cutwidth 4 induces the optimal marking sequence $(\mathtt{c}, \mathtt{b}, \mathtt{d}, \mathtt{a})$ for α with marking number 2.

3 Locality and Cutwidth

In this section, we introduce polynomial-time reductions from Loc to Cutwidth and vice versa. The established close relationship between these two problems lets us derive several complexity-theoretic and algorithmic results for Loc. We also discuss approximation-preserving properties of our reductions.

First, we show a reduction from Loc to Cutwidth. For a word α and an integer $k \in \mathbb{N}$, we build a multigraph $H_{\alpha,k} = (V, E)$ whose set of nodes $V = \mathsf{alph}(\alpha) \cup \{\$, \#\}$ consists of symbols occurring in α and two additional characters $\$, \# \notin \mathsf{alph}(\alpha)$. The multiset of edges E contains an edge between nodes $x, y \in \mathsf{alph}(\alpha)$ for each occurrence of the factors xy and yx in α, as well as $2k$ edges between \$ and #, one edge between \$ and the first letter of α, and one edge between \$ and the last letter of α. An example is given in Figure 1.

▶ **Lemma 2.** *The graph $H_{\alpha,k}$ satisfies $\mathsf{cw}(H_{\alpha,k}) = 2k$ if and only if $\mathsf{loc}(\alpha) \leq k$.*

Proof. Suppose firstly that α is k-local, and let $\sigma = (x_1, x_2, \ldots, x_n)$ be an optimal marking sequence of α. Consider the linear arrangement $L = (x_1, x_2, \ldots, x_n, \$, \#)$. Clearly, $|\mathcal{C}(\{x_1, x_2, \ldots, x_n, \$\}, \{\#\})| = 2k$ and $|\mathcal{C}(\{x_1, x_2, \ldots, x_n\}, \{\$, \#\})| = 2$. Now consider a cut $(K_1, K_2) = (\{x_1, x_2, \ldots, x_i\}, \{x_{i+1}, \ldots, x_n, \$, \#\})$ for $1 \leq i < n$. Every edge $e \in \mathcal{C}(K_1, K_2)$ is of the form $\{x_j, x_h\}$ with $j \leq i < h$, or of the form $\{\alpha[1], \$\}$ or $\{\$, \alpha[|\alpha|]\}$. Consequently, every edge $e \in \mathcal{C}(K_1, K_2)$ corresponds to a unique factor $x_j x_h$ or $x_h x_j$ of α with $j \leq i < h$ and, after exactly the symbols $x_1, x_2, \ldots, x_i$ are marked, x_j is marked and x_h is not, or to a unique factor $\alpha[1]$ or $\alpha[|\alpha|]$ and, after exactly the symbols $x_1, x_2, \ldots, x_i$ are marked, $\alpha[1]$ or $\alpha[|\alpha|]$ is marked. Since there can be at most k marked blocks in α after marking the symbols $x_1, \ldots, x_i$, there are at most $2k$ such factors, which means that $|\mathcal{C}(K_1, K_2)| \leq 2k$. Thus $\mathsf{cw}(H_{\alpha,k}) \leq 2k$. Note that any linear arrangement must at some point separate the nodes \$ and #, meaning $\mathsf{cw}(H_{\alpha,k}) \geq 2k$, so we get that $\mathsf{cw}(H_{\alpha,k}) = 2k$.

Now suppose that the cutwidth of $H_{\alpha,k}$ is $2k$ and let L be an optimal linear arrangement witnessing this fact. Firstly, we note that L must either start with # followed by \$ (i.e., have the form $(\#, \$, \ldots)$) or end with # preceded by \$ (i.e., have the form $(\ldots, \$, \#)$. Otherwise, since $H_{\alpha,k}$ is connected, every cut separating \$ and # would be of size strictly greater than $2k$. Because a linear ordering and its mirror image have the same cutwidth, we may assume that the optimal linear arrangement has the form $L = (x_{\tau(1)}, x_{\tau(2)}, \ldots, x_{\tau(n)}, \$, \#)$ for some permutation τ of $\{1, \ldots, n\}$. Let σ be the marking sequence $(x_{\tau(1)}, x_{\tau(2)}, \ldots, x_{\tau(n)})$ of α induced by τ. Suppose, for contradiction, that for some i, with $1 \leq i < n$, after marking $x_{\tau(1)}, \ldots, x_{\tau(i)}$, we have $k' > k$ marked blocks. Furthermore, let $K_1 = \{x_{\tau(1)}, \ldots, x_{\tau(i)}\}$ and $K_2 = \{x_{\tau(i+1)}, \ldots, x_{\tau(n)}, \$, \#\}$. For every marked block $\alpha[s..t]$ that is not a prefix or a suffix of α, we have $\alpha[s], \alpha[t] \in K_1$ and $\alpha[s-1], \alpha[t+1] \in K_2$ and therefore $\{\alpha[s-1], \alpha[s]\}, \{\alpha[t], \alpha[t+1]\} \in \mathcal{C}(K_1, K_2)$. Moreover, for a marked prefix $\alpha[1..s]$, we have $\alpha[1], \alpha[s] \in K_1$ and $\$, \alpha[s+1] \in K_2$ and therefore $\{\alpha[1], \$\}, \{\alpha[s], \alpha[s+1]\} \in \mathcal{C}(K_1, K_2)$. Analogously, the existence of a marked suffix $\alpha[t..|\alpha|]$

leads to $\{\alpha[|\alpha|], \$\}, \{\alpha[t-1], \alpha[t]\} \in \mathcal{C}(K_1, K_2)$. Consequently, for each marked block, we have two unique edges in $\mathcal{C}(K_1, K_2)$, which implies $|\mathcal{C}(K_1, K_2)| \geq 2k' > 2k$. This contradicts the assumption that L is a witness that $H_{\alpha,k}$ has cutwidth $2k$. Thus, α must be k-local. ◀

In the following, we briefly discuss the complexity of this reduction. Suppose we are given a word α and an integer $k \leq |\alpha|$. It is usual in string algorithmics to assume that α is over an integer alphabet, i.e., $\mathsf{alph}(\alpha) \subseteq \{1, \ldots, |\alpha|\}$. In this framework, the multigraph $H_{\alpha,k}$ can be constructed in $\mathrm{O}(|\alpha|)$ time (e.g., represented as a list of vertices and a list of edges).

▶ **Lemma 3.** *If there is an $r(\mathsf{opt}, h)$-approximation algorithm for* MinCutwidth *running in $\mathrm{O}(f(h))$ time for an input multigraph with h edges, then there is an $(r(2\,\mathsf{opt}, |\alpha|) + \frac{1}{\mathsf{opt}})$-approximation algorithm for* MinLoc *running in $\mathrm{O}(f(|\alpha|) + |\alpha|)$ time on an input word α.*

Proof. As already indicated in the proof of Lemma 2, for $k = \mathsf{loc}(\alpha)$, every linear arrangement for $H_{\alpha,k}$ naturally translates to a marking sequence for α. However, in an approximate linear arrangement, the vertices # and \$ do not have to be at the first (or last) position. Still, the marking sequence corresponding to the linear arrangement L can have not more than $\frac{\mathsf{cw}(L)}{2} + 1$ blocks, since only suffix and prefix can be marked blocks which correspond to only one instead of two edges in a cut in $H_{\alpha,k}$. This observation remains valid if we do not include the extra vertices # and \$ in $H_{\alpha,k}$ in the reduction. Let H_α be the graph obtained from $H_{\alpha,k}$ (for some k) by removing the extra vertices # and \$ (observe that this also removes the dependence on k). Removing vertices only decreases the cutwidth, so Lemma 2 implies that $\mathsf{cw}(H_\alpha) \leq 2m^*(\alpha)$. Let α be an instance of MinLoc and $\mathcal{A}$ an $r(\mathsf{opt}, h)$-approximation for MinCutwidth on multigraphs. The approximation algorithm $\mathcal{A}$ run on H_α returns a linear arrangement $L = \mathcal{A}(H_\alpha)$ with $\mathsf{cw}(L) \leq r(\mathsf{opt}, h)\,\mathsf{cw}(H_\alpha)$. Let σ be the marking sequence corresponding to L, then $R(\alpha, \sigma) = \frac{\pi_\sigma(\alpha)}{m^*(\alpha)} \leq \frac{2}{\mathsf{cw}(H_\alpha)}(\frac{\mathsf{cw}(L)}{2} + 1) = \frac{\mathsf{cw}(L)}{\mathsf{cw}(H_\alpha)} + \frac{1}{m^*(\alpha)} = R(H_\alpha, L) + \frac{1}{m^*(\alpha)}$. The performance ratio $R(H_\alpha, L)$ is at most $r(\mathsf{opt}, h)$, where $h = |\alpha|$ is the number of edges in H_α. For the optimum value $k = m^*(\alpha)$, the cutwidth of $H_{\alpha,k}$ is at least $2k - 2$ and σ has performance ratio at most $r(2\,\mathsf{opt}, |\alpha|)$ (with respect to the optimum value k for MinLoc). The approximation procedure builds the graph H_α in $\mathrm{O}(|\Sigma|)$, runs $\mathcal{A}$ on H_α in $\mathrm{O}(f(|\alpha|))$ and translates the linear arrangement into a marking sequence σ in $\mathrm{O}(|\Sigma|)$. This gives an $(r(2\,\mathsf{opt}, |\alpha|) + \frac{1}{\mathsf{opt}})$-approximation for MinLoc running time in $\mathrm{O}(f(|\alpha|) + |\alpha|)$ time. ◀

For a reduction from Cutwidth to Loc, let $H = (V, E)$ be a connected multigraph, where V is the set of nodes and E the multiset of edges (for technical reasons, we assume $|V| \geq 2$). Let $H' = (V, E')$ be the multigraph obtained by duplicating every edge in H. As such, each node in H' has even degree, so there exists an Eulerian cycle C (i.e., a cycle visiting each edge exactly once) in H', and, moreover, $\mathsf{cw}(H') = 2\,\mathsf{cw}(H)$. For each edge $e \in E'$, let α_e be the word over V that corresponds to an arbitrary traversal of the Eulerian path P obtained from C by deleting e; see Figure 2 for an example.

▶ **Lemma 4.** *For any edge e in E', the word α_e satisfies $\mathsf{cw}(H) \leq \mathsf{loc}(\alpha_e) \leq \mathsf{cw}(H) + 1$. Moreover, there is a vertex $v \in V$ such that $\mathsf{loc}(\alpha_e) = \mathsf{cw}(H)$ for every edge e incident to v.*

Consequences. In the following, we overview a series of complexity-theoretic and algorithmic consequences of the reductions provided above. We first discuss negative results and note that we can close one of the main problems left open in [14].

▶ **Theorem 5.** *The* Loc *problem is NP-complete.*

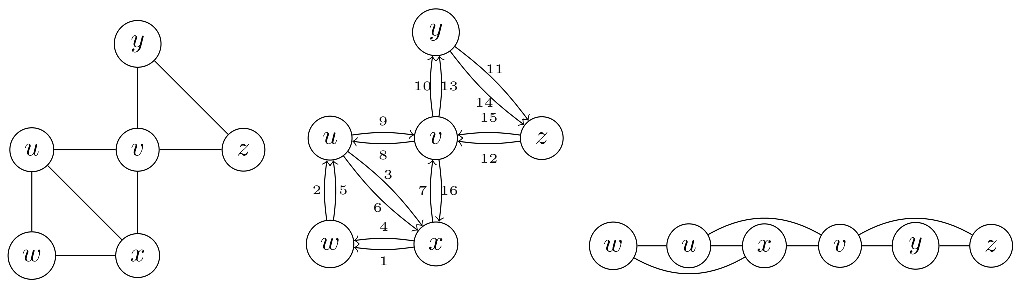

Figure 2 A graph H and its multigraph H' obtained by doubling the edges; the edge labels describe a Eulerian cycle that starts and ends in x. Deleting the edge (v, x) in this cycle yields the word $\alpha_{(v,x)} = xwuxwuxvuvyzvyzv$, which has an optimal marking sequence (w, u, x, v, y, z) with marking number 3, and, thus, induces an optimal linear arrangement of H with cutwidth 3.

Theorem 5 follows from the Turing reduction from Cutwidth to Loc, but it can also be proved using a polynomial-time one-to-many reduction from the well known NP-complete problem Clique. This alternative approach is more technically involved but has the merit of emphasising how the combinatorial properties of the locality number can be used to construct computationally hard instances of Loc. Moreover, by the word-combinatorial observations about locality made in Section 2, it is clear that Loc is NP-complete also for words with special structure, e.g., palindromes and repetitions.

With respect to approximation, it is known that, assuming the Small Set Expansion Conjecture (denoted SSE; see [40]), there exists no constant-ratio approximation for MinCutwidth (see [48]). Consequently, approximating MinLoc within any constant factor is also SSE-hard. In particular, we point out that stronger inapproximability results for MinCutwidth are not known. Positive approximation results for MinLoc will be discussed in Section 4.

On certain graph classes, the SSE conjecture is equivalent to the Unique Games Conjecture [32] (see [40, 41]), which, at its turn, was used to show that many approximation algorithms are tight [33] and is considered a major conjecture in inapproximability. However, some works seem to provide evidence that could lead to a refutation of SSE; see [3, 6, 29]. In this context, we show in Section 4 a series of unconditional results which state that multiple natural greedy strategies do not provide low-ratio approximations of MinLoc.

As formally stated next, Lemma 2 extends algorithmic results for computing cutwidth to determining the locality number (we formulate this result so that it also covers fpt-algorithms with respect to the standard parameters $\mathsf{cw}(G)$ and $\mathsf{loc}(\alpha)$). Note that the maximum degree in a multigraph G is bounded from above by $2\,\mathsf{cw}(G)$, so the number of nodes n and the number of edges h satisfy $h \leq n \cdot \mathsf{cw}(G)$. Hence, we state the complexity in terms of n and $\mathsf{cw}(G)$ rather than with respect to h, which is the actual input size.

▶ **Lemma 6.** *If* MinCutwidth *(resp.* Cutwidth*) can be solved in* $O(f(\mathsf{cw}(G), n))$ *time for a multigraph* G *with* n *vertices, then the* MinLoc *(resp.,* Loc*) problem can be solved in* $O(f(2\,\mathsf{loc}(\alpha), |\Sigma| + 2) + |\alpha|)$ *time for a word* α *over an alphabet* Σ.

In particular, we can draw the following corollaries using Lemma 6 and known results from the literature. Due to the algorithms of [12], which also work for multigraphs[1], MinLoc can be solved in $O^*(2^{|\Sigma|})$ time and space, or in $O^*(4^{|\Sigma|})$ time and polynomial space. In

[1] These algorithms actually support weighted graphs without any major modification and in the same complexity. In this setting, parallel edges connecting two vertices are replaced by a single “super-edge” whose weight is the number of parallel edges.

particular, this also implies that Loc is fixed-parameter tractable with respect to the alphabet size. Moreover, the fpt-algorithm from [47] directly implies that MinLoc is fixed-parameter tractable for parameter $\mathsf{loc}(\alpha)$ with linear fpt-running-time $g(\mathsf{loc}(\alpha))\,\mathrm{O}(n)$. Since Cutwidth is NP-complete already for graphs with maximum degree 3 (see [37]), we also derive a stronger statement compared to Theorem 5: Loc is NP-complete even if every symbol has at most 3 occurrences; if every symbol has at most 2 occurrences, the complexity of Loc is open, while the case where every symbol has only one occurrence is trivial. If, on the other hand, the symbols have many occurrences in comparison to $|\alpha|$, i.e., $|\Sigma| = \mathrm{O}(\log(|\alpha|))$, then Loc can be solved in polynomial time, e.g., using the $\mathrm{O}^*(2^{|\Sigma|})$-time algorithm mentioned above.

4 Locality and Pathwidth

In this section, we consider the approximability of the minimisation problem MinLoc. Since a marking sequence is just a linear arrangement of the symbols of the input word, this problem seems to be well tailored to greedy algorithms: until all symbols are marked, we choose an unmarked symbol according to some greedy strategy and mark it. There are two aspects that motivate the investigation of such approaches. Firstly, ruling out simple strategies is a natural initial step in the search for approximation algorithms for a new problem. Secondly, due to the results of Section 3, the obvious greedy approaches for computing the locality number may also provide a new angle to approximating the cutwidth of a graph, i.e., some greedy strategies may only become apparent in the locality number point of view and hard to see in the graph formulation of the problem. Given the fact that, as formally stated later as Theorem 10, approximating the cutwidth via approximation of the locality number does, in fact, improve the best currently known cutwidth approximation ratio, this seems to be a rather important aspect.

Unfortunately, we can formally show that many natural candidates for greedy strategies fail to yield promising approximation algorithms (and are therefore also not helpful for cutwidth approximation). We just briefly mention these negative results. The four considered basic strategies are the following: (1) prefer symbols with few occurrences, (2) symbols with many occurrences, (3) symbols leading to fewer blocks after marking, (4) symbols with earlier leftmost occurrence. All these strategies fail in a sense that there are arbitrarily long (condensed) words α with constant locality numbers for which these strategies yield marking sequences with marking numbers $\Omega(|\alpha|)$.

A more promising approach is to choose among symbols that extend at least one already marked block (except when marking the first symbol). We denote this strategy by BlockExt and marking sequences that can be obtained by it are called BlockExt-marking sequences. Intuitively, marking a symbol that has only isolated occurrences, and therefore will increase the current number of marked blocks by the number of its occurrences, seems a bad choice. This raises a general question of whether every word has a BlockExt-marking sequence that is also optimal for this word. We answer this question negatively: all BlockExt-marking sequences for words like $x_1yx_2yx_3y\dots x_{2k}y$ achieve a marking number of at least $2k-1$, while first marking $x_2, x_3, \dots, x_{k+1}$ in this order (which all have only isolated occurrences), then y, and then the rest of the symbols in some order, yields at most k marked blocks. However, this only shows a lower bound of roughly 2 for the approximation ratio of algorithms based on BlockExt, so BlockExt might still be a promising candidate. However, in order to devise a BlockExt-based approximation algorithm, we still face the problem of deciding which of the extending symbols should be chosen; trying out all of them is obviously too costly. Unfortunately, if we handle this decision by one of the basic strategies (1)–(4) from above,

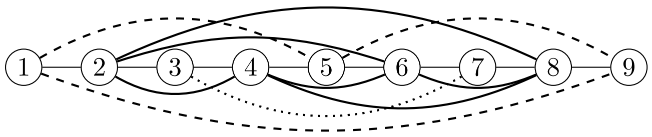

Figure 3 The graph G_α for $\alpha = \texttt{cabacabac}$; the three cliques are drawn with different edge-types.

e.g., choosing among all extending symbols one that leads to fewer new blocks, we again end up with poor approximation ratios. More precisely, we can again find arbitrarily long words α with constant locality numbers for which these algorithms yield marking numbers $\Omega(|\alpha|)$. Moreover, this is also true if we choose among all extending symbols one that has a maximum number of extending occurrences or one that maximises the ratio $\frac{\text{\#extending occ.}}{\text{\#occ.}}$.

While we obviously have not investigated *all* reasonable greedy strategies, we consider our negative results as sufficient evidence that a worthwhile approximation algorithm for computing the locality number most likely does not follow from such simple greedy strategies.

In the following, we adopt a more sophisticated approach of approximating the locality number: we devise a reduction to the problem of computing the pathwidth of a graph. To this end, we first have to describe how a (condensed) word can be represented as a graph: For a condensed word α, the graph $G_\alpha = (V_\alpha, E_\alpha)$ is defined by $V_\alpha = \{1, 2, \ldots, |\alpha|\}$ and $E_\alpha = \{\{i, i{+}1\} \mid 1 \leq i \leq |\alpha|{-}1\} \cup \{\{i, j\} \mid \{i, j\} \subseteq \mathsf{ps}_x(\alpha) \text{ for some } x \in \mathsf{alph}(\alpha)\}$. Intuitively, G_α is obtained by interpreting every position of α as a vertex, connecting neighbouring positions by edges, and turning every set $\mathsf{ps}_x(\alpha)$, $x \in \mathsf{alph}(\alpha)$, into a clique (see Figure 3).

We use G_α as a unique graph representation for condensed words and whenever we talk about a path decomposition for α, we actually refer to a path decomposition of G_α and, since G_α has the positions of α as its vertices, the marking scheme behind a path decomposition (and its respective terminology) directly translates to a marking scheme of the positions of α.

▶ **Lemma 7.** *Let α be a condensed word with $|\alpha| \geq 2$. Then $\mathsf{loc}(\alpha) \leq \mathsf{pw}(G_\alpha) \leq 2\,\mathsf{loc}(\alpha)$.*

Proof Sketch. We only sketch how a marking sequence translates into a path decomposition and vice versa. Let $\sigma = (x_1, x_2, \ldots, x_m)$ be a marking sequence for a condensed word α with $\pi_\sigma(\alpha) = k$. We describe a path decomposition Q of G_α as a marking scheme. First, for every i, $1 \leq i \leq m$, let p_i be a step of Q (corresponding to one of the bags of Q) that represents stage i of σ: every position that is a border position of a marked block is `active`, every other marked position is `closed`, and all other positions are `open`. The path decomposition produces these steps in the order $p_1, p_2, \ldots, p_m$ and such a step p_i is reached from the predecessor step p_{i-1} by a sequence of intermediate steps as follows. Step p_1 is obtained from the initial one by setting all positions in $\mathsf{ps}_{x_1}(\alpha)$ to `active`, the final step of Q, where all positions are `closed`, is obtained from step p_m by setting the only `active` positions 1 and $|\alpha|$ to `closed`. In order to produce step p_{i+1} from step p_i, we do the following. For every $j \in \mathsf{ps}_{x_{i+1}}(\alpha)$ that does not create a new marked block of size 1, we set position j to `active` and immediately after that, we set all `active` neighbours of j to `closed` if they do not have `open` neighbours anymore. Next, we set all remaining positions from $\mathsf{ps}_{x_{i+1}}(\alpha)$ to `active` and, finally, we set all positions from $\mathsf{ps}_{x_{i+1}}(\alpha)$ that have no `open` neighbours to `closed`. It can be verified with a moderate effort that we have now obtained step p_{i+1} and also that Q is, in fact, a valid path decomposition of G_α. In order to see that $\mathsf{pw}(Q) \leq 2k$, we first note that the number of `active` positions in each step p_i is clearly bounded by $2k$. In going from p_i to p_{i+1}, we necessarily reach a step where all positions of $\mathsf{ps}_{x_{i+1}}(\alpha)$ are now `active`, while some of the previous border positions might also still be `active`. It requires a more involved and careful counting argument to see that the total number of `active` positions in these intermediate steps never exceeds $2k+1$.

For the other direction, let $Q = (B_0, B_1, B_2, \ldots, B_{2|\alpha|})$ be an arbitrary nice path decomposition of G_α. For every i, $1 \leq i \leq m$, let p_i be the first step of Q where all positions of $\mathsf{ps}_{x_i}(\alpha)$ are `active`. We order the characters x_i so that $p_1 < p_2 < \ldots < p_m$ and define a marking sequence for α by setting $\sigma = (x_1, x_2, \ldots, x_m)$. It is comparatively easy to show that at every step p_i of Q there is at least one `active` position per marked block of stage i of σ. However, this only shows that $\pi_\sigma(\alpha) - 1 \leq \mathsf{pw}(Q)$ and in order to prove that, in fact, $\pi_\sigma(\alpha) \leq \mathsf{pw}(Q)$ holds, we show that either there is a step of Q with at least $\pi_\sigma(\alpha) + 1$ `active` positions or, if this is not the case, then there must be a marking sequence σ' with $\pi_{\sigma'}(\alpha) = \pi_\sigma(\alpha) - 1$. This implies that, for every path decomposition Q of G_α, $\mathsf{loc}(\alpha) \leq \mathsf{pw}(Q)$ and therefore also $\mathsf{loc}(\alpha) \leq \mathsf{pw}(G_\alpha)$. Proving this claim requires a long and technically involved case analysis. ◀

Note that Lemma 7 is not true for condensed words α of size 1, since then $\mathsf{loc}(\alpha) = 1$ and $\mathsf{pw}(G_\alpha) = 0$. The reason why $\mathsf{pw}(G_\alpha)$ can range between $\mathsf{loc}(\alpha)$ and $2\,\mathsf{loc}(\alpha)$ (rather than $\mathsf{pw}(G_\alpha) = 2\,\mathsf{loc}(\alpha)$) is that in a marking sequence, every marked block accounts for one unit of the quantity $\mathsf{loc}(\alpha)$, while in the path decomposition, a marked block is represented either by two `active` vertices or by only one (if the block has size one). There are (condensed) examples that reach the extremes $\mathsf{loc}(\alpha)$ and $2\,\mathsf{loc}(\alpha)$, i.e., the bounds of Lemma 7 are tight.

▶ **Proposition 8.** *Let $\alpha = (x_1 x_2 \ldots x_n x_{n-1} \ldots x_2)^k x_1$ with $n \geq 3$, and let $\beta = (x_1 x_2)^k$. Then we have $\mathsf{loc}(\alpha) = k$ and $\mathsf{pw}(G_\alpha) = 2k$, and $\mathsf{loc}(\beta) = \mathsf{pw}(G_\beta) = k$.*

Note that the construction of a graph G_α from a word α does not technically provide a reduction from the decision problem Loc to Pathwidth (due to the fact that $\mathsf{pw}(G_\alpha)$ lies between $\mathsf{loc}(\alpha)$ and $2\,\mathsf{loc}(\alpha)$) and therefore cannot be used to solve MinLoc exactly. Its main purpose is to carry over approximation results from MinPathwidth to MinLoc, which is formally stated by the next lemma (in this regard, note that exact algorithms for MinLoc are obtained in Section 3 via a reduction to MinCutwidth instead).

▶ **Lemma 9.** *If* MinPathwidth *admits an $\mathrm{O}(f(n))$-time $r(\mathsf{opt}, n)$-approximation algorithm, then* MinLoc *admits an $\mathrm{O}(f(|\alpha|) + |\alpha|^2)$-time $2r(2\,\mathsf{opt}, |\alpha|)$-approximation algorithm.*

Consequently, approximation algorithms for MinPathwidth carry over to MinLoc. To the knowledge of the authors, the currently best approximation algorithm for MinPathwidth is due to [20], with an approximation ratio of $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$. This implies the following.

▶ **Theorem 10.** *There is an $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for* MinLoc.

Another consequence that is worth mentioning is due to the fact that an optimal path decomposition can be computed faster than $\mathrm{O}^*(2^n)$. More precisely, it is shown in [46] that for computing path decompositions, there is an exact algorithm with running time $\mathrm{O}^*((1.9657)^n)$, and even an additive approximation algorithm with running time $\mathrm{O}^*((1.89)^n)$. Consequently, there is a 2-approximation algorithm for MinLoc with running time $\mathrm{O}^*((1.9657)^n)$ and an asymptotic 2-approximation algorithm with running time $\mathrm{O}^*((1.89)^n)$ for MinLoc.

By combining the reduction from MinCutwidth to MinLoc from Section 3 with the reduction from MinLoc to MinPathwidth defined above, we obtain a reduction from MinCutwidth to MinPathwidth that carries over the pathwidth-approximation from [20] to MinCutwidth as follows (in particular, this improves the state-of-the-art approximation algorithm for MinCutwidth from [35]).

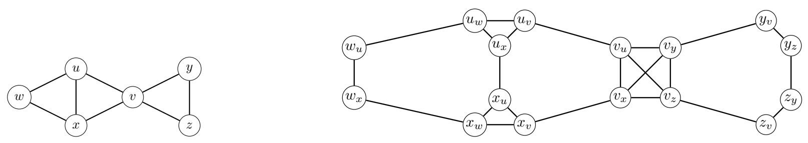

Figure 4 A graph G and the corresponding graph G' obtained by the reduction.

▶ **Theorem 11.** *There is a* $\mathrm{O}(\sqrt{\log(\mathit{opt})}\log(n))$*-approximation for* MinCutwidth.

Note that Theorem 11 only applies to simple graphs; see Section 5 for the case of multigraphs.

Many existing algorithms constructing path decompositions are of theoretical interest only, and this disadvantage carries over to the possible algorithms computing the locality number or cutwidth based on them. However, the reduction of Lemma 7 is also applicable in a purely practical scenario, since any kind of practical algorithm constructing path decompositions can be used in order to compute marking sequences (the additional tasks of building G_α and the translation of a path decomposition for it back to a marking sequence are computationally simple). This observation is particularly interesting since developing practical algorithms constructing tree and path decompositions of small width is a vibrant research area.[2]

5 Pathwidth and Cutwidth

Since pathwidth and cutwidth are classical graph parameters that play an important role for graph algorithms, independent from our application for computing the locality number, we also present a direct reduction from MinCutwidth to MinPathwidth.

For a graph $G = (V, E)$, we construct the graph $G' = (V', E')$ with $V' = \{v_u \mid \{u, v\} \in E\}$ and $E' = \{\{u_v, v_u\} \mid \{u, v\} \in E\} \cup \{\{v_u, v_w\} \mid \{u, v\}, \{w, v\} \in E, u \neq w\}$; see Figure 4.

▶ **Lemma 12.** *Let G be a graph with at least one edge. Then* $\mathsf{cw}(G) \leq \mathsf{pw}(G') \leq 2\,\mathsf{cw}(G)$.

Lemma 12 does not only prove that $\mathsf{cw}(G) \leq \mathsf{pw}(G') \leq 2\,\mathsf{cw}(G)$, but also yields a constructive way to compute a linear arrangement for G of cut at most k from a path decomposition of width k for G'. Further, Lemma 12 remains true if G is a multigraph; observe that the reduction still constructs a simple graph G'. This gives the following result.

▶ **Lemma 13.** *If there is an* $r(\mathsf{opt}, |V|)$*-approximation algorithm for* MinPathwidth *with running-time* $\mathrm{O}(f(|V|))$*, then there is also an* $2r(2\,\mathsf{opt}, h)$*-approximation algorithm for* MinCutwidth *on multigraphs with running time* $\mathrm{O}(f(h) + h^2 + n)$*, where n is the number of vertices and h is the number of edges.*

With the $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation for MinPathwidth from [20], Lemma 13 gives the following approximation for MinCutwidth on multigraphs.

▶ **Theorem 14.** *There is a (polynomial-time)* $\mathrm{O}(\sqrt{\log(\mathit{opt})}\log(h))$*-approximation algorithm for* MinCutwidth *on multigraphs with h edges.*

In accordance with Theorem 11, Theorem 14 yields an $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for simple graphs. Analogously, Theorem 11 could be formulated for multigraphs, which would also change the approximation-ratio to $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(h))$.

[2] See, e.g., the work [13] and the references therein for practical algorithms constructing path decompositions; also note that designing exact and heuristic algorithms for constructing tree decompositions was part of the "PACE 2017 Parameterized Algorithms and Computational Experiments Challenge" [16].

References

1 Amihood Amir and Igor Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5(3):514–523, 2007. doi:10.1016/j.jda.2006.10.001.

2 Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980. doi:10.1016/0022-0000(80)90041-0.

3 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential Algorithms for Unique Games and Related Problems. *Journal of the ACM*, 62(5):42:1–42:25, 2015. doi:10.1145/2775105.

4 Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):5:1–5:37, 2009. doi:10.1145/1502793.1502794.

5 Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi, and Viggo Kann. *Complexity and approximation: combinatorial optimization problems and their approximability properties*. Springer, 1999. doi:10.1007/978-3-642-58412-1.

6 Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding Semidefinite Programming Hierarchies via Global Correlation. In Rafail Ostrovsky, editor, *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011*, pages 472–481. IEEE, 2011. doi:10.1109/focs.2011.95.

7 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Transactions on Database Systems*, 37(4):1–46, 2012. doi:10.1145/2389241.2389250.

8 Hans L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11(1–2):1–21, 1993. URL: http://www.inf.u-szeged.hu/actacybernetica/edb/vol11n1_2/pdf/Bodlaender_1993_ActaCybernetica.pdf.

9 Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(5):1305–1317, 1996. doi:10.1137/s0097539793251219.

10 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

11 Hans L. Bodlaender. Fixed-Parameter Tractability of Treewidth and Pathwidth. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *LNCS*, pages 196–227, 2012. doi:10.1007/978-3-642-30891-8_12.

12 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. A Note on Exact Algorithms for Vertex Ordering Problems on Graphs. *Theory of Computing Systems*, 50(3):420–432, 2012. doi:10.1007/s00224-011-9312-0.

13 David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental Evaluation of a Branch-and-Bound Algorithm for Computing Pathwidth and Directed Pathwidth. *ACM Journal of Experimental Algorithmics*, 21(1):1.3:1–1.3:23, 2016. doi:10.1145/2851494.

14 Joel D. Day, Pamela Fleischmann, Florin Manea, and Dirk Nowotka. Local Patterns. In Satya V. Lokam and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017*, volume 93 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.24.

15 Joel D. Day, Pamela Fleischmann, Florin Manea, Dirk Nowotka, and Markus L. Schmid. On Matching Generalised Repetitive Patterns. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory, DLT 2018*, volume 11088 of *LNCS*, pages 269–281. Springer, 2018. doi:10.1007/978-3-319-98654-8_22.

16 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Parameterized and Exact Computation, IPEC 2017*, volume 89 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

17 Josep Díaz, Jordi Petit, and Maria Serna. A Survey of Graph Layout Problems. *ACM Computing Surveys*, 34(3):313–356, 2002. doi:10.1145/568522.568523.

18 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

19 Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Angelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261(1):119–156, 2001. `doi:10.1016/s0304-3975(00)00136-5`.

20 Uriel Feige, MohammadTaghi HajiAghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. `doi:10.1137/05064299x`.

21 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Pattern Matching with Variables: Fast Algorithms and New Hardness Results. In Ernst W. Mayr and Nicolas Ollinger, editors, *Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPIcs*, pages 302–315. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.STACS.2015.302`.

22 Henning Fernau, Florin Manea, Robert Mercaş, and Markus L. Schmid. Revisiting Shinohara's Algorithm for Computing Descriptive Patterns. *Theoretical Computer Science*, 733:44–54, 2018. `doi:10.1016/j.tcs.2018.04.035`.

23 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015. `doi:10.1016/j.ic.2015.03.006`.

24 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the Parameterised Complexity of String Morphism Problems. *Theory of Computing Systems*, 59(1):24–51, 2016. `doi:10.1007/s00224-015-9635-3`.

25 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. `doi:10.1007/3-540-29953-X`.

26 Dominik D. Freydenberger. Extended Regular Expressions: Succinctness and Decidability. *Theory of Computing Systems*, 53(2):159–193, 2013. `doi:10.1007/s00224-012-9389-0`.

27 Dominik D. Freydenberger and Markus L. Schmid. Deterministic regular expressions with back-references. *Journal of Computer and System Sciences*, 2019. `doi:10.1016/j.jcss.2019.04.001`.

28 Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O'Reilly, Sebastopol, CA, 3rd edition, 2006.

29 Venkatesan Guruswami and Ali Kemal Sinop. Lasserre Hierarchy, Higher Eigenvalues, and Approximation Schemes for Graph Partitioning and Quadratic Integer Programming with PSD Objectives. In Rafail Ostrovsky, editor, *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2011*, pages 482–491. IEEE, 2011. `doi:10.1109/FOCS.2011.36`.

30 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47(3):483–505, 2000. `doi:10.1145/337244.337255`.

31 Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In Ronald L. Rivest, David Haussler, and Manfred K. Warmuth, editors, *Computational Learning Theory, COLT 1989*, pages 57–71. Morgan Kaufmann, 1989. `doi:10.1016/b978-0-08-094829-4.50007-6`.

32 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *34th Annual ACM Symposium on Theory of Computing, STOC 2002*, pages 767–775. ACM, 2002. `doi:10.1145/509907.510017`.

33 Subhash Khot. On the Unique Games Conjecture (Invited Survey). In *Computational Complexity, CCC 2010*, pages 99–121. IEEE, 2010. `doi:10.1109/CCC.2010.19`.

34 Ton Kloks, editor. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. `doi:10.1007/BFb0045375`.

35 Tom Leighton and Satish Rao. Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms. *Journal of the ACM*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

36 M. Lothaire, editor. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. doi:10.1017/cbo9781107326019.

37 Fillia Makedon, Christos H. Papadimitriou, and Ivan Hal Sudborough. Topological Bandwidth. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):418–444, 1985. doi:10.1137/0606044.

38 Yen Kaow Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397(1–3):150–165, 2008. doi:10.1016/j.tcs.2008.02.028.

39 Jordi Petit. Addenda to the Survey of Layout Problems. *Bulletin of the EATCS*, 105:177–201, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/98.

40 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In Leonard J. Schulman, editor, *42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 755–764. ACM, 2010. doi:10.1145/1806689.1806792.

41 Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between Expansion Problems. In *Computational Complexity, CCC 2012*, pages 64–73. IEEE, 2012. doi:10.1109/CCC.2012.43.

42 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397(1–3):166–193, 2008. doi:10.1016/j.tcs.2008.02.029.

43 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded Treewidth. *Information and Computation*, 239:87–99, 2014. doi:10.1016/j.ic.2014.08.010.

44 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016. doi:10.1016/j.ic.2016.02.003.

45 Takeshi Shinohara. Polynomial Time Inference of Pattern Languages and Its Application. In *7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

46 Karol Suchan and Yngve Villanger. Computing Pathwidth Faster Than 2^n. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, IWPEC 2009*, volume 5917 of *LNCS*, pages 324–335. Springer, 2009. doi:10.1007/978-3-642-11269-0_27.

47 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

48 Yu (Ledell) Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of Treewidth and Related Problems. *Journal of Artificial Intelligence Research*, 49(1):569–600, 2014. doi:10.1613/jair.4030.