Computing rank-width exactly

Sang-il Oum*†

Department of Mathematical Sciences

KAIST

335 Gwahangno Yuseong-gu Daejeon 305-701 South Korea

September 9, 2008

Abstract

We prove that the rank-width of an n-vertex graph can be computed exactly in time $O(2^n n^3 \log^2 n \log\log n)$. To improve over a trivial $O(3^n \log n)$-time algorithm, we develop a general framework for decompositions on which an optimal decomposition can be computed efficiently. This framework may be used for other width parameters, including the branch-width of matroids and the carving-width of graphs.

Keyword: graph algorithms; rank-width; branch-width; exact exponential algorithms; fast subset convolution

1 Introduction

We present an exponential-time algorithm to calculate the rank-width of a graph, whose running time is faster than the trivial exponential-time algorithm. In order to present the result in a general framework, we define a decomposition of a finite set, which is roughly a description how one can split a finite set recursively until no further split can be performed.

*Supported by the SRC Program of Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R11-2007-035-01002-0).

†Email: `sangil@kaist.edu`

To be precise, we define a *rooted binary tree* as a directed tree with a specified vertex r called the *root* such that the root has two incoming edges and no outgoing edges and every vertex other than the root has exactly one outgoing edges and either two or zero incoming edges. A *leaf* of a rooted binary tree is a vertex with no incoming edges. A *descendent* of an edge e of a rooted binary tree is the set of vertices from which there exists a directed path to e.

A *decomposition* of a finite set V is a pair (T, μ) of a rooted binary tree T and a bijection μ from V to the set of all leaves of V. For a given set function f on the subsets of a finite set V, we measure the *f-width* of a decomposition (T, μ) of V as the maximum $f(\mu^{-1}(X_e))$ over all edges e of T where X_e is the set of leaves that are descendents of e in T. The *width* of f on a finite set V, denoted by $\mathrm{w}(f, V)$, is the minimum f-width over all possible decompositions of V. If $|V| \leq 1$, then V admits no decompositions but we let $\mathrm{w}(f, V) = f(V)$.

From the definition, it is straightforward to observe the following lemma. We write 2^V to denote the set of all subsets of V. Let $\mathbb{Z}$ be the set of integers. Throughout the paper, we let $n = |V|$ and let $M = \max_{X \subseteq V} |f(X)|$.

Lemma 1. *Let V be a finite set and let $f : 2^V \to \mathbb{Z}$ be a function. For a subset X of V,*

$$\mathrm{w}(f, X) = \begin{cases} \min_{\emptyset \neq Y \subsetneq X} \max(f(Y), f(X \setminus Y), \mathrm{w}(f, Y), \mathrm{w}(f, X \setminus Y)) & \text{if } |X| \geq 2, \\ f(X) & \text{if } |X| \leq 1. \end{cases}$$

Lemma 1 allows a simple exponential-time algorithm based on dynamic programming to compute $\mathrm{w}(f, V)$. In this paper, our model of computation is the random access machine on which arithmetic operations on integers as well as comparisons can be performed in a unit time only for integers of constant size. We need at most $\lceil \log_2 M \rceil$ bits to store $f(X)$ and $\mathrm{w}(f, X)$ for each subset X of V and so comparing $f(Y)$, $f(X \setminus Y)$, $\mathrm{w}(f, Y)$ and $\mathrm{w}(f, X \setminus Y)$ takes time $O(\log M)$. Assume that evaluating $f(X)$ takes time $O(\alpha)$. Then the running time is $O(2^n \alpha + \sum_{k=2}^{n} \binom{n}{k} 2^k \log M) = O(3^n \log M + 2^n \alpha)$.

In this paper, we will present an algorithm to compute $\mathrm{w}(f, V)$ in time

$$O(2^n(n^3 \log n \log\log n \log M + \log^2 M + \alpha)).$$

To construct this algorithm, we use the fast subset convolution of Björklund et al. [1], which we will discuss in Section 2.

Our algorithm may be applied to various width parameters. We will discuss some of them in Section 4. Rank-width of graphs was defined by Oum and Seymour [5]. As a consequence of our algorithm, we obtain an exact algorithm to compute the rank-width of an n-vertex graph in time $O(2^n n^3 \log^2 n \log\log n)$. This is clearly better than the above $O(3^n \log n)$-time algorithm. We remark that it is NP-hard to compute the rank-width [4].

2 Fast Subset Convolution

We summarize the algorithm for the fast subset convolution developed in [1]. Let R be a ring. Let $f : 2^V \to R$ and $g : 2^V \to R$ be functions. The *convolution* of f and g is the function $f * g : 2^V \to R$ such that

$$(f * g)(S) = \sum_{T \subseteq S} f(T) g(S \setminus T).$$

Theorem 2 (Björklund, Husfeldt, Kaski, and Koivisto [1]). *Let V be a finite set with n elements. Let R be a fixed ring. Given functions $f : 2^V \to R$ and $g : 2^V \to R$, the convolution $f * g$ can be computed in $O(2^n n^2)$ ring operations.*

Computing $f * g$ means completing a big table containing all values of $(f * g)(S)$ for all subsets S of V. For the completeness of this paper, we include the whole algorithm for the convolution. For the correctness of the algorithm, please refer to their paper.

Let $V = \{1, 2, \ldots, n\}$.

(A) Compute $\hat{f}(k, X) = \sum_{\substack{S \subseteq X \\ |S| = k}} f(S)$ for each $k \in \{0, 1, \ldots, n\}$ and $X \subseteq V$ as follows:

$$\hat{f}_0(k, X) = \begin{cases} f(X) & \text{if } |X| = k, \\ 0 & \text{otherwise.} \end{cases}$$

For $j \in \{1, 2, \ldots, n\}$,

$$\hat{f}_j(k, X) = \begin{cases} \hat{f}_{j-1}(k, X) & \text{if } j \notin X, \\ \hat{f}_{j-1}(k, X) + \hat{f}_{j-1}(k, X \setminus \{j\}) & \text{if } j \in X. \end{cases}$$

$$\hat{f}(k, X) = \hat{f}_n(k, X).$$

Similarly, compute $\hat{g}(k, X) = \sum_{\substack{S \subseteq X \\ |S| = k}} g(S)$ for each $k \in \{0, 1, \ldots, n\}$ and $X \subseteq V$.

(B) Compute $(\hat{f} \circledast \hat{g})(k, X) = \sum_{j=0}^{k} \hat{f}(j, X)\hat{g}(k-j, X)$ for each $k \in \{0, 1, \ldots, n\}$ and $X \subseteq V$.

(C) Compute $(f * g)(k, X) = \sum_{S \subseteq X} (-1)^{|X \setminus S|} (\hat{f} \circledast \hat{g})(k, S)$ for each $k \in \{0, 1, \ldots, n\}$ and $X \subseteq V$ as follows:

$$(f * g)_0(k, X) = (\hat{f} \circledast \hat{g})(k, X).$$

For $j \in \{1, 2, \ldots, n\}$,

$$(f * g)_j(k, X) = \begin{cases} (f * g)_{j-1}(k, X) & \text{if } j \notin X, \\ (f * g)_{j-1}(k, X) - (f * g)_{j-1}(k, X \setminus \{j\}) & \text{if } j \in X. \end{cases}$$

$$(f * g)(k, X) = (f * g)_n(k, X).$$

(D) Compute $(f * g)(X) = (f * g)(|X|, X)$ for each subset X of V.

3 Computing the width of f

Lemma 3. *Let V be a finite set with n elements. Let k be some number. Suppose that we are given a big table containing the true/false information whether or not $f(X) \le k$ for each subset X of V. Then we can decide whether $\mathrm{w}(f, V) \le k$ in time $O(2^n(n^3 \log n \log\log n))$.*

Proof. Let us assume that $n = |V| \ge 2$, because otherwise the width of f on V is trivial. For all subsets X of V and $i \in \{1, 2, \ldots, n\}$, let

$$g_i(X) = \begin{cases} 1 & \text{if } 1 \le |X| \le i,\ X \ne V,\ \mathrm{w}(f, X) \le k, \text{ and } f(X) \le k, \\ 1 & \text{if } i = n,\ X = V, \text{ and } \mathrm{w}(f, X) \le k, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Then Lemma 1 implies the following:

$$\mathrm{w}(f, X) \le k \text{ if and only if } \begin{cases} (g_{|X|-1} * g_{|X|-1})(X) \ne 0 & \text{if } |X| \ge 2, \\ f(X) \le k & \text{if } |X| \le 1. \end{cases} \tag{2}$$

The equation (2) allows us to compute g_{i+1} from g_i recursively as follows. Constructing g_1 is trivial. By Theorem 4, we can compute $g_i * g_i$ from g_i and so we can compute g_{i+1} from g_i in $O(2^n n^2)$ ring operations.

However, if we look carefully, we can compute g_{i+1} from g_i in $O(2^n n)$ ring operations as follows. First, since we need the values of $(g_i * g_i)(X)$ only when $|X| = i+1$, we only need to compute $(g_i * g_i)(i+1, X)$ in (C). So if we are given $(\hat{g}_i \circledast \hat{g}_i)(i+1, X)$ for all subsets X of V, then we can compute $(g_i * g_i)(i+1, X)$ for all subsets X of V with $O(2^n n)$ ring operations. Second, to compute $(\hat{g}_i \circledast \hat{g}_i)(i+1, X)$ for all subsets X of V in (B), we need $\hat{g}_i(j, X)$ for all $1 \le j \le i$. However, observe that $\hat{g}_i(j, X) = \sum_{\substack{S \subseteq X \\ |S|=j}} g_i(S) = \hat{g}_j(j, X)$ whenever $0 < j \le i$ and so if $j < i$, we already have the values of $\hat{g}_i(j, X)$. So we only need the values of $\hat{g}_i(i, X)$, which can be computed in $O(2^n n)$ ring operations by (A). Therefore we can compute the values of $(g_i * g_i)(X)$ and $g_{i+1}(X)$ for all subsets

X of size $i+1$ in $O(2^n n)$ ring operations and hence we can build g_n from g_1 in $O(2^n n^2)$ ring operations.

Since $g_i(X) \in \{0,1\}$, the numbers appearing during the ring operations do not exceed n bits. Two n-bit integers can be multiplied in time $O(n \log n \log\log n)$ [7] (recently improved to $n \log n\, 2^{O(\log^* n)}$ in [3]), and therefore each ring operation can be done in time $O(n \log n \log\log n)$. Overall we we can compute g_n from g_1 in time $O(2^n n^3 \log n \log\log n)$ and so we can answer whether $\mathrm{w}(f,V) \leq k$ in time $O(2^n n^3 \log n \log\log n)$. $\square$

Theorem 4. *Let V be a finite set with n elements. Let f be an integer-valued function defined on the subsets of V. Let $M = \max_{X\subseteq V}|f(X)|$. We assume that f is given by an oracle and each oracle call takes time $O(\alpha)$. Then we can compute the width of f on V exactly in time $O(2^n(n^3 \log n \log\log n \log M + \log^2 M + \alpha))$.*

Proof. We call the oracle 2^n times to precompute values $f(X)$ for all subsets X of V. Then by binary search we find k such that the width of f on V is at most k. For each fixed k, we first compute whether or not $f(X) \leq k$; this takes time $O(2^n \log M)$. Then we compute g_n in time $O(2^n n^3 \log n \log\log n)$ for fixed k. Since the width of f on V is within $-M$ and M, we can find the minimum k in $O(\log M)$ tries. $\square$

We remark that the constant hidden in $O(2^n(n^3 \log n \log\log n \log M + \log^2 M + \alpha))$ can be reduced when $f(X) = f(V \setminus X)$ for all subsets X of V. This is because if the width is at most k, then there is a decomposition T of f-width at most k such that the root splits V into two sets of at most $m = \lfloor 2n/3 \rfloor$ vertices and therefore $(g_m * g_m)(V) = 1$ if and only if $\mathrm{w}(f,V) \leq k$. So in this case, it is not necessary to compute g_{m+1}, g_{m+2}, …, g_n and this would save about 1/3 of the computation.

4 Applications

4.1 Rank-width

Let $G = (V, E)$ be a simple graph. The *cut-rank* function $\rho_G(X)$ is defined as the rank of the $X \times (V \setminus X)$ matrix $A = (a_{ij})_{i \in X, j \in V \setminus X}$ over GF(2) where $a_{ij} = 1$ if $ij \in E$, and $a_{ij} = 0$ if $ij \notin E$. Rank-width was defined by Oum and Seymour [5]. The *rank-width* of a graph G can be equivalently defined as the width of ρ_G on V.

Then Theorem 2 implies the following.

Corollary 5. *The rank-width of an n-vertex simple graph can be computed in time $O(2^n n^3 \log^2 n \log\log n)$.*

Proof. Apply Theorem 2 with $M \le \lfloor n/2 \rfloor$ and $\alpha = O(n^3)$. □

4.2 Carving-width

Carving-width was introduced by Seymour and Thomas [8]. For a graph $G = (V, E)$ and a subset X of V, let $\eta_G(X)$ be the number of edges of G having one end in X and the other end in $V \setminus X$. Then the *carving-width* can be defined as the width of η_G on V.

Corollary 6. *The carving-width of a graph G (possibly with parallel edges) can be computed in time $O(2^n(n^3 \log n \log\log n \log m + m))$, when n is the number of vertices and m is the number of edges.*

Proof. We have $M = \max_{X \subseteq V} |\eta_G(X)| \le m$. □

4.3 Branch-width

Branch-width was introduced by Robertson and Seymour [6]. For a graph $G = (V, E)$ and a subset X of the edge-set E, let $b_G(X)$ be the number of vertices meeting an edge in X as well as an edge in $E \setminus X$. Then the *branch-width* can be equivalently regarded as the width of b_G on E. So if $e = |E|$ and $n = |V|$, then we can obtain an exact algorithm to compute the branch-width of a graph in time $2^e e^{O(1)}$. However, there is a faster algorithm known; Fomin, Mazoit, and Todinca [2] constructed an exact algorithm to compute branch-width of graphs in time $(2\sqrt{3})^n n^{O(1)}$.

Generalizing branch-width of graphs, branch-width of a connectivity function was defined similarly. For a finite set V, an integer-valued function f on the subsets of V is a *connectivity function* if $f(X) = f(V \setminus X)$, $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$, and $f(\emptyset) = 0$. The *branch-width* of a connectivity function f can be equivalently defined as the width of f on V. Rank-width, carving-width, and branch-width of graphs as well as branch-width of matroids are instances of branch-width of a connectivity function. So theorem 2 applies to any instances of branch-width of a connectivity function. For instance, the branch-width of an n-element matroid can be computed in time $O(2^n(n^3 \log^2 n \log\log n + \alpha))$ when an oracle answers the connectivity of the matroid in time $O(\alpha)$.

Acknowledgment. I would like to thank Dieter Kratsch. He posed the question and suggested to check Theorem 2.

References

[1] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74, New York, NY, USA, 2007. ACM.

[2] F. Fomin, F. Mazoit, and I. Todinca. Computing branchwidth via efficient triangulations and blocks. *Discrete Appl. Math.*, 2008. Accepted.

[3] M. Fürer. Faster integer multiplication. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 2007. ACM.

[4] P. Hliněný and S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

[5] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[6] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991.

[7] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[8] P. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.