HALG2PACE - Solving the One-Sided Crossing Minimization Problem Parameterize by Cutwidth

Narek Bojikian
Humboldt University of Berlin, Germany

Abstract

In this article, we describe 'HALG2PACE', a solver for the one-sided crossing minimization problem on graphs given together with a low-cutwidth linear arrangement. This solver was developed as part of the PACE challenge 2024 - parameterized track. The solver is based on a dynamic programming scheme over the given linear arrangement, and admits FPT running time with single-exponential dependence on the cutwidth of the given linear arrangement. The solver is implemented in C++ and meets the requirements presented by PACE challenge. The solver was submitted on optil.io under the username 'narekb95' and is available on github at `https://github.com/narekb95/ocr-ctw`

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases PACE Challenge, Cutwidth

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements I would like to thank Sophia Heck for a very helpful discussion about this problem.

1 Preliminaries

For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. A *Permutation* π of a set S is a bijective mapping from S to $[|S|]$. For $i \in [|S|]$ we denote by π_i the element $\pi^{-1}(i)$ of S. Each permutation π of a set corresponds (bijectively) to a linear ordering $\leq_\pi$, where for $u, v \in S$ it holds that $u \leq_\pi v$ if $\pi(u) \leq \pi(v)$. We will use these two terms interchangeably. In particular, we will call π an ordering, when we mean the ordering $\leq_\pi$ underlying the permutation π.

Given a graph $G = (V, E)$ and a vertex v of G, we denote by $N_G(v)$ the neighborhood of v in G. We define $N_G[v] = N_G(v) \cup \{v\}$ as the closed neighborhood of v in G. We omit the subscript G when the graph is clear from the context.

A two-layered drawing of a bipartite graph $G = (A, B, E)$ is a drawing that maps its vertices into two different horizontal lines, such that the vertices of U are mapped to one line, and the vertices of W are mapped to the other. Edges are drawn as straight-line segments between the points corresponding to their endpoints. A two-layered drawing is given as a mapping $\mu \colon V \to \mathbb{R}^2$. Implicitly, μ maps each edge $\{u, v\} \in E$ to the line segment between $\mu(u)$ and $\mu(v)$.

Let $X \in \{U, W\}$. Given an ordering π over X, a two-layered drawing μ of G respects π, if the order of the images of the vertices of X on the underlying horizontal line matches the order of the vertices of X themselves given by π, i.e. for $u, v \in X$, and for $\mu(u) = (x_u, y_0)$, and $\mu_v = (x_v, y_0)$, it holds that $x_u \leq x_v$ if $u \leq_\pi v$.

In a two-layered drawing μ, and for two edges $e_1, e_2 \in E$, we say that e_1 and e_2 cross in μ, if $\mu(e_1)$ and $\mu(e_2)$ intersect in a point that is not an endpoint of either line segment. The *number of crossings* of μ is the number of unordered pairs of edges $\{e_1, e_2\} \in \binom{E}{2}$ such that e_1 and e_2 cross in μ. The following observations are Folklore. We refer to [2] for further details.

▶ **Lemma 1.** *Given a bipartite graph $G = (U, W, E)$, together with two orderings π_1, π_2 over U and W respectively, the number of crossings of any two-layered drawing of G that respects*

PACE2HALG - Solving the one-sided crossing minimimization problem parameterized by cutwidth.
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

π_1 and π_2 is invariant of the drawing itself, and is determined solely by π_1 and π_2. We call the number of crossings of any such drawing as the crossing number *of* (π_1, π_2).

▶ **Definition 2.** *The* one-sided crossing minimization problem *is defined as follows: Given a bipartite graph $G = (U, W, E)$ together with a fixed ordering π_1 over U, asked is a permutation π_2 over W that minimizes the crossing number of (π_1, π_2).*

A linear arrangement of a graph $G = (V, E)$ is linear ordering $\ell = v_1, \ldots, v_n$ over V. Let $V_i = \{v_1, \ldots v_i\}$. For $v_i \in V$, we define the cut of ℓ at v_i as the set of edges having one endpoint in V_i and the other in $V \setminus V_i$, and call it the ith cut of ℓ. The cutwidth of a linear arrangement ℓ is the largest size of a cut of ℓ. For δ the largest degree of a graph, it holds that the cutwidth of any linear arrangement of this graph is at least $\lceil \delta/2 \rceil$. The cut-graph $H_i = (L_i, R_i, E_i)$ is given by the edges of the ith cut together with their endpoints, where $L_i \subseteq V_i$ are the endpoints in V_i, and $R_i \subseteq V \setminus V_i$ are the other endpoints. We define $A_i = V_i \setminus A_i$.

2 The algorithm

We introduce some notation for our algorithm. Along this work, we assume $G = (U, W, E)$ is the input graph, and ℓ a linear arrangement of G. Let k be the largest size of a cut of ℓ. Let $n = |V|$ and $m = |E|$. Let π_1 be the give fixed ordering over U, and π_2 be the asked ordering. Finally, for two different vertices $v_1, v_2 \in W$, we define $c(v_1, v_2)$ as

$$c(v_1, v_2) = |\{(w_2, w_1) \in N(v_2) \times N(v_1) \colon w_2 <_{\pi_1} w_1\}|.$$

For two disjoint sets of vertices $X, X' \subseteq W$, we define

$$c(X, X') = \sum_{(v_1, v_2) \in X \times X'} c(v_1, v_2).$$

Moreover, for a set $X \subseteq W$ we define $c(X)$ as the minimum crossing number of (π_1, π') in $G[V_1, X]$ over all orderings π' of X.

▶ **Algorithm 1.** *We provide a subroutine to compute $c(X)$ for a small set X in time $2^{|X|}\operatorname{poly}(|X| + k)$. The algorithm follows a similar approach to the well-known dynamic programming algorithm for the Hamiltonian Cycle given by Held and Karp [3] and independently by Bellman [1]. The subroutine iterates over all subsets of X in an increasing order (given by the subset relation), and computes an optimal ordering induced by each subset using the recursive formula*

$$c(X) = min\{c(X \setminus \{v\}) + c(X \setminus \{v\}, v) \colon v \in X\}.$$

We call a pair of vertices $v_1, v_2 \in W$ suited, if $c(v_1, v_2) \cdot c(v_2, v_1) = 0$. Our algorithm is based on the following lemma:

▶ **Lemma 3** ([2, Fact 5])**.** *Let $u, v \in W$ be two suited vertices. In any optimal solution π_2, it holds that if $u \leq_{\pi_2} v$ then $c(u, v) = 0$.*

Our algorithm follows a dynamic programming approach over the cuts of the linear arrangement ℓ. For $i \in [n]$, we define the set $S_i = W \cap V(H_i)$ as the set of vertices of W that are incident to edges of the ith cut. The dynamic programming tables are indexed by subsets of S_i. Formally, for $i \in [n]$ let $\mathcal{S}_i = \mathcal{P}(S_i)$ the power-set of S_i. we define the dynamic programming tables as vectors $T_i \in \mathbb{N}^{\mathcal{S}_i}$, where for $X \subseteq S_i$ it holds that:

$$T_i[X] = c(A_i \cup X) + c(A_i, V \setminus (A_i \cup X)). \tag{1}$$

Intuitively, for each subset $X \subseteq S_i$ we fix $A_i \cup X$ as a prefix of the final ordering and count the optimal number of crossings in an optimal ordering of $A_i \cup X$ plus the number of crossings between A_i and the rest of S_i. It follows by Lemma 3 that the vertices of A_i precede any vertex of $W \setminus (A_i \cup S_i)$ in an optimal ordering.

At each cut i our algorithm proceeds as follows: Let $S'_i = S_{i-1} \cup S_i$, and $F_i = S_{i-1} \setminus S_i$. We call F_i the set of forget vertices at the ith cut. The algorithm first computes both S' and F. Then for each set X with $F_i \subseteq X \subseteq S'_i$, i.e. X is a subset of S'_i and a super set of F_i, let $X' = X \setminus F_i$. The algorithm computes $T_i[X']$ as follows:

$$T_i[X'] = \min_{Y \subseteq X} c_1 + c_2 + c_3 + c_4, \tag{2}$$

- where $c_1 = T_{i-1}(Y)$,
- $c_2 = c(X \setminus Y)$,
- $c_3 = c(Y, X \setminus Y)$,
- and $c_4 = c(F, S_i \setminus X')$.

Intuitively, the algorithm fixes an ordering with $A_{i-1} \cup Y$ as a prefix, and append $X \setminus Y$ to this prefix (using the best ordering for $X \setminus Y$ given by Algorithm 1). Now we explain each summand in the states sum:

- c_1 is the number of crossings induced by the prefix $A_{i-1} \cup Y$ and the crossings between A_{i-1} and $V \setminus A_i$.
- c_2 is the number of crossings in induced by $X \setminus Y$.
- c_3 is the number of crossings introduced by appending Y to the prefix ordering (edges between A_{i-1} and Y are accounted for in c_1).
- c_4 is the number of edges between F and $V \setminus (A_{i-1} \cup X)$, since $F = A_i \setminus A_{i-1}$.

3 Implementation details

Our solver starts by removing isolated vertices from the graph, building a new graph G'. This ensures that each vertex has at least one neighbor. The solver sorts the adjacencies of each vertex by their order on the linear arrangement ℓ and assigns a range to each vertex, given by the first and the last index (in the linear arrangement) of a neighbor of this vertex. This allows to compute crossing numbers between two vertices in polynomial time in k. After computing an optimal solution, the solver assigns to each vertex its original id, and appends isolated vertices in an arbitrary order.

We use bit-masks to represent sets. To iterate over all supersets of F that are subsets of S' with constant time steps, we compute $S'_i \setminus F_i$, iterate over all its subsets X' and compute $X = X' \cup F$ as the sets we are looking for.

In order to output the ordering we keep track of all sets S_i, and for each subset $X \subset S_i$ we keep track of the set $X \setminus Y$ that was appended to $A_i \cup Y$ to compute an optimal ordering at X. We use backtracking to generate each suffix at each step, and for each such suffix we use an additional call to Algorithm 1 to output an optimal ordering for this set.

4 Sketch of Correctness

It follows from Lemma 3 that an optimal ordering has all vertices of A_i ordered before $V \setminus (A_i \cup S_i)$. Hence, the correctness and the optimality of the algorithm follow by induction over i where we show that the recursive formula for T_i Equation (2) computes exactly the number of cuts presented in the definition of T_i Equation (1).

Since each cut-edge has at most one endpoint in W, ist holds that $|S_i| \leq k$. The running time of the algorithm can be bounded by $3^k \operatorname{poly}(k)n$, since we iterate over all cuts i, and for each we iterate over all subsets of S_i, and over each subset of these subsets. We spend polynomial time at each such subset of a subset, so the running time can be bounded in

$$\sum_{i=1}^{n} \sum_{S \subseteq S_i} \sum_{X \subseteq S} \operatorname{poly}(k) = n3^k \operatorname{poly}(k).$$

References

1. Richard Bellman. *Combinatorial processes and dynamic programming.* Rand Corporation, 1958.
2. Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004. URL: https://doi.org/10.1007/s00453-004-1093-2, doi:10.1007/S00453-004-1093-2.
3. Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In Thomas C. Rowan, editor, *Proceedings of the 16th ACM national meeting, ACM 1961, USA*, page 71. ACM, 1961. doi:10.1145/800029.808532.