Dummy title

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 user name 'narekb95' and is available on github at `https://github.com/narekb95/ocr-ctw`

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases PACE Challenge, cutwidth

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements [TODO]

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.

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 V^1 are mapped to one line, and the vertices of V^2 are mapped to the other. Edges are drawn as straight-line segments between the points corresponding to their endpoints. A two-layered is given as a mapping $\mu\colon V \to \mathbb{N}^2$. Implicitly, μ maps each edge $\{u, v\} \in E$ to the line segment between $\mu(u)$ and $\mu(v)$. For $i \in [2]$, V^1, given an ordering π over V_i, a two-layered drawing of G respects π if the order of the points corresponding to the images of the vertices of V_i on the underlying horizontal line matches the order of the vertices themselves given by π.

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 μe_2 intersect in a point that is not an endpoint of either line segments. 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 more information.

▶ **Lemma 1.** *Given a bipartite graph $G = (V^1, V^2, E)$, together with two orderings π_1, π_2 over V^1 and V^2 respectively, the number of crossings of any two-layered drawing of G that respects π_1 and π_2 is invariant of the drawing itself, and determined solely by π_1 and π_2. We call this number of crossings of any such map as the* crossings number *of (π_1, pi_2).*

▶ **Definition 2.** *The* one-sided crossing minimization problem *is defined as follows: Given a bipartite graph $G = (V^1, V^2, E)$ together with a fixed ordering π_1 over V^1, asked is a permutation π_2 over V^2 that minimizes the crossing number of (π_1, π_2).*

A linear arrangement of a graph $G = (V, E)$ is linear ordering $\ell = v_1 \leq \dots \leq v_n$ over the vertices of the graph. Let $V_i = \{v_1, \dots v_i\}$. 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 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.

2 The algorithm

Finally, we introduce some notation for our algorithm. Along this work, we assume $G = (V^1, V^2, E)$ 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 V^1, and π_2 be the asked ordering. Finally, for two different vertices $v_1, v_2 \in V^2$, 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 $X, X' \subseteq V_2$, we define

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

Moreover, for a set $X \subseteq V^2$ we define $c(X)$ as the minimum crossing number of (π_1, π') of $G[V_1, X]$ over all orderings π' of X.

▶ **Algorithm 1.** *We start a subroutine to compute $c(X)$ for a small set X in time $2^n \operatorname{poly}(n)$. 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 increasing order (given by the subset relation), and computes an optimal ordering induced by each subset, by the recusive formula*

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

▶ **Definition 3.** *Given a bipartite graph $G = (V_1, V_2, E)$ together with a permutation π_1 over V_1, a pair of vertices $(v_1, v_2) \in V_2^2$ is called suited (in respect to π_1), if $w_1 \leq_{pi_1} w_2$ for each $w_1 \in N_G(v_1)$ and $w_2 \in N_G(v_2)$.*

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

▶ **Lemma 4** ([2, Fact 5])**.** *Let $u, v \in V^2$ 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 $S_i = V^2 \cap V(H_i)$ as the set of vertices of V^2 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 $X \subseteq S_i$ we define $T_i[X]$ as follows:

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

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

$$T_i[X'] = \min_{Y \subseteq X} c_1 + c2 + c3, \tag{2}$$

where

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

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

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

3 Implementation details

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 $V_i \cup Y$ 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 4 that an optimal ordering has all vertices of V_i ordered before $V \setminus (V_i \cup S_i)$. Hence, the correctness and the optimality of the algorithm follow by induction over T_i where we show that the recursive formula Equation (2) computes exactly the number of edges presented in the definition of T_i Equation (1).

Since each cut-edge has at most one endpoint in V_1, 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.