Simple Linear Time Approximation Algorithm for Betweenness

Yury Makarychev
Microsoft Research New England

Abstract

In this technical report, we present a simple combinatorial algorithm for the Betweenness problem. In the Betweenness problem, we are given a set of vertices and *betweenness* constraints. Each betweenness constraint of the form $u \rightsquigarrow \{v, w\}$ requires that the vertex u lies between vertices v and w. Our goal is to find an ordering of vertices that maximizes the number of satisfied constraint. In 1995, Chor and Sudan [2] constructed an SDP algorithm that satisfies half of all constraints if the instance is (completely) satisfiable. We present a simple linear time algorithm with the same approximation guarantee.

1 Introduction

In the Betweenness problem, we are given a set V of n vertices and a set $\mathcal{C}$ of m *betweenness* constraints. A linear order $<$ on V satisfies a betweenness constraint $u \rightsquigarrow \{v, w\}$ if either $v < u < w$ or $w < u < v$ (we identify constraints $u \rightsquigarrow \{v, w\}$ and $u \rightsquigarrow \{w, v\}$). Our goal is to find an ordering of vertices that maximizes the number of satisfied constraint.

In 1979, Opatrny [3] proved that the decision version of the problem is NP-hard. Chor and Sudan [2] showed that moreover it is MAX SNP hard. The approximability of the problem crucially depends on whether the instance is (completely) satisfiable or not. There is always a trivial 3-approximation algorithm: choose a random ordering. In general, one cannot get a better approximation factor as was recently shown by Charikar, Guruswami, and Manokaran [1] (assuming the Unique Games Conjecture). However, Chor and Sudan [2] proved that if the instance is satisfiable, it is possible to satisfy at least half of all constraints. Their approximation algorithm is based on semidefinite programming and thus it is not very fast. In this technical report, we design a very simple combinatorial algorithm with running time $O(m+n)$ that achieves the same approximation guarantee.

2 Algorithm

Definition 2.1. *Given a subset of vertices $A \subset V$, we say that a vertex $u \in A$ is free in A if there is no constraint of the form $u \rightsquigarrow \{v, w\}$ with $v, w \in A$.*

Note that if a set has a free vertex, it is easy to find it. We will show now that if the instance is satisfiable then every set has a free vertex.

Claim 2.2. *Suppose the betweenness instance is satisfiable. Then there is a free vertex in every non-empty subset A of vertices.*

Proof. Consider a vertex ordering that satisfies all constraints. Let u be the first (least) vertex w.r.t. this ordering in A. Then u does not lie between any two vertices of A. Since all constraints are satisfied, u is a free vertex in A. □

Corollary 2.3. *Suppose the betweenness instance is satisfiable. Then there exists an ordering $u_1, \dots, u_n$ of V such that each u_i is free in $\{u_i, \dots, u_n\}$.*

Proof. Let u_1 be a free vertex in V, u_2 be a free vertex in $V \setminus \{u_1\}$, ..., u_{i+1} be a free vertex in $V \setminus \{u_1, \dots, u_i\}$. Since $\{u_i, \dots, u_n\} = V \setminus \{u_1, \dots, u_{i-1}\}$, the corollary holds. □

We denote the set $\{u_i, \dots, u_n\}$ by A_i. Let $\mathcal{C}_i$ be the set of constraints that involve only vertices from A_i.

Recall that the ordinal sum of two ordered sets $(A, <_A)$ and $(B, <_B)$ is the ordered set on $A \cup B$ in which every element of A is less than any element of B; the orders on A and B are defined by $<_A$ and $<_B$ correspondingly. We will denote the sum of $(A, <_A)$ and the ordered set $\{u\}$ (formally, $(\{u\}, \varnothing)$) by $(A, <_A) + \{u\}$, and similarly the sum of $\{u\}$ and $(A, <_A)$ by $\{u\} + (A, <_A)$.

Lemma 2.4. *We recursively define a linear order $<_i$ on A_i as follows. Let $<_n$ be the trivial order on A_n. Given $<_{i+1}$ consider two orders $(A_i, <_i^L) \equiv \{u_i\} + (A_{i+1}, <_{i+1})$ and $(A_i, <_i^R) \equiv (A_{i+1}, <_{i+1}) + \{u_i\}$. Let $(A_i, <_i^L)$ be the*

```
function $OrderSet$
    Input: subset of vertices $A_i$;
    Output: linear order $order = (A_i, <_i)$.
begin
    if $A$ contains two or less elements return any order.
    find a free vertex $u_i$ in $A_i$
    $(A, <_{i+1}) = OrderSet(A \setminus \{u\})$
    return the better of the two orders, $\{u_i\} + (A, <_{i+1})$ and $(A, <_{i+1}) + \{u_i\}$
end
```

Figure 1: Approximation Algorithm for Betweenness. $OrderSet(V)$ returns a solution to the problem.

one of the two that satisfies more constraints in $\mathcal{C}_i$ (break the ties arbitrarily). Then $(A_i, <_i)$ satisfies at least half of constraints in $\mathcal{C}_i$. In particular, the order $<_1$ on $V = A_1$ satisfies half of all constraints.

Proof. The proof is by induction. For $i = n$ the statement trivially holds. Consider the order $<_i$. Observe that $<_i$ satisfies a constraint C from $\mathcal{C}_{i+1}$ if and only if $<_{i+1}$ satisfies C. Therefore, $<_i$ satisfies at least half of constraints in C_{i+1}.

Now consider a constraint in $\mathcal{C}_i \setminus \mathcal{C}_{i+1}$. Since u_i is free in A_i, the constraint has form $u_j \rightsquigarrow \{u_i, u_k\}$, where $j > i$ and $k > i$. Note that if $u_j <_{i+1} u_k$ then $u_i <_i^L u_j <_i^L u_k$ and thus the order $<_i^L$ satisfies the constraint; similarly, $u_k <_{i+1} u_j$ then the order $<_i^R$ satisfies the constraint. That is, each constraint in $\mathcal{C}_i \setminus \mathcal{C}_{i+1}$ is satisfied by (exactly) one of the orders $<_i^L$ and $<_i^R$. So one of them satisfies at least half of constraints in $\mathcal{C}_i \setminus \mathcal{C}_{i+1}$ and hence in $\mathcal{C}_i$. $\square$

The lemma is constructive and provides an approximation algorithm for the problem: first find an ordering $u_1, \ldots, u_n$ from Corollary 2.3, then recursively find orders $<_n, \ldots, <_1$. Figure 1 presents the algorithm.

variable	**how we store it**
A_i	Each vertex has a boolean flag that says whether it belongs to A_i. Initially, all vertices are in A_i. We can remove a vertex from A_i in constant time.
$\mathcal{C}_i$	Each constraint has a boolean flag that says whether it belongs to $\mathcal{C}_i$. Initially, all constraints are in C_i.
list of free vertices in A_i	We store all free vertices in a stack S. For each vertex u, we also store the number d_u of constraints in $\mathcal{C}_i$ of the form $u \rightsquigarrow \{v, w\}$. Initially, S contains all vertices free in V. Every time we remove a constraint $u \rightsquigarrow \{v, w\}$ from $\mathcal{C}_i$, we decrease d_u. If d_u becomes equal 0, we push u onto S. Using S, we can find a free vertex in constant time (i.e., pop it from A_i).
order $<_i$	We store the order $<_i$ in a circular buffer B of capacity n. We can insert an element before or after all other elements in constant time. Every vertex in the support of $<_i$ stores a pointer to its location in the buffer. We can compare two elements w.r.t. $<_i$ in constant time.

Figure 2: We use the following data structures to store all the variables.

Theorem 2.5. *Given a satisfiable instance of the Betweenness problem, the algorithm presented in Figure 1 outputs an ordering that satisfies at least half of all constraints. The algorithm runs in time $O(m + n)$.*

Proof. The correctness of the algorithm follows from Corollary 2.3 and Lemma 2.4. We shall explain now how to implement the algorithm so that it runs in linear time. We store the variables in data structures described in Figure 2.

Let the degree of a vertex be the number of constraints it participates in. Denote the degree of u by $\deg(u)$. Clearly, $\sum_{u \in V} \deg(u) = 3m$.

Let us bound the number of operations in one iteration of the algorithm.

- Execution of step 5 takes constant time.
- At step 6, we pop u_i from S. That takes constant time.
- At step 7, we remove u_i from A_i, increment i, and remove constraints involving u_i from $\mathcal{C}_i$ (for each removed constraint $v \rightsquigarrow \{u, w\}$, we update d_v, and, if necessary, S; see Figure 2). The total time is $O(\deg(u_i) + 1)$.

- At step 8, we loop through all constraints of the form $v \rightsquigarrow \{u_i, w\}$ in $\mathcal{C}_i$ and determine whether for the majority of constraints $v <_i w$ or $w <_i v$. Correspondingly, we insert u_i into B at the left or right end. That takes time $O(\deg(u_i) + 1)$.

Therefore, each iteration takes time $O(\deg(u_i) + 1)$. The total running time is $O\left(\sum_{u \in V}(\deg(u) + 1)\right) = O(m + n)$. $\square$

References

[1] M. Charikar, V. Guruswami, and R. Manokaran. *Every Permutation CSP of arity 3 is Approximation Resistant,* to appear in IEEE Conference on Computational Complexity, 2009.

[2] B. Chor and M. Sudan. *A geometric approach to betweenness,* SIAM Journal on Discrete Mathematics, 11(4):511-523, Nov. 1998.

[3] J. Opantry. *Total Ordering Problem,* SIAM Journal on Computing, 8(1):111–114, Feb. 1979.