Relaxed Voting and Competitive Location on Trees under Monotonous Gain Functions

J. Spoerhase and H.-C. Wirth

March 20, 2007

We examine problems of placing facilities in a tree graph to serve customers. The decision of placement is driven by an election process amongst the users, where the user preference is modeled by distances in the tree. Relaxed user preferences introduce a tolerance of users against small differences in distances. Monotonous gain functions are introduced to provide general results of which well known problems including Simpson, security, Stackelberg and Nash are special cases.

1 Introduction and Preliminaries

Location problems on graphs are characterized as follows: An edge weighted graph models distances in a universe. Weighted nodes of the graph represent customers and their demands. The customers have to be served by facilities which can be placed at the nodes of the graph or at inner points of the edges. The goal is to find an optimal placement of the facilities. Several objective functions are in common use, e.g. maximum or average distance to the closest facility (*center* and *median* problem), or total sum of distances and costs for opening the selected set of facilities (*facility location* problem).

Voting problems are a means of modeling the process of finding compromises in a group of social individuals. Here global decisions are often based on individual preferences which can, more or less explicitely, be treated as a formal election between alternative solutions. It is assumed that the resulting solutions are accepted by all participants and hence stable, since they are preferred by a significant majority of the users.

Voting location problems are a way to combine both lines of research: The static universe is modeled by a weighted graph, while the optimal placement of facilities is the result of an election process performed by the individual users. Here the user preferences are fully determined by the distances in the underlying graph. In this paper we are only investigating *single* location problems where only one facility is subject to be placed, and we use trees as the underlying graphs.

In the problems under investigation in this paper we are particularly interested in stable solutions which are characterized by the fact that the chosen candidate is confronted with only weak oppositions. There are several measures for this stability, including *Simpson* and *security score* as defined later.

It can be observed that in this model small differences between distances can have a huge impact on the result. This is not a desired property, since it is not reflected by the behavior of users in the real world: here it is typical that users are undecided between facilities at similar distances, and distinguished preferences emerge only when there are significant differences between the distances. This is modeled in the following by *relaxed* user preferences as introduced in [CM03].

Applications of voting location problems can be found e.g. in the area of public resources planning. In the classical facility location problem, decisions are performed based solely on abstract cost functions which reflect mainly the view of the supplier. In contrast to that, a voting solution can guarantee a wide acceptance since the opinions of the customers are taken into account.

1.1 Problem Formulation

The definitions in this section follow closely those of [CM03, CM02]. The input instance of the problems is given as a tree $T = (V, E)$. A positive edge weight function $d\colon E \to \mathbb{R}^+$ denotes lengths of edges and induces a distance function $d\colon V \times V \to \mathbb{R}_0^+$. Nonnegative node weights $w\colon V \to \mathbb{R}_0^+$ specify the demand of individual user nodes. A nonnegative number $\alpha \in \mathbb{R}_0^+$ is used as a parameter to describe the users' tolerance against small differences in distances as follows:

Definition 1 (Relaxed user preference) A user u *prefers* node x over node y, denoted by $x \prec_u y$, if

$$d(u, x) < d(u, y) - \alpha\,.$$

The user u *is undecided*, $x \sim_u y$, if $|d(u, x) - d(u, y)| \leq \alpha$.

We use the following notation: The set of users preferring x over y is denoted by $U_\alpha(x \prec y) := \{\, u \in V \mid x \prec_u y \,\}$, and its weight by $W_\alpha(x \prec y) := w(U_\alpha(x \prec y))$. Now we are enabled to formulate the main problems under investigation in this paper.

Definition 2 (Relaxed Simpson) The α-*Simpson score* of a candidate node x is defined as

$$\Gamma_\alpha(x) := \max_{y \in V} W_\alpha(y \prec x)\,.$$

The α-*Simpson score* of a graph is defined as $\Gamma_\alpha^* := \min_{x \in V} \Gamma_\alpha(x)$. An α-*Simpson solution* of a graph is a candidate node x with $\Gamma_\alpha(x) = \Gamma_\alpha^*$.

Observe that in the definition of the Simpson score of a node x, all undecided users are actually treated as if they voted for x. This can pretend a high stability of a solution

which does not exist. To take into account only the sets of users actually preferring x and y, respectively, and to ignore the undecided users, we set

$$\Delta_\alpha(y \prec x) := \begin{cases} W_\alpha(y \prec x) - W_\alpha(x \prec y) & \text{if } d(x,y) > \alpha \\ -\infty & \text{otherwise} \end{cases}$$

and define the security score as follows:

Definition 3 (Relaxed Security) The *α-security score* of a candidate node x is defined as

$$\Delta_\alpha(x) := \max_{y \in V} \Delta_\alpha(y \prec x)\,.$$

The *α-security score* of a graph is defined as $\Delta^*_\alpha := \min_{x \in V} \Delta_\alpha(x)$. An *$\alpha$-security solution* of a graph is a candidate node x with $\Delta_\alpha(x) = \Delta^*_\alpha$.

The definition of security is slightly different from that of [CM02]. According to our definition for a given candidate node it is required that any possible opposition node has a distance of more than α to the candidate node, in other words, the opposition is anticipated to attain a significantly different standpoint. This is motivated by the following observation. Assume we define $\Delta'_\alpha(x) := \max_y\big(W_\alpha(y \prec x) - W_\alpha(x \prec y)\big)$ without the restriction $d(y,x) > \alpha$. Then for any candidate node x, by selecting opposition $y := x$ we have $\Delta'_\alpha(y \prec x) = 0$ and thus always $\Delta'_\alpha(x) \geq 0$. This trivial property blurs the view to the details of the graph instance, as the relation $\Delta'_\alpha(x) = \max\{0, \Delta_\alpha(x)\}$ always holds. These details can be revealed when we add the requirement $d(y,x) > \alpha$. In other words, the function Δ_α yields a refinement (namely a subset) of the solutions described by the unrestricted Δ'_α.

Throughout the paper we assume that the tolerance bound α is fixed. So we omit indication of α in all notation where there is no ambiguity.

We use the following notation for trees. The unique path between two nodes u, v in T is denoted by $P(u,v)$. Let $r, v \in V$ be two nodes in T and let the tree be rooted at node r. Then the subtree of T hanging from v is denoted by $T_r(v)$.

1.2 Related Work and Contribution of this Paper

Single voting location under the unrelaxed user preference model, i.e. $\alpha = 0$, and allowing placement of facilities on inner points of edges (see Section 3 for a formal definition) has been discussed in [HTW90]. For general graphs there are no fast algorithms known albeit the problems are polynomial time solvable. If the underlying graph is a tree, then most of these unrelaxed problems can be solved in linear time. This is due to the fact that they all become equivalent to the median problem [HTW90].

This equivalence is no longer true (see Figure 1 for an example) if we turn over to the relaxed user preference model ($\alpha \geq 0$) which has been introduced in [CM03]. Clearly this generalization does not lead to problems which are easier solvable. However, there is an algorithm for the relaxed Simpson problem on trees with running time $O(n \log n)$ as stated in [NSW07].

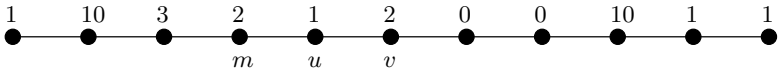


Figure 1: Example where median $\{m\}$, relaxed security solution $\{u\}$ and relaxed Simpson solution $\{v\}$ are unique and pairwise different. The edge lengths are 1, the indifference is $\alpha = 4$.

In this paper we provide fast algorithms for the relaxations of security solution, Stackelberg solution and Nash solution (see Section 4 for an exact definition) on trees. To this end we introduce the notion of *monotonous gain function* which allows to describe all those problems in a uniform way and to develop a general algorithmic framework.

The paper is organized as follows: in Section 2 we formally introduce monotonous gain functions and provide the general scheme for formulating algorithms which evaluate the *score* of a single node and find a *solution* in the tree, i.e., a node with optimal score. In Section 3 we extend the discussion to the *absolute* score: here it is allowed to place candidates and oppositions not only at nodes but also at inner points of edges. In Section 4 we describe how the developed framework can be applied to the several problems under investigation and suggest run time optimized implementations of the algorithms.

2 Relaxed Score and Solution under Monotonous Gain Functions

Observe that in the definition of Simpson and security scores, the only difference is that function Γ is replaced by Δ. This suggests a generalization to an arbitrary *gain function* $\Phi\colon V \times V \to \mathbb{R}$. This function maps a node pair (y, x) to the value $\Phi(y \prec x)$ which measures in some sense the influence of an opposition node y after candidate node x has already been placed into the graph. Given a gain function, the notions *score* and *solution* are defined similarly as in Section 1.1:

Definition 4 (Φ-Score and Φ-Solution) For any gain function Φ, the Φ-*score* of a candidate node x is defined as

$$\Phi(x) := \max_{y \in V} \Phi(y \prec x)\,.$$

The Φ-*score* of a graph is defined as $\Phi^* := \min_{x \in V} \Phi(x)$. A Φ-*solution* of a graph is a candidate node x with $\Phi(x) = \Phi^*$.

Clearly this definition is too weak to derive a general algorithm for Φ-score and solution beyond the limits of a trivial enumeration. A minimum requirement is that a gain function reflects the user preference induced by the graph. In particular, if we start with two nodes x, y and move them in such a way that the influence area $U(y \prec x)$ of the opposition y increases while the influence area $U(x \prec y)$ of the candidate x decreases,

one would expect that this does not decrease the value $\Phi(y \prec x)$. We call gain functions with this property to be *monotonous*:

Definition 5 (Monotonous Gain Function) A gain function $\Phi(y \prec x)$ is called *monotonous*, if for all candidate-opposition pairs (x, y) and (x', y') where

$$U(y \prec x) \supseteq U(y' \prec x') \quad \text{and} \quad U(x \prec y) \subseteq U(x' \prec y')$$

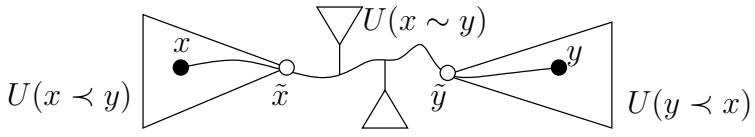
we have that

$$\Phi(y \prec x) \geq \Phi(y' \prec x') \,.$$

We will discuss in Section 4 that especially the gain functions associated with the Simpson, security, Stackelberg, Centroid, and Nash problems are in fact monotonous. Observe further that the value of a monotonous gain function is already determined by the preferences of the users; in particular $\Phi(y \prec x) = \Phi(y' \prec x')$ follows already from $U(y \prec x) = U(y' \prec x')$ and $U(x \prec y) = U(x' \prec y')$.

2.1 Computing the Relaxed Score of a Node

We first investigate the problem of computing the relaxed Φ-score of a node in a tree. Consider the case where two nodes x, y are given. By definition the three sets $U(x \prec y)$, $U(x \sim y)$, and $U(y \prec x)$ form a partition of the tree T. One can show that each of the three sets is actually a connected subtree (in the case $d(x, y) \leq \alpha$ the sets $U(x \prec y)$ and $U(y \prec x)$ are empty).



Lemma 2.1 (Front nodes) *Let x, y be nodes such that $d(x, y) > \alpha$. Then there are nodes $\tilde{x}, \tilde{y}$ on path $P(x, y)$ such that $U(x \prec y) = T_{\tilde{y}}(\tilde{x})$ and $U(y \prec x) = T_{\tilde{x}}(\tilde{y})$.*

The nodes $\tilde{x}, \tilde{y}$ are denoted as *front nodes*. These nodes are of special importance since they completely characterize the above mentioned partition of the tree, from which the desired value $\Phi(y \prec x)$ can be derived. We remark that $\tilde{x} = x$ or $\tilde{y} = y$ can occur.

Proof (of Lemma 2.1). For any node z, let the *projection* of z be the node $\bar{z}$ on $P(x, y)$ where $d(z, \bar{z})$ is minimal. Since $d(z, x) - d(z, y) = d(\bar{z}, x) - d(\bar{z}, y)$ it follows that preferences $x \prec_z y$ and $x \prec_{\bar{z}} y$ are identical. Choose $\tilde{x} \in P(x, y)$ such that $x \prec_{\tilde{x}} y$ and $d(x, \tilde{x})$ is maximal. Then the nodes on path $P(x, y)$ which prefer x are exactly those on the subpath $P(x, \tilde{x})$. If $v \in T$ is an arbitrary node, then $x \prec_v y$ if and only if its projection $\bar{v}$ prefers x, i.e., $\bar{v} \in P(x, \tilde{x})$. This is equivalent with $v \in T_y(\tilde{x})$. The situation for $\tilde{y}$ is symmetric. □

Now assume that x, y satisfy $d(x,y) > \alpha$. When we move y along the path $P(x,y)$ towards x, this increases $U(y \prec x)$ and decreases $U(x \prec y)$, hence $\Phi(y \prec x)$ does not decrease. This is true until $d(x,y) \leq \alpha$. It appears that nodes at distance α play an important role:

Definition 6 (α-Neighborhood) Let $x \in V$ be a candidate node. A node $y \in V$ is called an *α-neighbor of x* if it is the only node on the path $P(x,y)$ whose distance to x is strictly greater than α. The set of all α-neighbors of x is denoted by $N(x,\alpha)$.

From the above observations we can conclude that in order to compute $\Phi(x)$ it suffices to consider nodes y in the α-neighborhood of x:

Theorem 2.2 (Computation of Φ-Score) *For each x there is a node $y \in N(x,\alpha) \cup \{x\}$ such that $\Phi(x) = \Phi(y \prec x)$.*

Proof. Let y' be an opposition node such that $\Phi(x) = \Phi(y' \prec x)$. If $d(x,y') \leq \alpha$, then all nodes are undecided, i.e., $U(y' \sim x) = V$. Since also $U(x \sim x) = V$ we can conclude $\Phi(x \prec x) = \Phi(y' \prec x) = \Phi(x)$.

If $d(x,y') > \alpha$, then consider the α-neighbor y of x on path $P(x,y')$. Obviously, $U(y \prec x) \supseteq U(y' \prec x)$ and $U(x \prec y) \subseteq U(x \prec y')$. Thus by monotonicity of Φ we have $\Phi(y \prec x) \geq \Phi(y' \prec x)$. Since $\Phi(y' \prec x)$ was maximal this shows $\Phi(y \prec x) = \Phi(x)$. □

We have stated before that for a given pair (y,x) the value $\Phi(y \prec x)$ can be derived when we know the associated front nodes $(\tilde{y}, \tilde{x})$. Combining both results yields that in order to compute $\Phi(x)$ it suffices to enumerate all α-neighbors y and the corresponding pair $(\tilde{y}, \tilde{x})$ of associated front nodes. The following theorem provides how front nodes can be constructed.

Theorem 2.3 (Characterization of Front Nodes) *Let x be a candidate node and y be an α-neighbor of x. Then the front node $\tilde{x}$ is the $\left(\frac{d(x,y)+\alpha}{2}\right)$-neighbor of y on $P(x,y)$, and the front node $\tilde{y}$ is y itself. Thus we have*

$$U(x \prec y) = T_y(\tilde{x}) \qquad \text{and} \qquad U(y \prec x) = T_{\tilde{x}}(y)\,.$$

Proof. As in the proof of Lemma 2.1 we only consider projections of nodes on $P(x,y)$. A node $z \in P(x,y)$ prefers x if and only if

$$d(z,y) - d(z,x) > \alpha \iff 2 \cdot d(z,y) - d(x,y) > \alpha \iff d(z,y) > \frac{d(x,y)+\alpha}{2}\,.$$

The node z on $P(x,y)$ which satisfies this condition and maximizes $d(x,z)$ is the front node $\tilde{x}$. Since it simultaneously minimizes $d(z,y)$, it is the $\left(\frac{d(x,y)+\alpha}{2}\right)$-neighbor of y on the path.

The remaining claim, $\tilde{y} = y$, is clear since y is chosen to be an α-neighbor of x. □

To determine all α-neighbors of x, we perform a DFS traversal from x and maintain distances $d(x, y)$ to the root where y is the currently traversed node. If y is an α-neighbor, the corresponding front node $\tilde{x}$ can be found as follows: Let $x = v_0, v_1, \ldots, v_k = y$ be the nodes on the DFS path from x to y. Then $\tilde{x} = v_i$ where i is the maximum index such that $d(v_i, x) < (d(x, y) - \alpha)/2$. This node can be found in $O(\log k) \subseteq O(\log n)$ by binary search in the array $(v_i)_i$.

Theorem 2.4 (Enumeration of Front Node Pairs) *For a given node x, the set of all pairs $(\tilde{x}, \tilde{y})$ where y is an α-neighbor and $\tilde{x}, \tilde{y}$ are the corresponding front nodes can be computed in time $O(n \log n)$.* □

For most problems investigated in this paper the value of Φ depends only on the weights $W(y \prec x) = w(T_{\tilde{x}}(\tilde{y}))$ and $W(x \prec y) = w(T_{\tilde{y}}(\tilde{x}))$ of the subtrees hanging from the front nodes. If we assume that the weights of all possible subtrees $\{ T_u(v), T_v(u) \mid (u, v) \in E \}$ have already been computed as a preprocessing step (which needs two DFS traversals and therefore runs in $O(n)$) the function value $\Phi(y \prec x)$ can be evaluated in $O(1)$ when the front nodes $(\tilde{y}, \tilde{x})$ are known. We call a gain function with this property to be *front node computable* in the sequel.

Corollary 2.5 (Computation of Φ-Score) *For all front node computable monotonous gain functions Φ, the score $\Phi(x)$ of a node x can be computed in $O(n \log n)$.*

2.2 Computing the Relaxed Solution

In this section we investigate the computation of the Φ-score of a tree and a corresponding node x where this score is attained. It is trivial that iterating the algorithm from the previous section over all nodes yields to an algorithm which needs $O(n)$ evaluations of this subroutine. We now propose a divide and conquer approach which takes only $O(\log n)$ evaluations of Φ-score.

The algorithm maintains a node $x_{\min}$ with currently minimal value $\Phi(x_{\min})$ and a subtree T_{active} which is guaranteed to contain all better nodes, if there exist any. In each iteration, the algorithm selects a pivot node x. Then it uses the subroutine from the previous section to determine a new smaller subtree with the above properties which hangs from a neighbor of x. We refer to this neighbor as a *guide node* which is formally defined as follows:

Definition 7 (Guide Node) Let x be an arbitrary node and $y \in N(x, \alpha)$ such that $\Phi(x) = \Phi(y \prec x)$. Then the neighbor of x on path $P(x, y)$ is called a *guide node* of x.

Notice that a guide node does not always need to exist. We remark further that the algorithm for computing the Φ-score of a node (described in the previous section) already yields a guide node as well with no additional time effort.

The crucial point of the divide and conquer approach, namely the division of the current subtree, is justified by the following result:

Theorem 2.6 (Divide Step) *Let x be an arbitrary candidate node. If there exists a guide node v of x, then all nodes x' with $\Phi(x') < \Phi(x)$ must lie in subtree $T_x(v)$. Otherwise, $\Phi(x) = \Phi^*$.*

Proof. At first we consider the case where x has at least one guide node v. Let y be an α-neighbor of x in subtree $T_x(v)$ with $\Phi(x) = \Phi(y \prec x)$. Further let x' be an arbitrary node in $T - T_x(v)$. We show that $\Phi(x') \geq \Phi(x)$: Since the path from x' to y meets x, it follows that $U(y \prec x) \subseteq U(y \prec x')$ and $U(x' \prec y) \subseteq U(x \prec y)$. Then we conclude

$$\Phi(x') \geq \Phi(y \prec x') \geq \Phi(y \prec x) = \Phi(x)$$

where we exploit the monotonicity property of Φ.

Secondly, assume that x does not have a guide node. Then $\Phi(x) = \Phi(x \prec x) =: \varphi_0$. By monotonicity it follows that for all nodes x' we have the same score $\Phi(x' \prec x') = \varphi_0$. Thus φ_0 is a lower bound on $\Phi(x')$ for all nodes x' and therefore $\Phi^* \geq \Phi(x)$. □

```
1  input tree T
2  initialize set of active nodes T_active ← T
3  initialize minimum score φ_min ← +∞
4  while T_active ≠ ∅
5      let x be an (unweighted) median node of T_active
6      compute Φ(x) and a guide node v, both with respect to T
7      if there is no guide node then output x, Φ(x) and stop
8      if Φ(x) < φ_min
9          remember φ_min ← Φ(x) and x_min ← x
10     set T_active ← T_active ∩ T_x(v)
11 output x_min, φ_min
```

Figure 2: Algorithm for computing the Φ-score of a tree.

The details of the algorithm which computes the Φ-score of a tree are depicted in Figure 2. In Line 6 of this algorithm, we compute the score $\Phi(x)$ by DFS as described in Section 2.1. This routine yields not only the score $\Phi(x)$ but also additionally returns a corresponding guide node or the information that x does not have a guide node. We remark that this subroutine runs always on the full input tree T regardless of the current restriction T_{active}.

Theorem 2.7 (Φ-Solution of a Tree) *A Φ-solution of a tree can be computed in time $O(n + \log n \cdot t(n))$ where $t(n)$ is the running time needed for evaluating the Φ-score and a guide of a node.*

Proof. The correctness follows by a repeated application of Theorem 2.6. Let $x_1, \ldots, x_k$ be the sequence of median nodes selected after k iterations of the algorithm, and $v_1, \ldots, v_k$ the corresponding guide nodes. Assume that x is a node with $\Phi(x) < \Phi(x_i)$ for all $i = 1, \ldots, k$. Then Theorem 2.6 implies that $x \in \bigcap_i T_{x_i}(v_i)$ which equals the set T_{active} maintained by our algorithm.

The claim on the running time follows from the observation that for any tree G a median node m divides the tree such that none of the subtrees in $G - m$ has more than

$|G|/2$ nodes. Hence we have $O(\log n)$ many iterations. In each iteration we evaluate one Φ-score and compute a median. Since the number of active nodes shrinks at least by one half per iteration, the total sum of computing the median nodes in all active trees is still $O(n)$. □

Corollary 2.8 *For all front node computable monotonous gain functions Φ, a Φ-solution of a tree can be computed in $O(n(\log n)^2)$.*

A more detailed analysis shows that the time $t(n)$ depends on the number $s(n)$ of nodes in any ball of radius α around nodes. Obviously, $s \in O(n)$ is always true. However, there are classes of trees where $s \in o(n)$ (e.g. if the parameter α is small compared to the average edge length, or the tree is degree bounded). In those cases, the running time can be better estimated to be in $O(n + \log n \cdot s(n) \cdot \log s(n))$.

3 Extension to Absolute Score and Solution

In this section we extend the algorithms provided previously to the case where candidate and opposition can be placed not only at nodes but also at inner points of edges of the graph. To this end, an edge can be considered as an infinite set of *points* [MF90]. A point x on edge $e = (u, v)$ is specified by the distance from one of the endpoints of e, and the remaining distance is derived from the invariant $d(u, x) + d(x, v) = c(e)$. Notice that the set of points of a graph includes the set of nodes.

If the domain of the monotonous gain function Φ includes the set of all pairs of points of a graph, we are enabled to formulate the problems investigated before under the new placement model:

Definition 8 (Absolute Φ-Score, Absolute Φ-Solution) The notions *absolute Φ-score* and *absolute Φ-solution* are defined similarly as in Definition 4, where x and y are now allowed to be points rather than nodes only.

Since the domain of the gain function is now an infinite set, we must first ensure that $\Phi(x)$ is well defined at all. As the value $\Phi(y \prec x)$ is fully determined by the user sets $U(y \prec x)$ and $U(x \prec y)$ and there is only a finite number of user subsets, also the image of Φ is a finite set. This directly implies that the absolute Φ-score of a point is always well defined. We will later show that the tree decomposes into a finite family of disjoint intervals and points such that the absolute Φ-score is constant at each member of this family.

Definition 9 (α-Neighborhood) Let x be a candidate point. An α-neighbor of x is a point y where $d(x, y)$ is infinitesimally greater than α.

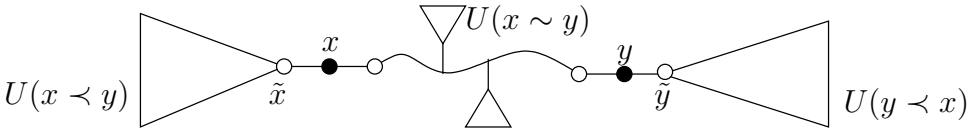
In the appendix we provide a more formal definition of such "infinitesimally close" points, together with the proof of the following lemma.

Lemma 3.1 *Let x be a candidate point and y be an α-neighbor. Then all inner nodes z on path $P(x, y)$ are undecided.*

Theorem 3.2 (Characterization of Front Nodes) *Let x be a candidate point and y be an α-neighbor of x. Let $\tilde{x}$ be the node nearest to x such that $P(\tilde{x}, x)$ does not meet inner points of $P(x, y)$; similarly define $\tilde{y}$. Then*

$$U(y \prec x) = T_{\tilde{x}}(\tilde{y}) \qquad \text{and} \qquad U(x \prec y) = T_{\tilde{y}}(\tilde{x})\,.$$

The nodes $\tilde{x}$ and $\tilde{y}$ are referred to as front nodes *of x and y, respectively.*



Proof. Consider an arbitrary node z and its projection $\bar{z}$ onto $P(\tilde{x}, \tilde{y})$. If $\bar{z}$ is an inner node on $P(\tilde{x}, \tilde{y})$, by Lemma 3.1 it is undecided. Otherwise, z is part of one of the subtrees $T_{\tilde{x}}(\tilde{y})$ or $T_{\tilde{y}}(\tilde{x})$ and clearly prefers x or y, respectively. □

Theorem 3.3 (Enumeration of Front Node Pairs) *For a given node x, the set of all pairs $(\tilde{x}, \tilde{y})$ where y is an α-neighbor and $\tilde{x}, \tilde{y}$ are the corresponding front nodes can be computed in time $O(n)$.* □

Corollary 3.4 (Computation of Absolute Φ-Score) *For all front node computable monotonous gain functions Φ, the absolute score $\Phi(x)$ of a node x can be computed in time $O(n)$.*

In the sequel we develop an algorithm to compute the absolute Φ-score of a tree, i.e., a node x with minimum value $\Phi(x)$. The algorithm consists of two phases. The first phase is merely a modification of the divide and conquer algorithm suggested in the previous section, and tests candidate points x at nodes of the tree only. The second phase considers the situation when an optimal solution is actually an inner point of an edge.

Definition 10 (Guide Edge) Let x be an arbitrary node and $y \in N(x, \alpha)$ such that $\Phi(x) = \Phi(y \prec x)$. Then the edge incident with x and sharing inner points with the path $P(x, y)$ is called a *guide edge* of x.

We now look at the divide step of the algorithm. The difference to the discrete case presented in Section 2.2 is that now after dividing the tree at node x we cannot exclude the points of the guide edge from further consideration. The rest of the proof of the following claim is along the lines drawn out in the proof of Theorem 2.6.

Theorem 3.5 (Divide Step) *Let x be an arbitrary node. If there exists a guide edge (x, v), then all points x' with $\Phi(x') < \Phi(x)$ must lie in the subtree $T_x(v) \cup \{(x, v)\}$.* □

At this point we can invoke a slightly modified version of the algorithm depicted in Figure 2 as the first phase of our algorithm. The obvious changes are replacement of

guide node by guide edge and the change to $T_{\text{active}} \leftarrow T_{\text{active}} \cap \big(T_x(v) \cup \{(x,v)\}\big)$ in Line 10. While in the original version it is true that the tree properly shrinks by at least one node in each division step, this holds no longer in the new setting when we arrive at two nodes connected by an edge. So in order to avoid endless loops, we adapt the termination rules: if the active set consists of exactly two nodes connected by one edge, then the phase terminates and enters phase two described in the sequel. (The case that the active set consists only of one single node is, unless the input tree consists only of one node, actually impossible, as is shown in Appendix 5.2.)

In Phase 2 of our algorithm we have to determine an optimal point when the active set has reduced to a single edge already. We reduce this case to a computation of an optimal point in a modified tree. To this end, we identify the set of *critical* points on the remaining edge $e = (u, v)$, i.e., those points where the value of Φ can change. Clearly, this set of critical points includes points with a distance of exactly α to a node of T. Let

$$X := \{\, \text{point } x \in e \mid d(x,z) = \alpha \text{ for some } z \in V \,\} \cup \{u, v\}$$

be a set of points on e. We sort point set X such that for $X = \{x_0, \ldots, x_k\}$, $x_0 = u$, $x_k = v$, and $d(u, x_i) < d(u, x_{i+1})$ for all i. Notice that $|X| \leq n$.

We claim that there are no more critical points on the edge e; in detail, the value of Φ is constant on each of the open intervals (x_i, x_{i+1}). This follows from applying Theorem 3.2. For any two inner points x', x'' of the same interval, the sets of edges which α-neighbors of x' and x'' lie on are identical. Moreover, also the sets of corresponding front nodes coincide. Hence the user preferences and finally the Φ-scores must be the same.

We restart the algorithm of the first phase on the tree with node set X added, and with initial active set $T_{\text{active}} := X$. This phase will terminate again with a subedge (x_i, x_{i+1}) of e. At this point we choose an arbitrary inner point x of this subedge and evaluate the absolute scores $\Phi(x_i), \Phi(x), \Phi(x_{i+1})$ and compare these with the best result so far.

Theorem 3.6 (Absolute Φ-Solution of a Tree) *An absolute Φ-solution of a tree can be computed in time $O(\log n \cdot (t(n) + n))$ where $t(n)$ is the time needed for evaluating the Φ-score and a guide of a node.* □

Corollary 3.7 *For all front node computable monotonous gain functions Φ, an absolute Φ-solution of a tree can be computed in $O(n \log n)$.*

4 Applications

4.1 Relaxed Simpson Solution

If we set $\Phi(y \prec x) := \Gamma(y \prec x)$ we describe the Simpson problem. Obviously Φ satisfies the conditions of a monotonous gain function in this case. Since $\Gamma(y \prec x) = w(T_{\tilde{x}}(\tilde{y}))$ this function is also front node computable which allows to compute a Simpson solution in time $O(n(\log n)^2)$ according to Corollary 2.8.

However, since the value $\Gamma(y \prec x)$ does not depend on $U(x \prec y)$ we need no knowledge about the front node $\tilde{x}$. Hence we do not need the binary search described in the proof

of Theorem 2.4 and end up with an algorithm running in time $O(n \log n)$. (This matches the result from [NSW07] which was derived using a different approach.)

Corollary 4.1 *The relaxed Simpson score and the relaxed absolute Simpson score of a tree can be computed in time $O(n \log n)$.*

4.2 Relaxed Security Solution

We now consider the gain function $\Phi(y \prec x) := \Delta(y \prec x)$ to describe the security problem. It is easy to observe that this is a monotonous gain function and also front node computable.

Corollary 4.2 *The relaxed security score of a tree can be computed in time $O(n \cdot (\log n)^2)$ and the relaxed absolute security score in time $O(n \log n)$.*

We briefly remark that this result can be applied also to variations of the standard security score, e.g. when the difference is replaced by the ratio where we end up with a gain function $\Phi(y \prec x) = W(y \prec x)/W(x \prec y)$.

4.3 Stackelberg Solution

In the sequel we leave the area of voting problems and turn over to the family of *competitive location* problems [ELT93]. Here two providers, called *leader* and *follower*, sequentially locate facilities in a graph in order to maximize their market share. In the case of *Stackelberg* solutions, users prefer nearest facilities. If a user is undecided, its demand is split equally amongst the two competing providers [HTW90]. Once the leader has settled down on position x the follower chooses a location y which maximizes its individual gain. In the framework of this paper the market share of the follower can be modeled by the monotonous gain function $\Phi(y \prec x) = W(y \prec x) + \frac{1}{2}W(y \sim x)$, and finding an optimal position for the follower is equivalent to computing the Φ-score of the leader. Hence the behavior of the follower is fully foreseeable, and if the leader tries to maximize his gain, this is equivalent to the problem of determining the Φ-score of the graph.

We generalize this problem by introducing a function $f\colon V \to [0,1]$ which specifies the individual user demand gained by the follower in the case where the user is undecided. (The above problem is then the special case of $f \equiv \frac{1}{2}$.) For simplicity we use the notation

$$F(U) := (f \cdot w)(U) = \sum_{u \in U} f(u) \cdot w(u) \qquad \text{for all user sets } U$$

and write $F(y \sim x) := F(U(y \sim x))$. Thus we end up with a gain function

$$\Phi(y \prec x) := W(y \prec x) + F(y \sim x)$$

in the above framework. Clearly, this function is a monotonous gain function.

To convince ourselves that Φ is front node computable, we enhance the preprocessing phase described in Section 2.1 to compute also the values $F(T_u(v))$ and $F(T_v(u))$ for

each edge $(u, v) \in E$. Since the value $F(y \sim x)$ can be computed with the help of the identity

$$F(y \sim x) = F(T) - F(y \prec x) - F(x \prec y) = F(T) - F(T_{\tilde{y}}(\tilde{x})) - F(T_{\tilde{x}}(\tilde{y}))$$

in constant time, the claim follows.

Corollary 4.3 *The relaxed Stackelberg solution on a tree can be computed in time $O(n(\log n)^2)$ and the relaxed absolute Stackelberg solution in time $O(n \log n)$.*

We note that in the case of $f \equiv 0$ and $\alpha = 0$ we arrive at the $(1|1)$-centroid problem which is equivalent to the Simpson solution described earlier [Hak90].

4.4 Verification of Nash Solution

A pair (x, y) of nodes in a graph is called a *Nash solution* if no party can increase its payoff by moving to another location [HTW90]. The payoff is given as in the previous section as $\Phi(y \prec x) = W(y \prec x) + \frac{1}{2}W(y \sim x)$. Formally, the pair must satisfy

$$\Phi(y \prec x) \geq \Phi(y' \prec x) \qquad \text{and} \qquad \Phi(x \prec y) \geq \Phi(x' \prec y) \qquad \text{for all } x', y' .$$

Obviously these conditions are equivalent to $\Phi(x) = \Phi(y \prec x)$ and $\Phi(y) = \Phi(x \prec y)$. So the test can be performed in time $O(n \log n)$ as shown before.

Corollary 4.4 *On a tree, the verification whether a pair of nodes (a pair of points) is a Nash solution (an absolute Nash solution) can be performed in time $O(n \log n)$ (in time $O(n)$).*

References

[CM02] C. M. Campos Rodríguez and J. A. Moreno Peréz · *Multiple voting location problems* · unpublished manuscript, 2002.

[CM03] C. M. Campos Rodríguez and J. A. Moreno Peréz · *Relaxation of the condorcet and simpson conditions in voting location* · European Journal of Operations Research **145** (2003), 673–683.

[ELT93] H. A. Eiselt, G. Laporte, and J.-F. Thisse · *Competitive location models: A framework and bibliography* · Transportation Science **27** (1993), no. 1, 44–54.

[Hak90] S. L. Hakimi · *Locations with spatial interactions: Competitive locations and games* · in [MF90], 1990, pp. 439–478.

[HTW90] P. Hansen, J.-F. Thisse, and R. E. Wendell · *Equilibrium analysis for voting and competitive location problems* · in [MF90], 1990, pp. 479–501.

[MF90] P. B. Mirchandani and R. L. Francis · *Discrete location theory* · Series in Discrete Mathematics and Optimization, Wiley-Interscience, 1990.

[NSW07] H. Noltemeier, J. Spoerhase, and H.-C. Wirth · *Multiple voting location and single voting location on trees* · to appear in European Journal of Operations Research (EJOR), 2007.

5 Appendix

5.1 Justification of Infinitesimality in Definition 9

In Definition 9 we define an α-neighbor of candidate point x to be a point y where $d(x, y)$ is "infinitesimaly greater" than α. We illustrate now the idea behind this definition. Moreover we show also that an α-neighbor is ready for an implementation and can in particular be computed efficiently.

Let y' be a point such that $d(x, y') = \alpha$. We distinguish two cases:

1. y' is a node of the tree. A user at node y' is undecided. Let $z_1, \ldots, z_k$ be those node neighbors of y' where the edge (y', z_i) does not have any inner point in common with $P(x, y')$.

   While choosing $y := z_i$ for some i guarantees that $U(y \prec x) = T_x(y)$, this leaves the chance that there exists a node $\tilde{x}$ which is an inner node of $P(x, y')$ but has distance $d(y, \tilde{x}) > \alpha$. This node $\tilde{x}$ would prefer x and hence the front node of x were an inner node of path $P(x, y)$.

   To avoid this we compute $d := d(x, x')$ where x' is the neighbor of x in direction towards y'. Then for some i, we choose y on edge (y', z_i) so that $d(y', y) < d$.

2. y' is not a node. Let z be the node neighbor of y' such that subedge (y', z) has no inner points in common with path $P(x, y')$. Similar to above, select y on subedge (y', z) such that $d(y', y) < d$.

Defining an α-neighbor this way, we can now present the missing proof from page 9.

Proof (of Lemma 3.1). Let x' be the neighbor of x and $d = d(x, x')$ as described above. Further, let $d' := d(y', y) < d$ be the distance by which y exceeds the α radius. Consider a node z being an inner node on path $P(x, y)$. We claim that $|d(x, z) - d(z, y)| \leq \alpha$. To this end,

$$d(x, z) - d(z, y) = d - d' + d(x', z) - d(z, y') > 0 - d(z, y') \geq -\alpha$$

and on the other hand

$$d(x, z) - d(z, y) = d + d(x', z) - d(z, y') - d' \leq d + d(x', z) \leq \alpha\,.$$

This shows the claim. □

5.2 Termination of Algorithm from Section 3

We show now that the first phase of the algorithm described in Section 3 does not need to deal with the case that $|T_{\text{active}}| = 1$ because this cannot happen as a result of the divide step. (Observe first that if the initial tree consists of one node only, then the algorithm terminates already correctly since there is no guide.)

To create a single node set $T_{\text{active}} = \{x\}$, it is necessary that the node x is selected in two different divide steps as a pivot element, and the corresponding guide edges in these steps being different.

Observe that after the first of these divide steps, node x is a leaf in the tree induced by T_{active} at that time. Since the algorithm selects always a median node of T_{active} as pivot element, the node x can only be selected for the second time when a leaf can be a median node.

This is impossible for trees containing at least three nodes. A tree with two nodes, i.e., a single edge, would leave phase 1 and enter phase 2. A tree with one node is impossible by induction. This concludes the claim.