Efficient Truthful Mechanisms for the Single-Source Shortest Paths Tree Problem[⋆]

Luciano Gualà[1] and Guido Proietti[1,2]

[1] Dipartimento di Informatica, Università di L'Aquila, Italy
{guala,proietti}@di.univaq.it
[2] Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy

Abstract. Let a communication network be modelled by an undirected graph $G = (V, E)$ of n nodes and m edges, and assume that each edge is controlled by a selfish agent. In this paper we analyze the problem of designing a truthful mechanism for computing one of the most used structures in communication networks, i.e., the *single-source shortest paths tree*. More precisely, we will show that under various realistic agents' behavior scenarios, it can be guaranteed not only the existence, but also the efficiency (in terms of running time complexity) of such mechanisms. In particular, for the fundamental case in which the problem is *utilitarian*, we will show that a truthful mechanism can be computed in $O(mn \log \alpha(m, n))$ time, where $\alpha(m, n)$ is the classic inverse of the Ackermann's function.

Keywords: Equilibria in Distributed Systems, Single-Source Shortest Paths Tree, Selfish Agents, Algorithmic Mechanism Design, Truthful Mechanisms.

1 Introduction

Mechanisms are a classical concept of the theory of non-cooperative games [16]. In these games there are several independent agents that have to work together in order to optimize a global objective function. However, each agent has her own valuation function and may lie in hope of getting a higher profit. This leads to economically suboptimal resource allocation and is therefore undesirable. The main objective of *mechanism design* theory is to study how to incentive the agents in order to cooperate with the solving algorithm. A *mechanism* is a pair $\mathcal{M} = \langle g(\cdot), p(\cdot) \rangle$, where $g(\cdot)$ is an algorithm computing a solution, and $p(\cdot)$ specifies the payments provided to the agents. Informally, a mechanism is *truthful* if its payments guarantee that agents are not stimulated to lie. Then, the problem of combining the game theoretic concept of designing a truthful mechanism, with the computational complexity requirement of designing an efficient algorithm, is exactly the topic of the *algorithmic mechanism design* (AMD) for selfish agents.

[⋆] Work partially supported by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research.

In their seminal paper concerned with AMD [15], Nisan and Ronen addressed the classic *shortest path* problem. This problem enjoys the property of being *utilitarian*. For utilitarian problems, there exists a well-known class of truthful mechanisms, i.e., the *Vickrey-Clarke-Groves (VCG) mechanisms* [4, 6, 21], and therefore the shortest path problem can be solved optimally. Afterwards, in a sequel of papers, efficient truthful mechanism have been designed for solving several other network design problems [8, 9, 12, 14, 15, 19].

In this paper, we focus on one of the most popular network topologies, that is the *single-source shortest paths tree* (SPT). What is interesting here is that an SPT naturally admits both utilitarian and non-utilitarian formulations. Indeed, as we will discuss later in the paper, it can well happen that an agent gives an evaluation of her contribution which is simply proportional to her private type, and this unavoidably makes the problem non-utilitarian. Therefore, we analyze both the scenarios, and we provide the following main results:

– In the utilitarian case, we provide a VCG-mechanism which can be implemented in $O(mn \log \alpha(m,n))$ time on a RAM, and in $O(mn\, \alpha(m,n))$ time on a pointer machine, where $\alpha(m,n)$ is the inverse of the Ackermann's function defined in [20];
– In the non-utilitarian case, we provide: (i) an n-approximate VCG-mechanism which can be implemented in almost optimal $O(m\, \alpha(m,n))$ time, and (ii) a mechanism guaranteeing both truthfulness and positive utilities for the agents which can be implemented in $O(m + n \log n)$ time.

The paper is organized as follows: In Section 2 we recall some basic definitions from both graph theory and algorithmic mechanism design; in Section 3 we deal with the utilitarian version of our problem, while in Section 4 we analyze the different solutions for the non-utilitarian case.

2 Basic Definitions

Let $G = (V, E)$ be an undirected graph, with $|V| = n$ nodes and $|E| = m$ edges, and with a positive real weight associated with each edge $e \in E$. Given a source node s and a destination node z in G, a path in G between s and z is a *shortest path*, say $P_G(s, z)$, if the sum of its edge weights (called *distance* in G between s and z, and denoted by $d_G(s, z)$) is minimum. Given a source node $s \in V$, we denote by $S_G(s)$ a *single-source shortest paths tree* (SPT) of G rooted in s, i.e, the union of all the shortest paths from s to every $v \in V \setminus \{s\}$. Given $u, v \in V$, we denote by $\mathrm{LCA}(u, v)$ the *least common ancestor* of u and v in $S_G(s)$, i.e, the ancestor of both u and v in $S_G(s)$ which is farthest from s.

Let $e = (u, v) \in S_G(s)$ be a tree edge, with u closer to s than v. Let $M(e)$ denote the set of nodes in $S_G(s)$ reachable from s without passing through edge e, and let $N(e) = V \backslash M(e)$ be the remaining nodes. Sets $M(e)$ and $N(e)$ define a cut in G, and $C(e) = \{(x, y) \in E \backslash \{e\} \mid (x \in M(e)) \wedge (y \in N(e))\}$ is the set of the edges crossing the cut. Moreover, we denote by $||e||$ the cardinality of $N(e)$.

Let a communication network be modelled by a 2-edge-connected graph G, and assume that each edge is owned by a selfish agent A_e, which holds a private information t_e. We call this value *input type* of the agent A_e. This value depends on various factors (e.g., bandwidth, reliability, etc.). Only agent A_e knows t_e, while everything else is public knowledge. Each agent has to declare a public *reported type* r_e to the mechanism. We will denote by t the vector of input types, and by r the vector of bids.

For a given optimization problem defined on G, there exists some set of feasible solutions F that the mechanism is allowed to choose. For each feasible solution $x \in F$, some measure function $\mu(x,t)$ is defined, which depends on the true types. The mechanism tries to optimize $\mu(x,t)$, but of course it does not know t directly.

For every agent A_e, a function $v_e(t_e,x)$ expresses A_e's *valuation* with respect to an output $x \in F$: this is a quantification of the service carried on by A_e into x. While t_e is known only by the agent A_e, the valuation function is public. In order to offset the costs deriving from these services, the mechanism provides some reward to agents participating to the computed solution, i.e., the mechanism makes a *payment* $p_e(r)$ to the agent A_e for the service provided in a solution which is computed as a function of the reported vector r.

A mechanism is a pair $\mathcal{M} = \langle g(r), p(r)\rangle$, where $g(r)$ is an algorithm that, given agents' bids, computes a feasible solution in F, and $p(b)$ is a scheme which describes the payments provided to the agents.

For each agent A_e and for each solution $g(r)$ computed by the mechanism, the *utility* function of A_e is defined as $u_e(t_e,r) = p_e(r) - v_e(t_e, g(r))$. We assume that each agent is selfish, i.e., she always attempts to maximize her utility. Let r_{-e} denote the vector of all bids besides r_e; the pair (r_{-e}, r_e) will denote the vector r. We say that *truth-telling* is a *dominant strategy* for agent A_e if bidding t_e always maximizes her utility, regardless of what the other agents bid, i.e., $u_e(t_e,(r_{-e},t_e)) \geq u_e(t_e,(r_{-e},r_e))$, for all r_{-e} and r_e. A mechanism is said *truthful* if, for every agent, truth-telling is a dominant strategy. Moreover, let $\varepsilon(\sigma)$ denote a positive real function of the input size σ. Then, an $\varepsilon(\sigma)$-*approximation mechanism* is a mechanism which returns a solution $g(r)$ which comes within a factor $\varepsilon(\sigma)$ from the optimum, i.e., $\mu(g(r),t) \leq \varepsilon(\sigma)\cdot\mu(x^*,t)$, where x^* is an optimal solution with respect to the vector t. We say that a mechanism is *poly-time computable* if $g(\cdot)$ and $p(\cdot)$ are computable in polynomial time and that satisfies the *voluntary participation* condition if agents never incur in a net loss.

One of the most important results in mechanism design theory are the well-know *Vickrey-Clarke-Groves* (VCG) mechanisms. A VCG-mechanism applies to mechanism design problems called *utilitarians* and enjoys the fundamental property of being truthful. A mechanism design problem is called *utilitarian* if its measure function satisfies $\mu(x,t) = \sum_{e\in E} v_e(t_e,x)$.

Definition 1 (VCG-mechanisms). *A mechanism is of the VCG-family if:*

1. $g(r) \in \arg\min_{x\in F}\left\{\sum_{e\in E} v_e(r_e,x)\right\}$.
2. $p_e(r) = \sum_{e'\neq e} v_{e'}(r_{e'}, g(r)) + h_e(r_{-e})$, *where* $h_e(r_{-e})$ *is an arbitrary function independent of* r_e.

3 The Utilitarian Case

Let be given a communication network modelled by an undirected graph $G = (V, E)$ in which each edge $e \in E$ is owned by a selfish agent. In the following, we will denote by G and $\widetilde{G}$ the input graph as weighted with respect to the reported values and the input types, respectively.

Suppose that A_e holds, as the private type t_e for the owned edge e, the length of the communication link, and thus the time needed to cross it, and assume that the system-wide goal is to minimize the completion time for delivering a message from a distinguished node $s \in V$ to every node $v \in V \setminus \{s\}$. This means that the system looks for an SPT rooted in s of $\widetilde{G}$.

3.1 A (Truthful) VCG-Mechanism

By using the notation introduced in the previous section, the problem can be formalized as follows. The set of feasible solutions F is the set of all the spanning trees (considered in the following as rooted in s) of $\widetilde{G}$, and a measure of a solution $T \in F$ is

$$\mu(T, t) = \sum_{v \in V} d_T(s, v). \tag{1}$$

To complete the description of the problem, we have to define the agents' valuation. It is clear that, if an agent A_e participates to the output with her edge e, she will incur in a transmission cost (i.e., the cost for forwarding a message through that edge). In our scenario it is reasonable to assume that the transmission cost is proportional to the length of the edge, i.e., proportional to the value t_e. Notice that the TCP/IP protocol used in Internet for broadcasting a message is the so-called *unicast*. In this protocol, if a source wants to send a message to a set of recipients, it must send a copy of the message for each destination. Therefore, if any solution $T \in F$ is used for broadcasting a message from s to all the other nodes, then the cost for the agent A_e can be expressed as follows:

$$v_e(t_e, T) = \begin{cases} t_e ||e|| & \text{if } e \in E(T); \\ 0 & \text{otherwise.} \end{cases}$$

Indeed, if an agent A_e participates to the output T, she will incur a transmission cost of t_e for each message which passes through e, and the number of these messages is exactly $||e||$.

From the above assumptions, it immediately follows that the problem is utilitarian. Indeed, the measure function (1) can be rewritten as

$$\mu(T, t) = \sum_{v \in V} d_T(s, v) = \sum_{e \in E(T)} t_e ||e|| = \sum_{e \in E} v_e(t_e, T).$$

This means that we can use a VCG-mechanism to solve the problem. Therefore, let $\mathcal{M}_1$ be a mechanism defined as follows:

1. The algorithmic output specification selects an SPT $S_G(s)$ of G;
2. Let $G - e = (V, E\backslash\{e\})$. Then, the payment function for A_e is defined as

$$p_e(r) = \begin{cases} \sum_{v \in V} d_{G-e}(s,v) - \big(\mu(S_G(s), r) - r_e\|e\|\big) & \text{if } e \in E(S_G(s)); \\ 0 & \text{otherwise.} \end{cases}$$

It is clear that the above payments obey to Definition 1 and then the mechanism is a (truthful) VCG-mechanism. Furthermore, this is a payment scheme inducing a so-called *pivotal mechanism*, which can be shown to satisfy the voluntary participation [4].

3.2 Mechanism Time Complexity

The algorithmic question is now the following: how fast can the above mechanism be computed? We start analyzing the cost for computing the payment scheme.

To compute $p_e(r)$ for each $e \in E(S_G(s))$, the bottle-neck is to find all the distances $d_{G-e}(s,v)$, for every $v \in V$. Indeed, it is not hard to see that the term $\mu(r, S_G(s)) - r_e\|e\|$ can be found, for all the edges $e \in E(S_G(s))$, in $O(n)$ time. A trivial solution consists in computing a new SPT of the graph $G - e$ from scratch, once for each edge $e \in E(S_G(s))$. This solution clearly takes $O(mn + n^2 \log n)$ time. We now show how to improve (on a RAM) the above bound to $O(mn \log \alpha(m,n))$ time.

We start by computing, for all the pairs $u, v \in V$, the distance $d_G(u,v)$. This can be done in $O(mn \log \alpha(m,n))$ time [17]. Then, we solve $n-1$ subproblems. Each subproblem is identified by a distinct destination node $z \in V$, and asks for computing the distance $d_{G-e}(s,z)$ for each edge e of the path in $S_G(s)$ between s and z. We have to solve exactly $n-1$ subproblems, one for each $z \in V\backslash\{s\}$, since the distance from s to z may increase – as a consequence of deleting the edge e – only if e belongs to $P_G(s,z)$. We will solve each subproblem in $O(m \log \alpha(m,n))$ time, by achieving a bound of $O(mn \log \alpha(m,n))$ time for the original problem.

Let $P_{G-e}(s,z)$ be a replacement shortest path for the edge e, i.e., a path from s to z in $G - e$ of (minimum) length $d_{G-e}(s,z)$. The problem of finding all the replacement shortest paths, one for each edge of $P_G(s,z)$, has been efficiently solved in $O(m + n \log n)$ time on a pointer machine [10], and $O(m\,\alpha(m,n))$ time on a word RAM [11], respectively. Both algorithms are based on a precomputation of the SPTs $S_G(s)$ and $S_G(z)$. We now show how to improve the above results to $O(m \log \alpha(m,n))$ time, by using a powerful structure called *Split-Findmin* [17].

Let $e = (u,v)$ be an edge on $P_G(s,z)$, with u closer to s than v. Since a replacement shortest path $P_{G-e}(s,z)$ joining s and z must contain an edge in $C(e)$, it follows that it corresponds to a path of length

$$d_{G-e}(s,z) = \min_{f=(x,y) \in C(e)} \big\{k(f) := d_{G-e}(s,x) + r_f + d_{G-e}(y,z)\big\},$$

which can be shown [10] to be equivalent to

$$d_{G-e}(s,z) = \min_{f \in C(e)} \big\{d_{S_G(s)}(s,x) + r_f + d_{S_G(z)}(y,z)\big\} = \min_{f \in C(e)} \big\{d_G(s,x) + r_f + d_G(y,z)\big\}. \quad (2)$$

Hence, since we have pre-computed the all-pairs distances in G, $k(f)$ is available in $O(1)$ time for fixed f. It then remains to select the minimum over $C(e)$. To do this efficiently, we use a Split-Findmin structure. This is a structure operating on a collection of disjoint sequences of n elements. Initially, there is only one sequence containing all the elements, and each element u has a key $k(u) := +\infty$. Then, the structure supports the following operations:

`split`(u): Split the sequence containing u into two sequences of elements: one up to and including u, the other sequence taking the rest;

`findmin`(u): Return the element (and the associated key) in u's sequence with minimum key;

`decrease-key`(u, k'): Set $k(u) := \min\{k(u), k'\}$.

We find all the distance $d_{G-e}(s, z)$ as follows. First, we label each non-tree edge f with the value (2). Then, we initialize a Split-Findmin structure, where the initial n-elements sequence consists of the vertices of $S_G(s)$ as sorted in any arbitrary post-order. We maintain two invariants: (1) every sequence in the Split-Findmin structure corresponds to some rooted subtree of $S_G(s)$, and (2) $k(u)$ corresponds to the label of a min-label edge connecting u to a vertex outside u's sequence (i.e., outside the subtree of $S_G(s)$ currently containing u).

Let now $e = (u, v) \in P_G(s, z)$. By invariants (1) and (2), if S is a sequence in the Split-Findmin structure and v is the root of the subtree corresponding to S, then `findmin`(v) will return a key $k(f_e)$, where f_e is a non-tree edge belonging to $P_{G-e}(s, z)$ and $k(f_e)$ is exactly the distance $d_{G-e}(s, z)$. Once f_e is determined, we proceed to solve the problem for the children of v along the path $P_G(s, z)$. Because of the post-order arrangement of the nodes, v is the rightmost element in its sequence. Then, we perform one split centered at the element preceding v in the sequence (this will sever v), and one additional split (in any arbitrary order) for each of the children of v in $S_G(s)$, to reestablish invariant (1). After, we focus on the sequence associated with the children of v in $P_G(s, z)$, say w, and we restore invariant (2) by performing a number of decrease-key operations. More precisely, for each edge $f = (w', y)$ such that $\mathrm{LCA}(w', y) = v$ and w' is a descendant of w in $S_G(s)$, we issue the operation `decrease-key`$(w', k(f))$ (see Figure 1). Concerning the time complexity, the following lemma holds:

Lemma 1. *Let $P_G(s, z)$ be a shortest path joining s and z. Given all the distances $d_G(s, x)$ and $d_G(z, x)$ for each $x \in V$, all the distances $d_{G-e}(s, z)$ for each $e \in P_G(s, z)$ can be determined in $O(m \log \alpha(m, n))$ time.*

Proof. Since $k(f)$ is available in $O(1)$ time for a fixed non-tree edge f, labelling all the non-tree edges takes $O(m)$ time. Concerning the Split-Findmin operations, in total there are $O(m)$ operations: $O(n)$ splits (one for each subtree whose root is adjacent to some node of $P_G(s, z)$), $O(n)$ findmins (one for each node of $P_G(s, z)$), and $O(m)$ decrease-keys (at most one for each non-tree edge). This takes $O(m \log \alpha(m, n))$ time [17]. Other costs, such as the post-order traversal and finding least common ancestors, are linear [2]. □

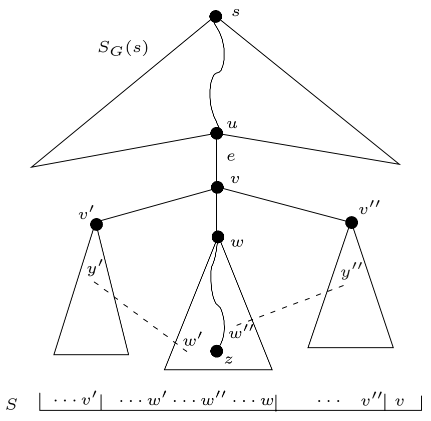

Fig. 1. The sequence S corresponding to the subtree of $S_G(s)$ rooted at v is split after the $\texttt{findmin}(v)$ operation. Dashed edges are those for which a decrease-key operation is performed.

We are now ready to prove the main result:

Theorem 1. *The mechanism $\mathcal{M}_1$ is a truthful mechanism for the utilitarian SPT problem, and can be computed on a RAM in $O(mn \log \alpha(m,n))$ time.*

Proof. The mechanism belongs to the VCG-family, and therefore it is truthful. Concerning the output specification, the fastest solution for computing an SPT is the classic Dijkstra's algorithm implemented with Fibonacci heaps, which runs in $O(m + n \log n)$ time [5]. On the other hand, as far as the payment scheme is concerned, we proceed as follows. First, we find the all-pairs distances in G in $O(mn \log \alpha(m,n))$ time [17], and we solve each of the above described subproblems in $O(m \log \alpha(m,n))$ time. Then, for each edge $e = (u,v) \in E(S_G(s))$, we extract from the solutions of the subproblems all the distances $d_{G-e}(s,x)$, for every x in the subtree of $S_G(s)$ rooted at v (all the other nodes clearly maintain their distance from s in $G - e$). Thus, we can easily compute $p_e(r)$ in $O(n)$ time, since $\mu(r, S_G(s))$ and $r_e||e||$ can be obtained in $O(n)$ time by a trivial modified post-order visit of $S_G(s)$. Since we have to compute exactly $n-1$ payment functions, one for each tree edge, the claim follows. $\square$

Notice that on a pure pointer machine model, the mechanism $\mathcal{M}_1$ can be computed in $O(mn\,\alpha(m,n))$ time, since in this case the Split-Findmin data structure requires $O(m\,\alpha(m,n))$ time for solving any given subproblem [17].

4 The Non-utilitarian Case

The utilitarian scenario assumes that each agent, in doing her valuation, starts from the assumption that each atomic operation will involve a traffic load on the owned edge which is proportional to the edge length times the size of the corresponding appended subtree of $S_G(s)$. However, in another reasonable scenario, an agent might evaluate her participation to an output $T \in F$ simply as follows:

$$v_e(t_e, T) = \begin{cases} t_e & \text{if } e \in E(T); \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

This scenario is realistic whenever the agent starts from the assumption that each atomic operation will involve a traffic load on the owned edge which is proportional only to the edge length (this can happen, for instance, when the transmission protocol replicates at each node a given message once for each descending node, like in the *Internet Protocol multicast* [3], so that each tree edge will simply afford the cost of forwarding a single message).

This setting makes the problem non-utilitarian, since the measure function associated with the SPT problem does not equal the sum of the agents' valuations. In the following, we show how to approach the problem from two different perspectives. In both cases, we make use of the pointer machine computational model, since we cannot take advantage of the addressing capabilities of a RAM, as we did for the utilitarian case.

4.1 An Approximate VCG-Mechanism

A brute-force solution consists in designing a mechanism from the VCG-family. Since the algorithmic output specification has to minimize the sum of the agents' valuations, this will clearly return an MST of $\widetilde{G}$. More formally, let $\mathcal{M}_2$ be the mechanism defined as follows:

1. The algorithmic output specification computes an MST of G;
2. Let $w(T_G) = \sum_{e \in E(T_G)} r_e$ denote the total weight of the solution T_G, as computed in G. Then, the payment function for A_e is defined as

$$p_e(r) = \begin{cases} w(T_{G-e}) - \big(w(T_G) - r_e\big) & \text{if } e \in E(T_G); \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2. *The mechanism $\mathcal{M}_2$ is a truthful n-approximation mechanism for the non-utilitarian SPT problem, and is computable on a pointer machine in $O(t^*(T_G)) = O(m\,\alpha(m,n))$ time, where $t^*(T_G)$ is the time needed to solve optimally the MST problem.*

Proof. It is easy to see that $g(\cdot)$ minimizes the sum of the agents' valuations and that the mechanism belongs to the VCG-family. Indeed, the above payments obey to Definition 1 and then the mechanism is truthful (and consequently T_G is an MST of $\widetilde{G}$). Concerning the approximation ratio, let $S_{\widetilde{G}}(s)$ be an optimal solution for the SPT problem. We now show that the solution returned by the VCG-mechanism is a factor-n approximation. Indeed, if we consider $T_{\widetilde{G}}$ as rooted in s, we have

$$\mu\left(t, T_{\widetilde{G}}\right) = \sum_{e \in E\left(T_{\widetilde{G}}\right)} t_e \|e\| \leq n \sum_{e \in E\left(T_{\widetilde{G}}\right)} t_e = n\, w\left(T_{\widetilde{G}}\right) \leq n\, w\left(S_{\widetilde{G}}(s)\right) \leq n\, \mu\left(t, S_{\widetilde{G}}(s)\right).$$

Concerning the time complexity, observe that the mechanism essentially requires the computation on a pointer machine of an MST of G (which can be done optimally through the algorithm presented in [18], which has an $O(m\,\alpha(m,n))$ runtime, but for which a tighter analysis is not known), and the solution of a sensitivity analysis problem on T_G. As showed in [7], such a problem can be solved in optimal time as well, but still a tight analysis cannot be provided, thought it is known that such a problem is not harder than the MST one. Summarizing, the time complexity is $O(t^*(T_G)) = O(m\,\alpha(m,n))$. $\square$

Notice that the above approximation ratio is tight, since it is easy to exhibit an example in which an MST is an n-approximation of an SPT. This means that we cannot hope to get a better approximate result by means of VCG-mechanisms.

4.2 An Exact Truthful Mechanism Satisfying the Voluntary Participation

The alternative solution we propose is inspired to the results in [1], where the authors show how to design truthful mechanisms for those problems in which each agent's valuation has the form $v_e(t_e, x) = t_e\, w_e(r)$, where $w_e(r)$ is called *work curve* for agent A_e and it is some amount of work that depends on the algorithmic output specification, which in its turn is a function of the reported types vector r. We say the output algorithm $g(r)$ is *decreasing* if each of the associated work curves is decreasing (i.e., $w_e(r_{-e}, r_e)$ is a decreasing function of r_e, for all A_e and fixed r_{-e}).

In [1] it is shown that a mechanism is truthful for a problem where each agent's valuation has the above form if and only if the output algorithm $g(r)$ is decreasing, and the payments are given by an explicit formula involving an integral of the the work curve. In particular, if $\int_0^{+\infty} w_e(r_{-e}, z)\, dz$ is bounded for all A_e and r_{-e}, the mechanism satisfies also the voluntary participation and the payment function is of the form

$$p_e(r_{-e}, r_e) = r_e\, w_e(r_{-e}, r_e) + \int_{r_e}^{+\infty} w_e(r_{-e}, z)\, dz. \tag{4}$$

As far as the SPT problem is concerned, let now $g(r)$ denote the output specification of an algorithm computing an SPT of G. Then, for each agent A_e, we can rewrite the valuation (3) as $v_e(t_e, g(r)) = t_e\, w_e(r)$, where

$$w_e(r) = \begin{cases} 1 & \text{if } e \in E(g(r)); \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Then, we can formally define the following mechanism $\mathcal{M}_3$:

1. The algorithmic output specification selects an SPT $S_G(s)$ of G;
2. The payment function for A_e is defined as (4)[1].

[1] From the assumption that the graph G is 2-edge-connected, it follows that (5) satisfies $\int_0^{+\infty} w_e(r_{-e}, z)\, dz < +\infty$, thus implying that we can apply (4), which satisfies voluntary participation.

We can now prove the following result:

Theorem 3. *The mechanism $\mathcal{M}_3$ is a truthful mechanism for the non-utilitarian SPT problem, and is computable on a pointer machine in $O(m + n \log n)$ time.*

Proof. The truthfulness follows from the fact that the output function is decreasing. Indeed, for each agent A_e, if we denote by $\hat{r}_e$ the maximum reported type for e such that e belongs to the computed solution, then the function $w_e(r_{-e}, r_e)$ is equal to 1 for $0 \leq r_e \leq \hat{r}_e$, and is equal to 0 for any $r_e > \hat{r}_e$. Thus, this is a mechanism in the class defined in [1].

From the time complexity point of view, once again the output specification can be computed in $O(m + n \log n)$ time. Concerning the payments, we have to compute all the integrals $\int_{r_e}^{+\infty} w_e(r_{-e}, z)\, dz$, one for each $e \in E(S_G(s))$ (for all other edges, the payment (4) is obviously equal to 0). By definition, we have

$$\int_{r_e}^{+\infty} w_e(r_{-e}, z)\, dz = \hat{r}_e - r_e,$$

from which it follows that for a tree edge, we have that $p_e = \hat{r}_e$. Let now $e = (u, v) \in E(S_G(s))$, with u closer to s that v. Then, e remains in $S_G(s)$ as long as $d_G(s, u) + r_e \leq d_{G-e}(s, v)$, from which it follows that $\hat{r}_e = d_{G-e}(s, v) - d_G(s, u)$. As shown in [12], computing $d_{G-e}(s, v)$ is equivalent to select a non-tree edge such that

$$d_{G-e}(s, v) = \min_{f=(x,y) \in C(e)} \left\{ d_{G-e}(s, x) + r_f + d_{G-e}(y, v) \right\}. \qquad (6)$$

The selection of all the non-tree edges (one for each tree edge) satisfying (6) costs $O(m\, \alpha(m, n))$ time [12]. This means that we can compute all the payments in $O(m\, \alpha(m, n)) = O(m + n \log n)$ time, from which the claim follows. □

References

1. A. Archer and É. Tardos, Truthful mechanisms for one-parameter agents, *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, 482–491, 2001.
2. A.L. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook, Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators, *Proc. 30th ACM Symp. on Theory of Computing (STOC'98)*, 279–288, 1998.
3. Cisco Systems Inc.©, *Internetworking Technologies Handbook*, Cisco Press, 2004.
4. E. Clarke, Multipart pricing of public goods, *Public Choice*, 8:17–33, 1971.
5. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. of the ACM*, 34(3):596–615, 1987.
6. T. Groves, Incentives in teams, *Econometrica*, 41(4):617–631, 1973.
7. L. Gualà and G. Proietti, Optimal MST sensitivity analysis on a pointer machine, *manuscript submitted for publication*, 2005.

8. L. Gualà and G. Proietti, A truthful (2-2/k)-approximation mechanism for the Steiner tree problem with k terminals, *11th Int. Computing and Combinatorics Conference (COCOON'05)*, to appear.
9. J. Hershberger and S. Suri, Vickrey prices and shortest paths: what is an edge worth?, *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS 2001)*, 252–259.
10. K. Malik, A.K. Mittal, and S.K. Gupta, The k most vital arcs in the shortest path problem, *Oper. Res. Letters*, 8:223–227, 1989.
11. E. Nardelli, G. Proietti, and P. Widmayer, A faster computation of the most vital edge of a shortest path, *Info. Proc. Letters*, 79(2):81–85, 2001.
12. E. Nardelli, G. Proietti, and P. Widmayer, Swapping a failing edge of a single source shortest paths tree is good and fast, *Algorithmica*, 36(4):361–374, 2003.
13. E. Nardelli, G. Proietti, and P. Widmayer, Finding the most vital node of a shortest path, *Theoretical Computer Science*, 296(1) (2003) 167–177.
14. E. Nardelli, G. Proietti, and P. Widmayer, Nearly linear time minimum spanning tree maintenance for transient node failures, *Algorithmica*, 40(2):119–132, 2004.
15. N. Nisan and A. Ronen, Algorithmic mechanism design, *Games and Economic Behaviour*, 35:166–196, 2001.
16. M.J. Osborne and A. Rubinstein, A course in Game Theory, *MIT Press*, 1994.
17. S. Pettie and V. Ramachandran, Computing shortest paths with comparisons and additions, *Proc. 13th ACM Symp. on Discrete Algorithms (SODA'02)*, 267–276, 2002.
18. S. Pettie and V. Ramachandran, An optimal minimum spanning tree algorithm, *J. of the ACM*, 49(1):16–34, 2002.
19. G. Proietti and P. Widmayer, A truthful mechanism for the non-utilitarian minimum radius spanning tree problem, *17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'05)*, to appear.
20. R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. of the ACM*, 22(2):215–225, 1975.
21. W. Vickrey, Counterspeculation, auctions and competitive sealed tenders, *J. of Finance*, 16:8–37, 1961.