Internet Mathematics Vol. 5, No. 4: 411–437

Approximate Mechanisms for the Graphical TSP and Other Graph-Traversal Problems

Davide Bilò, Luca Forlizzi, Luciano Gualà, and Guido Proietti

Abstract. Let $G = (V, E)$ be a graph modeling a network in which each edge is owned by a selfish agent, which establishes the cost for traversing its edge (i.e., assigns a weight to its edge) by pursuing only its personal utility. In such a setting, we aim at designing approximate *truthful mechanisms* for several NP-hard traversal problems on G, such as the *graphical traveling salesman problem*, the *rural postman problem*, and the *mixed Chinese postman problem*, each of which in general requires an edge of G to be used several times. Thus, in game-theoretic terms, these are one-parameter problems, but with a peculiarity: the workload of each agent is a natural number. In this paper we refine the classical notion of monotonicity of an algorithm so as to capture exactly this property, and we then provide a general mechanism-design technique that guarantees this monotonicity and that allows one to compute efficiently the corresponding payments. In this way, we show that the former two problems and the latter one admit a 3/2- and a 2-approximate truthful mechanism, respectively. Thus, for the first two problems we match the best known approximation ratios holding for their corresponding centralized versions, while for the third one we are only a 4/3-factor away from it.

1. Introduction

Nowadays, physical components of many large communication and transportation networks are often owned by different economic subjects, which, when asked to provide a service, tend to act selfishly and to pursue only their personal interests. On the other hand, from the user's point of view, there is an increasing demand for a rational allocation of network resources, meaning that one should

1542-7951/08 $0.50 per page

know at each instant, ideally, what is the real marginal contribution that a component can offer. Traditionally, when a systemwide goal has to be implemented on the network, the problem of reconciling this conflict of interest between owners and users has been exclusively addressed by the subfield of game theory known as *mechanism design*. However, more recently, the consciousness that besides economic factors, computational complexity and distributed computing issues must be taken into proper consideration as well has led to increasing involvement in the playground of the computer science community. This has resulted in the emergence of an active research field that is known today as *computational* (or *algorithmic*) *mechanism design* [Nisan and Ronen 01].

Informally speaking, an algorithmic mechanism-design problem can be thought of as a classical well-formulated optimization problem, but with the additional complication that part of the input is retained by the selfish agents. Hence, it turns out that one has to compute *efficiently* a feasible solution to the given optimization problem by *incentivizing* the agents, through suitable *payments*, to disclose to the system their secret data. More formally, a *mechanism* is a pair comprising an *algorithm* for computing a solution and a specification of the *payments* (which is a function of the inputs disclosed by the agents and of the corresponding computed solution) provided to the agents. A mechanism is *truthful* if its payments guarantee that agents are not encouraged to lie.

Since the Internet appears as the ultimate platform where algorithmic mechanism-design optimization problems find application, not surprisingly, most of the efforts so far have concentrated on designing efficient truthful mechanisms for solving several *communication network* problems—under various assumptions on the agents' ownerships—such as the *shortest-path* problem [Hershberger and Suri 01], the *shortest-paths tree* problem [Gualà and Proietti 05a], the *minimum spanning tree* problem [Nisan and Ronen 01], the *minimum-radius spanning tree* problem [Proietti and Widmayer 05], the *minimum-diameter spanning tree* problem [Penna et al. 06], the *minimum Steiner tree* problem [Gualà and Proietti 05b], among many others. All these mechanisms are based either on classical results (*VCG-mechanisms*) [Vickrey 61, Clarke 71, Groves 73], which are applicable whenever the underlying problem is *utilitarian*,[1] or on the results of [Archer and Tardos 01] for the so-called *one-parameter* problems, where the information held by each agent can be expressed through a single value. In particular, in [Archer and Tardos 01] it is shown that the truthfulness of a one-parameter mechanism is related to a property of the underlying algorithm known as *monotonicity*.[2]

[1] Intuitively, a problem is said to be utilitarian whenever the measure of any feasible solution coincides with the *sum* of all the agents' contributions.

[2] Intuitively, an algorithm is said to be monotone whenever it continues to use an agent that is part of a solution as soon as its announced cost decreases.

1.1. Our Results

In this paper, we aim to extend the horizon to a different category of optimization problems, namely that of *graph-traversal problems*. From a purely optimization perspective, such a class of problems has been addressed extensively by many researchers, mainly because of the immediate transportation and logistic applications.

Moreover, because of the renewed interest in toll roads, either managed by governments or private entities, in the past few years the literature devoted to *road pricing* has been considerably enriched, with contributions from both economists and operations researchers (see, for example, [Brotcorne et al. 01] and the literature cited therein).

Along this vein of research, in this paper we study, from an algorithmic mechanism-design point of view, a set of graph-traversal problems in an adversarial setting in which each agent owns a *single* edge of the underlying graph. More formally, let G denote a graph (either directed, undirected, or mixed), with positive real edge weights established by the agents' declarations. A *walk* of length h on G is a nonempty alternating sequence $v_0e_0v_1e_1 \dots e_{h-1}v_h$ of vertices and edges in G such that edge e_i connects vertices v_i, v_{i+1}, for all $i < h$. The *cost* of a walk is the sum of the weights of the edges belonging to it, as counted with their multiplicities (observe that vertices and edges can be repeated). If $v_0 = v_h$, the walk is *closed* and is called a *tour*.

A *path* is a walk in which all vertices are distinct. We will consider the following three classical graph-traversal problems:

1. The *graphical traveling salesman problem* (GTSP): assuming that G is undirected, find a minimum-cost spanning tour of G.

2. The *rural postman problem* (RPP): assuming that G is undirected, and given a subset R of edges of G, find a minimum-cost tour in G that traverses each edge of R *at least* once.

3. The *mixed Chinese postman problem* (MCPP): assuming that G is mixed, find a minimum-cost spanning tour of G traversing each edge of G *at least* once.

It is worth noticing that all the above problems can be considered as meaningful variations of the prominent *traveling salesman problem* (TSP), where the input instance is a *complete* graph, and one has to find a minimum-cost *Hamiltonian cycle* of G (i.e., a minimum-cost spanning tour of G in which all vertices are distinct, apart from the initial and the ending vertex). Unlike the TSP, however, our selected problems do not require the input graph to be complete, and

this is in full accordance with the motivating setting in which each network link must be physically held by a subject; it would be quite unrealistic to assume the existence of a link between each pair of vertices of the graph.

All the above are one-parameter problems, and they are easily seen to be NP-hard [Lenstra and Rinnooy Kan 76, Papadimitriou 76]. Thus, approximate truthful mechanisms (i.e., approximate monotone algorithms) need to be developed. To this end, one generally starts by looking at a corresponding approximate algorithm for the canonical centralized version of the problem, trying to check whether it happens to be (or can easily be transformed to become) monotone. Sometimes, this task is difficult to accomplish, since no general technique is known for establishing the monotonicity of an algorithm, or to "monotonize" it. This is exactly what happens in dealing with our problems. So, in order to devise a uniform approach to this issue, we focus our attention on a quite large class of one-parameter problems, namely those in which the workload of an agent is an integer.

For these problems, we first of all refine the classical notion of monotonicity used in [Archer and Tardos 01] to that of *step-integral monotonicity* (which conveys the fact that each edge can be used several times). Then we develop a general step-integral monotonicity-preserving composition technique between algorithms satisfying certain easier-to-check monotonicity properties, as explained in more detail in Section 3. The usefulness of our technique is twofold: on the one hand, we simplify the question of designing monotone algorithms, and on the other, we provide a way to compute efficiently the payments returned to the agents.

We regard this as the main contribution of this paper, since we foresee the application of this general technique to a wide class of combinatorial optimization problems. In particular, this technique can actually be used to address our problems, for which we are then able to design efficient (in terms of time complexity and approximation ratio) approximate truthful mechanisms. More precisely, as far as the approximation ratios are concerned, we achieve factors of 3/2 and 2 for the former two and the last of the above problems, respectively. Thus, for the first two problems we match the best known approximation ratios holding for their corresponding canonical centralized versions [Frederickson 79], while for the third one we are only a 4/3-factor away from it [Raghavachari and Veerasamy 99].

This paper is organized as follows. Section 2 recalls some preliminaries from mechanism design, while Section 3 describes the general composition technique that will be used to solve our problems. Sections 4, 5, 6 describe our mechanisms for GTSP, RPP, and MCPP, respectively. Finally, Section 7 concludes the paper, by outlining a few interesting issues for future research.

2. Preliminaries: Monotonicity and Truthfulness

Algorithmic mechanism design deals with algorithmic problems in a noncooperative setting, in which part of the input is owned by selfish agents. Since such agents may lie about their parts of the input, they are capable of manipulating the algorithm. The main task of mechanism-design theory is the study of how to incentivize the agents in order to encourage them to behave honestly with the algorithm. We will deal with the case in which each agent controls a single link of a network. We provide a simplified formalization below, and we refer the interested reader to [Nisan and Ronen 01] for deeper insight into the topic.

Let $G = (V, E)$ be a graph (either directed, undirected, or mixed). For an edge e of G owned by a selfish agent a_e, we denote by t_e the private information held by a_e. We call t_e the (private) *type* of the agent a_e, and we assume that t_e represents the real cost incurred by a_e for using its link. Each agent has to declare a (public) *bid* b_e to the mechanism. We denote by t the vector of private types, and by b the *bid profile*, namely the vector of all bids. Let b_{-e} denote the vector of all bids besides b_e; the pair (b_{-e}, b_e) will denote the bid profile b (for the sake of simplifying the notation, we will omit the parentheses whenever (b_{-e}, b_e) appears as the only argument of a function).

For a given optimization problem defined on G, let $\mathcal{F}$ denote the corresponding set of feasible solutions. For each feasible solution $x \in \mathcal{F}$, some measure function $\mu(x, t)$ is defined, which depends on the true types. A *mechanism* is a pair $\mathcal{M} = \langle \mathcal{A}(\mathcal{I}, b), p(b) \rangle$, where $\mathcal{A}(\mathcal{I}, b)$ is an algorithm that, given an instance $\mathcal{I}$ defined on G and given the agents' bids, returns a solution, and $p(b)$ is a scheme that describes the payments provided to the agents. Sometimes, we will simply write $\mathcal{A}(b)$ (respectively $\mathcal{A}$) whenever $\mathcal{I}$ (respectively $\mathcal{I}$ and b) is clear from the context. The *time complexity* of a mechanism corresponds to the time needed to compute $\mathcal{A}$ and p.

For each solution x, a_e incurs a cost $\nu_e(t_e, x)$ (sometimes called the *valuation of* a_e *with respect to* x). The *utility* of an agent is defined as the difference between its payment and its cost with respect to the computed solution. Each agent tries to maximize its utility, while an *exact* mechanism aims to compute a solution that optimizes (i.e., either minimizes or maximizes) $\mu(x, t)$ without knowing t directly. Similarly, if we denote by $\varepsilon(\sigma)$ a positive real function of the input size σ, an $\varepsilon(\sigma)$-*approximation* mechanism returns a solution whose measure comes within a factor $\varepsilon(\sigma)$ of the optimum. In a *truthful* mechanism this tension between the agents and the system is resolved, since each agent maximizes its utility when it declares its type, regardless of what the other agents do.

A mechanism-design problem is called *utilitarian* if its measure function satisfies $\mu(x, t) = \sum_{e \in E} \nu(t_e, x)$. For utilitarian problems, Vickrey, Clarke, and

Groves discovered [Vickrey 61, Clarke 71, Groves 73] a class of truthful mechanisms, i.e., the *VCG-mechanisms*. Basically, VCG-mechanisms handle arbitrary valuation functions, but only utilitarian problems. In [Archer and Tardos 01], it was shown how to design truthful mechanisms for nonutilitarian problems under the assumption that the problem is a *one-parameter* problem, that is, one in which (i) the type of each agent a_e can be expressed as a single parameter $t_e \in \mathbb{R}$, and (ii) each agent's valuation has the form $\nu_e(t_e, x) = t_e\, w_e(x)$, where $w_e(x)$ is called the *workload* for agent a_e in x, i.e., some amount of work assigned by the mechanism's algorithm that depends on the computed solution x.

A well-studied class of one-parameter problems is that of the *binary demand (BD)* problems [Kao et al. 05], in which for each agent a_e, its workload can be either 0 or 1. Given a solution x (respectively an algorithm $\mathcal{A}$), we will denote by $w(x)$ (respectively $w(\mathcal{A})$) the *workload vector* associated with x (respectively returned by $\mathcal{A}$). Recall that an algorithm $\mathcal{A}$ is said to be *monotone* if for all a_e and any fixed b_{-e}, $w_e(\mathcal{A}(b_{-e}, b_e))$ is a nonincreasing function of b_e. We sometimes use $w_e(b_{-e}, b_e)$ instead of $w_e(\mathcal{A}(b_{-e}, b_e))$ when the algorithm is clear from the context. In [Archer and Tardos 01], it is shown that a mechanism for a one-parameter problem is truthful if and only if it makes use of a monotone algorithm, and the payment provided to any agent a_e is equal to

$$p_e(b_{-e}, b_e) = h_e(b_{-e}) + b_e\, w_e(\mathcal{A}(b)) - \int_0^{b_e} w_e(\mathcal{A}(b_{-e}, z))\, dz, \tag{2.1}$$

where $h_e(b_{-e})$ is an arbitrary function independent of b_e. Moreover, in [Archer and Tardos 01] it is shown that if $\int_0^{+\infty} w_e(b_{-e}, z)\, dz < +\infty$ for all a_e and all b_{-e}, then we can use the following payment scheme to obtain a truthful mechanism guaranteeing that the agents' utilities are always nonnegative:

$$p_e(b_{-e}, b_e) = b_e\, w_e(\mathcal{A}(b)) + \int_{b_e}^{+\infty} w_e(\mathcal{A}(b_{-e}, z))\, dz. \tag{2.2}$$

3. The General Composition Scheme

All the algorithms presented in this paper can be naturally decomposed into two simpler ones. In this section we state some general results that will allow us to prove certain properties of a composed algorithm descending from those of the two constituent algorithms. Quite naturally, for all the problems we are going to deal with, we assume that $\mathcal{F} \neq \varnothing$. Moreover, we assume that no agent is

indispensable, namely that for each agent there always exists a feasible solution not depending on it. These two assumptions will be reflected by the connectivity properties that G needs to satisfy, depending on the specific problem. Finally, for each of the proposed algorithms, we tacitly assume that at each step a suitable tie-breaking rule is applied, if needed, in order to ensure monotonicity.

Given a solution x, let us assume that the cost incurred by any agent a_e in x is equal to its type t_e times the number of occurrences of e in x. It is easy to see that under this assumption, our problems fall within the class of one-parameter problems, since the number of occurrences of e in x is exactly the workload $w_e(x)$, and $w_e(x) = 0$ if e is not part of x. In general, we say that a solution x does not depend on e when $w_e(x) = 0$. However, notice that in the MCPP, for every edge e and any feasible solution x, from the definition of the problem, e must occur in x at least once. The same happens for every edge $e \in R$ in the RPP.

In both cases, we consider that a solution x does not depend on e if $w_e(x) = 1$. Moreover, when we compute the integral in (2.2) for e, to avoid technicalities, we do not count the first occurrence in the workload $w_e(\cdot)$, i.e., we take $w_e(\cdot)$ decremented by 1. This means that we implicity assume that the mechanism does not care about the cost incurred by a_e for the constrained occurrence. Thus, we consider one-parameter, utilitarian problems in which the workload of each agent is a natural number.

Definition 3.1. An algorithm $\mathcal{A}$ of a one-parameter mechanism is said to be *step-integral monotone* (SIM) if $\mathcal{A}$ is monotone and the workload of each agent belongs to $\mathbb{N}$.

For any SIM algorithm $\mathcal{A}$, and any fixed b_{-e}, we define the *thresholds* for a_e with respect to $\mathcal{A}$ to be the discontinuity points of the function $f_e(z) := w_e(\mathcal{A}(b_{-e}, z))$, and we denote them, sorted in increasing order, by $\theta_1, \theta_2, \ldots, \theta_h$. Notice that computing the integral in (2.1) and (2.2) essentially consists in determining the thresholds. Also note that for a BD problem, a monotone algorithm defines a unique threshold, and for an agent a_e, the payment $p_e(b)$ is exactly that threshold value if a_e owns a selected edge, and 0 otherwise. Basically, this threshold defines the supremum value that a_e is allowed to declare to be part of a solution.

We say that $\mathcal{A}$ is a *composition* of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2 := \mathcal{A}_2(\mathcal{I}(\mathcal{A}_1), b)$ (and we will write $\mathcal{A} = \mathcal{A}_2 \diamond \mathcal{A}_1$) if $\mathcal{A}$ adheres to the scheme listed in Algorithm 1. Notice that in such an algorithm, `CreateInstance` is a generic procedure that uses the output of the first algorithm to generate an instance of the second one.

Algorithm 1. $(\mathcal{A} = \mathcal{A}_2 \diamond \mathcal{A}_1)$

1. let x_1 be the output returned by $\mathcal{A}_1$;
2. $\mathcal{I}(\mathcal{A}_1) = \texttt{CreateInstance}(x_1)$;
3. let x_2 be the output returned by $\mathcal{A}_2(\mathcal{I}(\mathcal{A}_1), b)$;
4. let x be a solution built from x_1 and x_2 such that $w_e(x) = w_e(x_1) + w_e(x_2), \forall a_e$;
5. **return**: x

Finally, we say that $\mathcal{A}$ is *stable* if for any a_e and for any fixed b_{-e}, $\forall\, z, z' \in \mathbb{R}$, we have

$$w_e(\mathcal{A}(b_{-e}, z)) = w_e(\mathcal{A}(b_{-e}, z')) \text{ iff } \mathcal{A}(b_{-e}, z) = \mathcal{A}(b_{-e}, z').$$

A first useful consequence of the proposed decomposition scheme is that the SIM property of the composed algorithm is guaranteed whenever the composing algorithms satisfy suitable properties, as proved in the following theorem.

Theorem 3.2. *Let $\mathcal{A}_1$ be a stable SIM algorithm for a one-parameter problem, and let $\mathcal{A}_2$ be a monotone algorithm for a BD problem. Then $\mathcal{A} = \mathcal{A}_2 \diamond \mathcal{A}_1$ is a SIM algorithm.*

Proof. It is clear that workloads assigned by $\mathcal{A}$ are natural numbers. It remains to prove the monotonicity property. Let b_{-e} be fixed, and let b_e and b'_e be such that $b'_e \geq b_e$. We have to prove that $w_e(\mathcal{A}(b_{-e}, b_e)) \geq w_e(\mathcal{A}(b_{-e}, b'_e))$. Let then $\mathcal{I} := \mathcal{I}(\mathcal{A}_1(b_{-e}, b_e))$ and $\mathcal{I}' := \mathcal{I}(\mathcal{A}_1(b_{-e}, b'_e))$. By definition of $\mathcal{A}$, we have $w_e(\mathcal{A}(b_{-e}, b_e)) = w_e(\mathcal{A}_1(b_{-e}, b_e)) + w_e(\mathcal{A}_2(\mathcal{I}, (b_{-e}, b_e)))$, with $w_e(\mathcal{A}_1(b_{-e}, b_e)) \geq w_e(\mathcal{A}_1(b_{-e}, b'_e))$ holding from the monotonicity of $\mathcal{A}_1$. We have two cases:

1. $w_e(\mathcal{A}_1(b_{-e}, b_e)) = w_e(\mathcal{A}_1(b_{-e}, b'_e))$: Then since $\mathcal{A}_1$ is stable, we have $\mathcal{I} = \mathcal{I}'$, and then from the monotonicity of A_2 we have $w_e(\mathcal{A}_2(\mathcal{I}, (b_{-e}, b_e))) \geq w_e(\mathcal{A}_2(\mathcal{I}', (b_{-e}, b'_e)))$, from which the claim follows.
2. $w_e(\mathcal{A}_1(b_{-e}, b_e)) > w_e(\mathcal{A}_1(b_{-e}, b'_e))$: Then since $w_e(\mathcal{A}_2(\mathcal{I}', (b_{-e}, b'_e))) \leq 1$, the claim follows.

□

We now concentrate on the computation of payments, and we provide a general method to compute in polynomial time the thresholds for a certain class of problems. We denote by $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{P}_{\mathcal{A}}(e)$ the running time for algorithm $\mathcal{A}$ and for computing the payment for agent a_e with respect to $\mathcal{A}$, respectively. Then we have the following theorem.

Theorem 3.3. *Let $\mathcal{A} = \mathcal{A}_2 \diamond \mathcal{A}_1$ be a polynomial-time algorithm of a one-parameter mechanism, where $\mathcal{A}_1$ is a stable SIM algorithm for a one-parameter problem, and $\mathcal{A}_2$ is a monotone algorithm for a BD problem. If there exist polynomial-time algorithms for computing the thresholds with respect to $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, then the thresholds for agent a_e with respect to $\mathcal{A}$ can be computed in $O(\mathcal{P}_{\mathcal{A}_1}(e) + h(\mathcal{T}_{\mathrm{CI}} + \mathcal{P}_{\mathcal{A}_2}(e)))$ time, where h is the number of thresholds for a_e with respect to $\mathcal{A}_1$, and $\mathcal{T}_{\mathrm{CI}}$ is the time required by the procedure* `CreateInstance`.

Proof. We provide a polynomial-time procedure to compute thresholds with respect to $\mathcal{A}$. The proof of its correctness is quite simple, and we omit it. Let b_{-e} be fixed. Let $\theta_1, \ldots, \theta_h$ be the thresholds for a_e with respect to $\mathcal{A}_1$, and let $\theta_0 = 0$, $\theta_{h+1} = \infty$. For $1 \leq i \leq h$, we denote by ω_i the workload assigned by $\mathcal{A}_1$ to a_e when $\theta_{i-1} < b_e < \theta_i$.

We now show how to compute thresholds of a_e. A *pre-threshold* is a pair (θ, ω), where θ is a threshold value and ω is the corresponding workload value. The following procedure computes a list of pre-thresholds:

1. let L be the list of pairs (θ_i, ω_i), $\forall\, 1 \leq i \leq h$, ordered according to the θ_i;
2. $\forall\ 1 \leq i \leq h+1$:
   (a) let $\mathcal{I}(\mathcal{A}_1)$ be the instance computed by `CreateInstance`$(\mathcal{A}_1(b_{-e}, z))$, where $\theta_{i-1} < z < \theta_i$ (notice that since $\mathcal{A}_1$ is stable, the same instance will be computed for any z in the interval between θ_{i-1} and θ_i);
   (b) let β_i be the threshold for a_e with respect to $\mathcal{A}_2$ on the instance $\mathcal{I}(\mathcal{A}_1)$;
   (c) compare β_i and (θ_{i-1}, θ_i):
      i. case $\beta_i < \theta_{i-1}$: nothing happens;
      ii. case $\theta_{i-1} < \beta_i < \theta_i$: insert $(\beta_i, \omega_i + 1)$ in L (respecting the order);
      iii. case $\theta_i < \beta_i$: update (θ_i, ω_i) to $(\theta_i, \omega_i + 1)$.

Notice that L, computed as above, is sorted in increasing order with respect to the pairs' first values, and in nonincreasing order with respect to the pairs' second values. Thresholds of $\mathcal{A}$ are then obtained by selecting the maximum threshold value in each set of pre-thresholds having the same workload value. The bound on the running time follows immediately. □

The above result can be enhanced as soon as the algorithm $\mathcal{A}_1$ solves a one-parameter problem that is also utilitarian. Indeed, in this case the following lemma can be given, which shows how to compute thresholds for an agent with respect to such an algorithm using a Newton-like iterative algorithm.

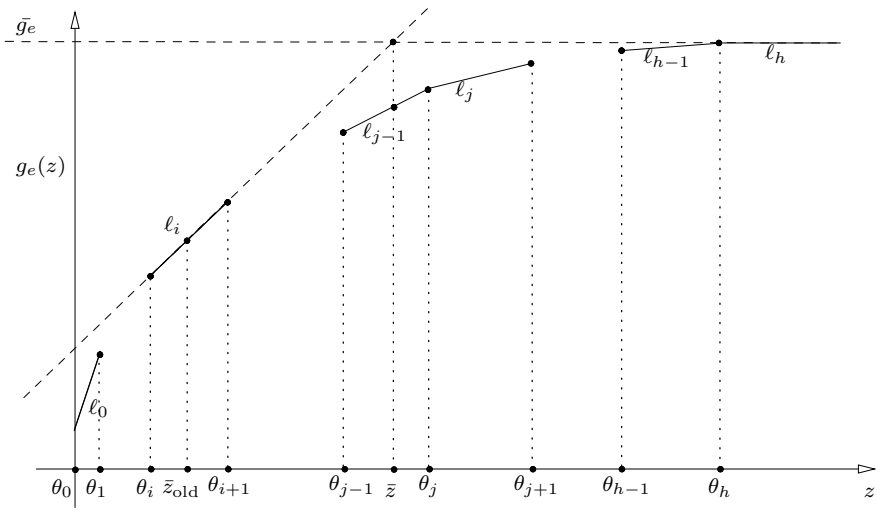

Figure 1. The auxiliary function $g(z)$ and a step of the execution of the Newton-like algorithm.

Lemma 3.4. *Given a one-parameter and utilitarian problem, solvable in polynomial time by a given stable SIM algorithm $\mathcal{A}$, the thresholds $\theta_1, \theta_2, \ldots, \theta_h$ with respect to $\mathcal{A}$ of any agent are computable in $O(h\,\mathcal{T}_{\mathcal{A}})$ time.*

Proof. For any vector bid b, $\mathcal{A}$ determines an optimal solution x and an associated workload vector $w(x)$ using agent bids as the real costs of network links. We recall that given an agent a_e, for any fixed b_{-e}, $w_e(b_{-e}, z)$ is the workload assigned to a_e by an optimal solution when the cost of e is z, and that in the cases of RPP and MCPP, if e is required to occur at least once in any solution, we implicitly consider (for the purpose of computing thresholds for a_e) $w_e(\cdot)$ decremented by 1. Then since $\mathcal{A}$ is a SIM algorithm, $w_e(b_{-e}, z)$ has the following properties:

1. for any z, $w_e(b_{-e}, z)$ is a nonnegative integer;
2. $w_e(b_{-e}, z)$ is a nonincreasing function of z;
3. $\lim_{z\to\infty} w_e(b_{-e}, z) = 0$.[3]

Then for any agent a_e, we consider the function

$$g_e(z) = \sum_{f\in E\setminus\{e\}} b_f w_f(b_{-e}, z) + w_e(b_{-e}, z)z,$$

[3]This follows from the assumption that no agent is indispensable, i.e., there is always a feasible solution not using e.

Algorithm 2. (Computation of the threshold θ_i)

input : ℓ_i
$\tilde{z} = 0$; {starting point of the sequence}
while $(\tilde{z}, g_e(\tilde{z})) \notin \ell_i$ {*loop until the threshold is discovered*} **do**
  let ℓ be the support line containing $(\tilde{z}, g_e(\tilde{z}))$;
  let $\tilde{z}$ be the abscissa of the intersection between ℓ and ℓ_i;
end
return: $\theta_i = \tilde{z}$ and $\ell_{i-1} = \ell$

which describes the evolution of the cost of the solution computed by $\mathcal{A}$ as a_e's bid varies but b_{-e} is fixed. From the above-mentioned properties of $w_e(\cdot)$, it is easy to see that $g_e(\cdot)$ is a piecewise-linear, continuous, nondecreasing, concave function (see Figure 1) whose first derivative coincides with $w_e(b_{-e}, z)$ when z is not a threshold.[4] Then to compute thresholds means to compute the discontinuity points of the first derivative of g_e. Also, the properties of $w_e(\cdot)$ imply that for $z \geq \theta_h$, $w_e(b_{-e}, z) = 0$, so $g_e(z)$ reaches its maximum value $\bar{g_e}$.

Let us introduce the useful dummy thresholds $\theta_0 = 0$ and $\theta_{h+1} = \infty$. For any $0 \leq i \leq h$ we define the line connecting $(\theta_i, g_e(\theta_i))$ with $(\theta_{i+1}, g_e(\theta_{i+1}))$ as the *support line* ℓ_i. Notice that since $g_e(\cdot)$ is a concave function, for any i_1, i_2 such that $i_1 < i_2 \leq h$, ℓ_{i_1} has slope greater than that of ℓ_{i_2} (see Figure 1).

For a fixed value z of a_e's bid, we can compute $w_e(z)$ and $g_e(z)$ by running $\mathcal{A}$. Having the values $w_e(z)$ and $g_e(z)$, it is trivial to determine the support line on which $(z, g_e(z))$ lies. In particular, the value $\bar{g_e}$ assumed by $g_e(\cdot)$ for points greater than the last threshold θ_h can be computed by running $\mathcal{A}$ with input bid profile (b_{-e}, ∞), and ℓ_h is the horizontal line having constant value $\bar{g_e}$.

Using the knowledge of the support line ℓ_i, it is possible to find the value of θ_i using a Newton-like iterative approximation algorithm. Such an algorithm gives as a byproduct the knowledge of support line ℓ_{i-1}, which can be used, applying again the algorithm, to compute the threshold θ_{i-1}. The idea is to compute, starting from the point $z_0 = 0$, a sequence of points $z_1, z_2, \ldots$ such that z_j is the intersection of the support line containing $(z_{j-1}, g_e(z_{j-1}))$ with ℓ_i. The sequence halts when a point z_j lying on ℓ_i is found. More formally, the computation of the θ_i follows the pseudocode of Algorithm 2:

We now show that the sequence $z_0, z_1, \ldots$ converges to θ_i. By definition, $z_0 \leq \theta_i$. Assuming inductively that $z_j \leq \theta_i$ for some $j \geq 0$, we now show that either $z_j = \theta_i$ or $z_{j+1} \leq \theta_i$. If $(z_j, g_e(z_j)) \notin \ell_i$, the body of the while-loop is executed and the next point in the sequence, i.e., z_{j+1}, is computed. Observe

[4] Note that when z is a threshold, $w_e(b_{-e}, z)$ is defined, while the first derivative of g_e is not.

that $(z_j, g_e(z_j)) \notin \ell_i$ immediately implies that $z_j < \theta_i$, that $(z_j, g_e(z_j))$ is strictly below ℓ_i (since g_e is a concave function), and that the support line ℓ_{z_j} containing $(z_j, g_e(z_j))$ has slope greater than that of ℓ_i. Then $z_j < z_{j+1}$ follows.

Similarly, by the concavity of g_e, the point $(\theta_i, g_e(\theta_i))$ is not above ℓ_{z_j}, and since ℓ_{z_j} has slope greater than that of ℓ_i, we have $z_{j+1} \leq \theta_i$ at the beginning of the next iteration of the loop. On the other hand, if $(z_j, g_e(z_j)) \in \ell_i$, the algorithm halts. Then since by definition, $\theta_i = \min_{(x, g_e(x)) \in \ell_i} x$, the inductive hypothesis implies $z_j = \theta_i$.

Now we show that the sequence $z_0, z_1, \ldots$ converges in a finite number of steps. More exactly, Algorithm 2 executes at most h iterations of the while-loop, because either the support line $\ell_{z_{j+1}}$ of $(z_{j+1}, g_e(z_{j+1}))$ is different from ℓ_{z_j}, or the algorithm halts. Indeed, suppose that $\ell_{z_{j+1}}$ is the same as ℓ_{z_j}. Then $(z_{j+1}, g_e(z_{j+1}))$ is the intersection point of ℓ_{z_j} and ℓ_i, so $(z_{j+1}, g_e(z_{j+1})) \in \ell_i$.

Notice that the support line containing the penultimate point in the sequence is ℓ_{i-1}. This knowledge allows us to compute θ_{i-1} by repeating the same algorithm. Therefore, since ℓ_h is known in advance, we can compute all thresholds $\theta_1, \theta_2, \ldots, \theta_h$ by iterating Algorithm 2 (at most) h times. Each run of Algorithm 2 calls $\mathcal{A}$ at most h times; hence to compute all thresholds requires $O(h^2\, \mathcal{T}_{\mathcal{A}})$ time.

We can in fact refine the upper bound of computation of the thresholds to $O(h\, \mathcal{T}_{\mathcal{A}})$ time. To this end, instead of computing a different sequence of points for each threshold, we store a global sequence of points $z_0, z_1, \ldots$ and the corresponding values of $g_e(\cdot)$ for all threshold computations. After determining the threshold θ_i, the last point in the sequence, say $(z_j, g_e(z_j))$, lies on ℓ_{i-1}, so it can be dropped from the sequence, while z_{j-1} can be used as a new starting point to discover more points up to a new threshold. It is clear that this improved algorithm computes only one point for each support line, and that the time complexity is proportional to the number of computed points. □

Finally, combining the results of Lemma 3.4 and Theorem 3.3, we obtain the following main result.

Theorem 3.5. *Let $\mathcal{A} = \mathcal{A}_2 \diamond \mathcal{A}_1$ be a polynomial-time algorithm of a one-parameter mechanism, where $\mathcal{A}_1$ is a stable SIM algorithm for a one-parameter utilitarian problem, and $\mathcal{A}_2$ is a monotone algorithm for a BD problem. If the threshold for a_e with respect to $\mathcal{A}_2$ can be computed in $O(\mathcal{T}_{\mathcal{A}_2})$ time, then the thresholds for a_e with respect to $\mathcal{A}$ can be computed in $O(h\, \mathcal{T}_{\mathcal{A}})$ time, where h is the number of thresholds for a_e with respect to $\mathcal{A}_1$.*

Proof. The theorem follows immediately from Theorem 3.3 if one uses the fact that $\mathcal{P}_{\mathcal{A}_2}(e) = O(\mathcal{T}_{\mathcal{A}_2})$ and computes thresholds for a_e with respect to $\mathcal{A}_1$ with the method described in Lemma 3.4. □

4. Solving the GTSP

Given an undirected graph $G = (V, E)$, with $n = |V|$, $m = |E|$, we consider the problem of computing a minimum-cost spanning tour of G. This problem is utilitarian and is equivalent to the classical TSP for the metric instances. Thus we can use for our problem a modified version of Christofides algorithm [Christofides 76], where a *path matching* [Böckenhauer et al. 02] instead of a matching takes place, and where we do not shortcut repeated vertex occurrences (such modifications allow us to prove easily the algorithm's monotonicity, and do not change its approximation ratio):

1. Compute the minimum spanning tree (MST) T of G.
2. Let $U \subseteq V$ be the set of odd-degree vertices in T, and let D be the complete graph on U, where edge $\{u, v\}$ has weight equal to the cost of a shortest path connecting u and v in G, say $d_G(u, v)$.
3. *Path-matching (PM) algorithm*: compute a minimum-cost perfect matching M of D, and let F be the *expansion of* M, i.e., the multiset composed by taking, for each $\{u, v\} \in M$, the edges forming the shortest path between u and v in G.
4. Form an Eulerian multigraph $H = (V, E')$ on G consisting of the edges of T and the edges in F;
5. Return an Eulerian tour of H.

Notice that our algorithm is a composition (as defined in Section 3) of an algorithm for the MST and the PM algorithms. It is easy to prove that it has approximation ratio 3/2 and that its time complexity is dominated by the $O(n^3)$ time required to compute a minimum-cost perfect matching on a complete graph. It is well known that any MST algorithm is monotone, BD, as well as stable. Since the expansion of M forms an edge-disjoint forest in G (see [Böckenhauer et al. 02]), it is straightforward to see that the PM algorithm is BD (notice that this implies that the workloads will be at most 2). Then to prove the step-integral monotonicity of our algorithm, we use Theorem 3.2 after showing that PM is a monotone algorithm.

Lemma 4.1. *The PM algorithm is monotone.*

Proof. Let us fix a graph G and a set of vertices $U \subseteq V$ as the input of the PM algorithm. We denote the cost of a set S of edges of G when the bid profile is b by $c(b, S) = \sum_{e \in S} b_e$. Let us consider a bid profile b given as input to PM, and let F be the corresponding computed solution. Assume that $e \in E \setminus F$ is an edge not selected as part of the output. For $b'_e > b_e$, let F' be the solution computed by PM when the input bid profile is $b' = (b_{-e}, b'_e)$. We have to show that $e \notin F'$.

For the sake of contradiction, assume that $e \in F'$, which implies that $c(b, F') < c(b', F')$. Since PM computes an optimal solution, $c(b, F) \leq c(b, F')$, and since $e \notin F$, it follows that $c(b', F) = c(b, F)$. Then we have $c(b', F) < c(b', F')$, which is a contradiction, since F' is the output of PM when the bid profile is b'. □

Now we show how to compute thresholds with respect to the PM algorithm. Given a bid profile b and $U \subseteq V$, let F be the solution computed by the PM algorithm in b, and assume that $e \in F$ is a selected edge. We denote by $F_{\neg e}$ the solution returned by the PM algorithm in $G - e$. It is easy to see that the threshold for a_e is $c(b, F_{\neg e}) - c(b, F) + b_e$, which is clearly computable with the same asymptotic time bound as PM. Similarly, the thresholds for a_e with respect to the MST algorithm can be computed by running the MST algorithm on $G - e$. Then from Theorem 3.5, and observing that there are at most $O(n)$ agents in a feasible solution, we have the following theorem.

Theorem 4.2. *There exists an $O(n^4)$-time 3/2-approximate truthful mechanism for the GTSP in which each edge is owned by a distinct selfish agent.*

5. Solving the RPP

In the rural postman problem (RPP), we are given an undirected graph $G = (V, E)$, with $n = |V|$, $m = |E|$, and a set $R \subseteq E$, with $k = |R|$. We are required to compute a minimum-cost tour in G traversing each edge of R *at least* once.

As observed in [Frederickson 79], this problem has a 3/2-approximation algorithm that is a minor modification of Christofides' algorithm. Similarly, our algorithm for the GTSP can be adapted to this problem, still giving a 3/2-approximation in $O(n^3)$ time (notice that in this case, edge workloads may be greater than 2):

1. Let G' be a complete graph on the vertex set $V' = \{v \in V \mid \exists e \in R \wedge v \in e\}$, where an edge e of G' has weight equal to b_e if $e = \{u, v\} \in R$, and to

$d_G(u, v)$ otherwise, and let T' be an MST of G' containing R; then build the multiset of edges x_1 as the union of R and of the edges forming a shortest path in G between u and v, for any $\{u, v\} \in E(T') \setminus R$.

2. Let $U' \subseteq V'$ be the set of odd-degree vertices in T', and let D' be the complete graph on U', where an edge $\{u, v\}$ of D' has weight $d_G(u, v)$.

3. (PM algorithm): compute a minimum-cost perfect matching M' of D', and let x_2 be the multiset of edges forming a shortest path in G between u and v, $\forall\{u, v\} \in M'$.

4. Form the Eulerian multigraph induced by edges in x_1 and x_2, and compute an Eulerian tour x on it.

5. Return x.

To prove that the algorithm is SIM, it suffices to show that step 1 is a stable SIM algorithm.

Lemma 5.1. *Step 1 of the above algorithm is a stable SIM algorithm.*

Proof. To prove that step 1 is a SIM algorithm, it suffices to show that it is monotone. We first observe that any agent holding an edge in R has workload 1. Let e be an edge of $E \setminus R$. Then e belongs to some shortest paths in G between pairs of vertices in V'. If a_e increases its bid, step 1 computes a new solution x'_1 that retains all paths in x_1 not containing e, but in which some of the paths of x_1 containing e may be replaced by an equal number of shortest (with respect to the new bid profile) paths. Such new paths do not contain e; otherwise, they would be cheaper than the previous ones also with respect to the old bid profile. Then the workload of a_e in x'_1 is not greater than that in x_1. The stability follows from the tie-breaking rules, since if a_e's load in x'_1 is the same as in x_1, then x_1 has the same cost as x'_1 with respect to the new profile. □

Then by Lemma 4.1 and Theorem 3.2 we have that our algorithm is SIM. Since we are able to compute the thresholds with respect to the PM algorithm (see Section 4), in view of Theorem 3.3, to compute payments for the complete algorithm we have only to find the thresholds of the algorithm in step 1. To this end, we use the method given in Lemma 3.4. Referring to such an algorithm, we observe that for each $e \in R$, the workload of a_e is at least 1. We recall that for such edges, we implicitly decrease the workload by 1 when computing the integral appearing in the payment scheme (2.2), in order to guarantee that it has a finite value.

To analyze the time complexity of the mechanism, we first observe that the time complexity of the algorithm is dominated by the PM algorithm on a complete graph with $2k$ vertices, which requires $O(k^3)$ time, and by the computation of G'. The latter requires the computation of the all-pairs shortest paths between $2k$ vertices of G, which, depending on k, can be accomplished either in $O(nm \log \alpha(m,n))$ time by computing the all-pairs distances of G [Pettie and Ramachandran 02], or in $O(k(m + n \log n))$ time by k executions of Dijkstra's algorithm. To determine the time complexity of the computation of the thresholds, we then use the result of Theorem 3.5, by noticing that for each agent a_e we have $w_e(x_1) \leq k$, and that there are at most $O(n + k)$ distinct agents in a feasible solution. Thus, the following theorem summarizes the results of this section.

Theorem 5.2. *There exists an* $O((n+k)\, k\, (\min\{nm \log \alpha(m,n), k(m+n \log n)\}+k^3))$*-time* $3/2$*-approximate truthful mechanism for the RPP in which each edge is owned by a distinct selfish agent.*

6. Solving the MCPP

In the Chinese postman problem, we are given a graph G and we are required to compute a minimum-cost spanning tour of G traversing each edge *at least* once. The problem was shown to be efficiently solvable in [Edmonds 65] in the case that the input graph is undirected, and in [Edmonds and Johnson 73] when the input graph is directed. On the other hand, when a mixed input graph is permitted, we have the MCPP, whose decision version was shown to be NP-complete in [Papadimitriou 76]. We point out that the main difficulty of the MCPP is that to obtain a feasible solution we have to extend the input graph G by duplicating some edges, in order to obtain a multigraph satisfying two distinct properties:

1. Each vertex has even degree (*even* property).

2. Each vertex has equal in- and out-degrees (*in–out* property).

Note that it is possible to compute, in polynomial time, a cheapest extension of G satisfying each of the above properties taken alone. The most successful approach to approximating MCPP is to employ two distinct simpler algorithms. Roughly speaking, the first algorithm duplicates edges optimally so to satisfy the even property and then performs other duplications to satisfy the in–out property. The second algorithm adopts the converse strategy. On each input instance, both simpler algorithms are executed, and the best outcome is returned.

This approach was introduced in [Frederickson 79], leading to a 5/3-approximate algorithm that was later refined in [Raghavachari and Veerasamy 99] to a 3/2-approximate algorithm. Unfortunately, both algorithms turn out not to be monotone, as shown in Section 6.1. On the other hand, we show in Section 6.2 that one of the above-mentioned simpler algorithms is, on its own, a monotone 2-approximate algorithm.

6.1. Some Good Algorithms for MCPP Are Not Monotone

We first describe the 5/3-approximate algorithm designed in [Frederickson 79]. The algorithm selects the best outcome returned by two simpler algorithms, named `Mixed1` and `Mixed2`.

`Mixed1` computes a feasible solution by first satisfying the even property and then the in–out property. It proceeds as follows:

1. Extend G to a multigraph G' satisfying the even property by running on G a procedure called `EvenDegree`, which adds a min-cost path-matching (ignoring edge directions) of the odd-degree vertices.

2. Satisfy the in–out property by means of Algorithm `InOutDegree`, described in Section 6.2 and in [Frederickson 79], which orients some undirected edges and duplicates some edges.

3. Restore the even property in case it was violated as a result of the previous step. This is feasible in polynomial time, without increasing the cost of the provided solution, using the procedure `EvenParity` described in [Frederickson 79].

`Mixed2` computes a feasible solution by first satisfying the in–out property and then the even property, as follows:

1. Duplicate edges of G so as to satisfy the in–out property using `InOutDegree`.

2. Duplicate edges in the outcome of previous step to satisfy the even property by adding a min-cost path-matching, formed by undirected edges only, of the odd-degree vertices.

In Section 6.2 we will study in more detail `Mixed2`, showing that on its own, it is a monotone algorithm.

The following example shows that the 5/3-approximate algorithm designed in [Frederickson 79] is not monotone. Consider the input instance G depicted in Figure 2, where the cost b_e of edge e is not yet fixed and ε is a very small value.

Since the input instance satisfies the in–out property, the first step of `Mixed2` leaves it unmodified. For any value of b_e, the min-cost path-matching of the

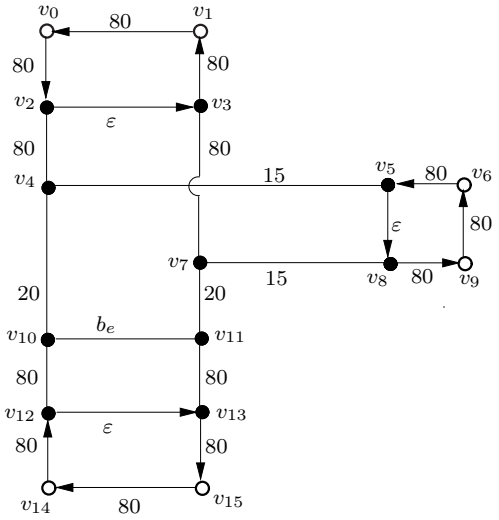

Figure 2. Input instance for MCPP: filled vertices have odd degree.

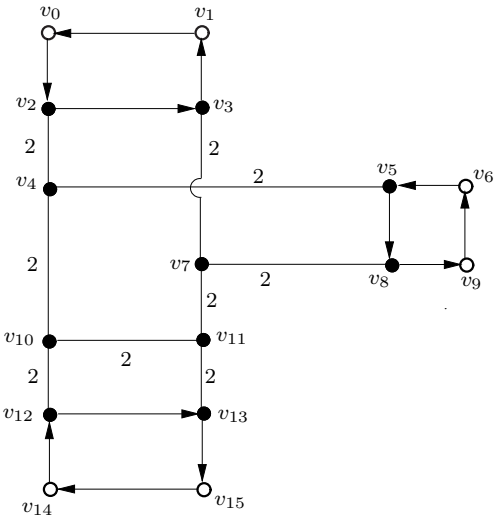

Figure 3. Outcome of `Mixed2`: labels count edge repetitions.

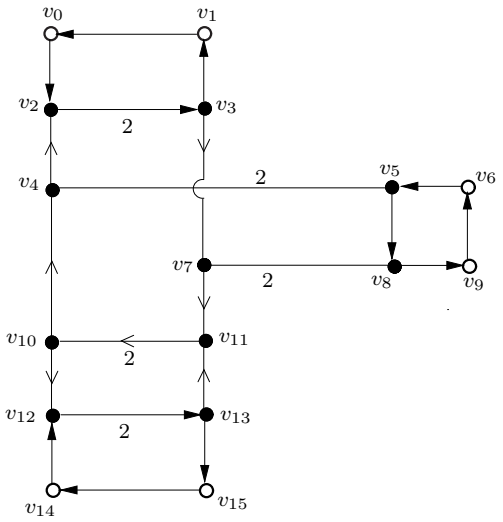

Figure 4. Outcome of `Mixed1` for $b_e = 9$: labels count edge repetitions, and added arrows indicate undirected edges oriented by the algorithm.

odd-degree vertices formed by undirected edges only, added in the second step, doubles any undirected edge, forming the feasible solution illustrated in Figure 3. Such a solution costs $1500 + 3 \cdot \varepsilon + 2 \cdot b_e$.

Suppose now that $b_e = 9$, and consider what `Mixed1` does. In the first place, it computes a min-cost path-matching (actually, a matching) of the odd-degree vertices. It consists of the edges $(v_2, v_3), \{v_4, v_5\}, \{v_7, v_8\}, \{v_{10}, v_{11}\}, (v_{12}, v_{13})$. Note that the augmented graph violates the in–out property. In its second step, `Mixed1` satisfies such a property just by orienting some undirected edges, thus obtaining a feasible solution with no additional cost increase, as shown in Figure 4 (observe that in this case, the even property is preserved after the execution of the second step; hence the third step does nothing).

The solution computed by `Mixed1` for $b_e = 9$ costs $1140 + 5 \cdot \varepsilon + 2 \cdot b_e$, which is smaller than the cost of the solution computed by `Mixed2`. Therefore, the former is the solution returned by the whole algorithm, and $w_e(b_{-e}, 9) = 2$.

Suppose now that a_e increases its bid up to $b_e = 11$. Then the min-cost path-matching of the odd-degree vertices computed in the first step changes. Now it consists of the edges $(v_2, v_3), \{v_4, v_{10}\}, \{v_5, v_8\}, \{v_7, v_{11}\}, (v_{12}, v_{13})$. Also in this case, the augmented graph violates the in–out property. To restore that property, it is not hard to check that `Mixed1` orients some undirected edges and also adds two additional copies of e. The obtained feasible solution is shown in Figure 5 (in this case, too, the second step of `Mixed1` preserves the even property, so the third one does nothing).

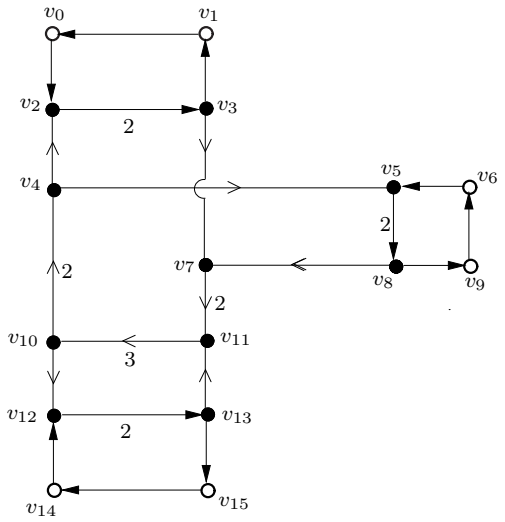

Figure 5. Outcome of `Mixed1` for $b_e = 11$: labels count edge repetitions, and added arrows indicate undirected edges oriented by the algorithm.

The solution computed by `Mixed1` for $b_e = 11$ costs $1150 + 6 \cdot \varepsilon + 3 \cdot b_e$, which is again smaller than the cost of the solution computed by `Mixed2`. Therefore, the former is the solution returned by the whole algorithm, and $w_e(b_{-e}, 11) = 3 > w_e(b_{-e}, 9)$, thus showing the nonmonotonicity.

We now study the 3/2-approximate algorithm designed in [Raghavachari and Veerasamy 99]. The algorithm selects the best outcome returned by `Mixed2` and an algorithm named `Modified Mixed1`.

`Modified Mixed1`, like `Mixed1`, from which it derives, computes a feasible solution by first satisfying the even property, and then the in–out property. It diverges from `Mixed1` in that it executes a preliminary step that changes the cost of some edge just for the following step. Then the input instance is extended to satisfy the even property in a suboptimal way, which then makes it possible also to satisfy the in–out property with a better approximation guarantee. In detail:

1. Run `InOutDegree` applied to G to determine a minimum-cost extension M of G that satisfies the in–out property.

2. Run `EvenDegree` applied to G with costs of edges contained in M set to 0, so as to satisfy the even property.

3. Run `InOutDegree` applied to the outcome of the previous step, so as to satisfy the in–out property.

4. In case the outcome of the previous step violates the even property, restore it by running `EvenParity`.

Let us see how the 3/2-approximate algorithm designed in [Raghavachari and Veerasamy 99] performs on the input instance in Figure 2. `Mixed2` runs as when used within the 5/3-approximate algorithm, as previously described. On the other hand, since the input instance already satisfies the in–out property, the first step of `Modified Mixed1` does not extend it, and sets M equal to the set of directed edges of G. Therefore, in the second step, `EvenDegree` is run on G with the cost of the directed edges set to 0. In both cases $b_e = 9$ and $b_e = 11$, the minimum-cost matching computed with all edge costs set to the original values (i.e., by `Mixed1`) is a minimum-cost matching with the modified costs, too. If such a matching is returned in the second step, the algorithm proceeds like `Mixed1`, producing the same feasible solutions. Hence $w_e(b_{-e}, 11) = 3 > 2 = w_e(b_{-e}, 9)$, which means that the 3/2-approximate algorithm, too, is not monotone. Notice that in both cases $b_e = 9$ and $b_e = 11$, other minimum-cost matching could be computed (for instance

$$(v_3, v_1),\ (v_1, v_0),\ (v_0, v_2),\ \{v_4, v_5\},\ \{v_7, v_8\},\ \{v_{10},\ v_{11}\},\ (v_{12}, v_{13})$$

for $b_e = 9$), leading to different feasible solutions not proving nonmonotonicity. However, if we change the value of ε to a greater value, say 80, one can check that the 3/2-approximate algorithm computes feasible solutions exhibiting nonmonotonicity regardless of the selected minimum-cost matchings.

6.2. A Monotone Algorithm for MCPP

In search of an approximate truthful mechanism for the MCPP, we restrict our attention to the algorithm `Mixed2`, which was originally developed in [Frederickson 79], and for which we can prove monotonicity. This algorithm was shown to have an approximation ratio of 2.

`Mixed2` takes as input a mixed graph $G = (V, E, A)$, where E is the set of undirected edges and A is the set of directed edges, with $n = |V|$, $m = |E| + |A|$. Following a common strategy to attack the MCPP, the algorithm starts by inserting into the graph a multiset of *additional directed edges*, which are obtained either by duplicating a directed edge of G or by orienting a copy of an undirected edge of G:

1. Find the minimum-cost multiset set of additional directed edges of G, so as to obtain a multigraph (V, N, M) in which the in-degree and the out-degree of each vertex are equal.

2. Let $U \subseteq V$ be the set of odd-degree vertices in $G' = (V, N)$, and let D be the complete undirected graph on U where the weight of $\{u, v\}$ is the cost of the shortest path between u and v in the graph $G'' = (V, E)$.

3. (PM algorithm): compute a minimum-cost perfect matching P of D; for any edge $\{u, v\} \in P$, add to N edges forming the corresponding shortest path in G''.

4. Form an Eulerian multigraph (V, N, M).

5. Return an Eulerian tour of (V, N, M).

The problem addressed in step 1 can be formulated in terms of a network minimum-cost flow problem (see [Edmonds and Johnson 73]), in which the cost of the additional directed edges is minimized, as follows. We let $F = \{(u, v), (v, u) \mid \{u, v\} \in E\}$ contain a pair of opposed directed copies of each undirected edge; then we set $E_1 = E_2 = F$ and $E_t = E_1 \uplus E_2 \uplus A$, where $\uplus$ denotes the multiset union operator. For each $v \in V$ we let $\delta_v = \delta_A^-(v) - \delta_A^+(v)$ be the difference between the in-degree and the out-degree of the vertex.

Step 1 is implemented by algorithm `InOutDegree`, which proceeds as follows.

1. Solve the integer linear program (ILP) defined by

$$\min \mu(y) = \sum_{e \in A} b_e y_e + \sum_{e \in E_1} b_e y_e,$$

$$\sum_{\substack{e \in E_t \\ e \nwarrow v}} y_e - \sum_{\substack{e \in E_t \\ e \nearrow v}} y_e = \delta_v, \forall v \in V,$$

$$y_e \in \{0, 1\}\ \forall e \in E_2,$$

$$y_e \in \mathbb{N}\ \forall e \in E_1 \uplus A,$$

   where $e \nwarrow v$ (respectively $e \nearrow v$) means that e is directed away from (respectively toward) v.

2. Let $N = \varnothing$ and $M = A$.

3. For all $e_1, e_2 \in E_2$ such that $e_1 = (u, v)$, $e_2 = (v, u)$, if $y_{e_1} + y_{e_2} = 1$, then add y_{e_1} copies of e_1 and y_{e_2} to M; else add $\{u, v\}$ to N.

4. For all $e \in A$ add y_e copies of e to M.

5. For all $e \in E_1$ add y_e copies of e to M.

6. $x_1 = M \uplus N$.

We again take advantage of Theorem 3.2, which allows us to deduce the step-integral monotonicity of `Mixed2` after proving that `InOutDegree` is a stable SIM algorithm. The first step toward such a proof is the following lemma, giving an upper bound on the agents' workloads.

Lemma 6.1. *For each agent a_e, $w_e(x_1) \leq |A| + 1$.*

Proof. Let y^* be an optimal solution for the ILP. We will transform y^* in polynomial time into another optimal solution $\tilde{y}$ such that the corresponding solution x_1 computed by the algorithm satisfies the lemma. Notice that it suffices to prove that:

(i) $\tilde{y}_e \leq |A|, \forall e \in A$;

(ii) $\tilde{y}_{e'} + \tilde{y}_{e''} \leq |A|, \forall e \in E$, where $e', e'' \in E_1$ are the two corresponding oriented versions of e.

Start with $\tilde{y} = y^*$. Clearly the multigraph $G' = (V, F)$, where F is equal to A plus y^*_e copies of $e, \forall e \in E_t$, is made up of $G_1, \ldots, G_k$, $k \geq 1$, strong connected components. Moreover, since each vertex has its in-degree equal to its out-degree, each G_i, for $i = 1, \ldots, k$, admits an Eulerian tour.

It is easy to see that it suffices to prove the claim for one of the G_i. Hence we can assume without loss of generality that G' is strongly connected and that $\mathcal{E} = v_0 e_0 v_1 e_1 \ldots v_{h-1} e_{h-1} v_h$ is an Eulerian tour of G'. Let $\phi_X(e)$ denote the number of occurrences of the edges e in a multigraph X.

For any $e \in A$ define a Boolean variable ψ_e and set its initial value to 0. Now repeat the following until $\mathcal{E}$ does not contain any edge:

1. find the minimum integer j such that $v_i = v_j$ for some $i < j$;
2. let C be the cycle $v_i e_i v_{i+1} \ldots v_{j-1} e_{j-1} v_j$;
3. if $\forall k = i, \ldots, j-1$ such that $e_k \in A$, the corresponding variable ψ_{e_k} is equal to 1, then $\forall e \in E_t$ update $\tilde{y}_e = \tilde{y}_e - \phi_C(e)$;
4. $\forall k = i, \ldots, j-1$ such that $e_k \in A$ set y_{e_k} to 1;
5. delete C from $\mathcal{E}$.

Since C is a cycle, whenever $\tilde{y}$ is updated, it continues to be a feasible solution for the ILP, with $\mu(\tilde{y}) \leq \mu(y^*)$; hence it is optimal. Moreover, since the number of steps in which $\tilde{y}$ is not updated is at most $|A|$ (because there always exists $e \in A$ such that $\psi_e = 0$) and since C is a cycle, (i) and (ii) follow. □

We also need a tie-breaking rule for the optimal solutions of the ILP. One is provided by the following lemma.

Lemma 6.2. *It is possible to modify the objective function in order to have a unique optimal solution for the ILP. This implies the existence of a tie-breaking rule.*

Proof. Let $K = |A| + 2$, $m = |E_1 \uplus A|$, and let $\chi : E_1 \uplus A \longrightarrow \{0, \ldots, K-1\}$ be an injective function. Now let us consider the function $\widehat{\mu}$ obtained by replacing in μ each b_e with the value $\widehat{b_e} = b_e K^m + K^{\chi(e)}$. For any solution y of the ILP, $\widehat{\mu}(y)$ consists of the value $\mu(y)$ scaled by a factor K^m, plus a term less than K^m; hence suboptimal solutions are not also optimal minimizing $\widehat{\mu}$. Let y^* be an optimal solution for the ILP. Since for each e we have $y^*_e \leq |A| + 1$, as shown in the proof of Lemma 6.1, it is clear that $\widehat{\mu}(y^*) \bmod K^m$ is an encoding of y^* (in a base-K representation of $\widehat{\mu}(y^*) \bmod K^m$, the $\chi(e)$th digit is the value assumed by y^*_e). Therefore, we can replace the objective function μ by $\widehat{\mu}$ in the ILP and satisfy the claim. Since each b'_e needs $O(\log b_e K^m) = O(\log b_e + m \log K) = O(\log b_e + m \log m)$ binary digits to be represented, the replacement takes polynomial time. □

We are now ready to prove `InOutDegree`'s monotonicity.

Lemma 6.3. *Algorithm `InOutDegree` is a stable SIM algorithm.*

Proof. First of all, notice that after step 3 of $\mathcal{A}_1$, $M \uplus N$ contains exactly one copy of A and exactly one copy (either directed or undirected) of every $e \in E$. Hence, in order to prove monotonicity, it suffices to consider the workload deriving from variables y_e appearing in the objective function μ.

Concerning the step-integral monotonicity, let $w_e(x_1)$ be a_e's workload when a_e's bid is b_e. Suppose, for the sake of a contradiction, that for a new bid $b'_e = b_e + \Delta_e$, with $\Delta_e > 0$, we have $w_e(x'_1) > w_e(x_1)$, where x'_1 is the solution computed by `InOutDegree` for a bid profile $b' = (b_{-e}, b'_e)$. Let y_1 and y_2 be the optimal solutions for bid profiles b and b', respectively, and let μ_1 and μ_2 be the corresponding values of the objective function. Since tie-breaking rules guarantee a unique optimal solution, by evaluating y_1 for bid profile b', we get $\mu_2 < \mu_1 + \Delta_e(w_e(x_1) - 1)$. Then we have a contradiction, since the value of y_2 for the bid profile b is

$$\begin{aligned}\mu_2 - \Delta_e(w_e(x'_1) - 1) &< \mu_1 + \Delta_e(w_e(x_1) - 1) - \Delta_e(w_e(x'_1) - 1) \\ &< \mu_1 + \Delta_e(w_e(x_1) - w_e(x'_1)) < \mu_1.\end{aligned}$$

On the other hand, the stability follows from the tie-breaking rules. □

To compute payments, from Theorem 3.3 it suffices once again to find, for each edge e in x_1, the thresholds with respect to `InOutDegree`. We use the method described in Lemma 3.4, which allows us to compute thresholds $\theta_1, \ldots, \theta_h$ for `InOutDegree` in $O(h\,\mathcal{T}_{\texttt{InOutDegree}})$ time. The ILP used in `InOutDegree` models a minimum-cost flow problem. Then the time complexity of `Mixed2` is dominated by the $\mathcal{T}_{\mathrm{MCF}} = O(m^2 \log n + mn \log^2 n)$ time required to compute a minimum-cost flow (see [Schrijver 03]), and the $O(n^3)$ time required by the PM algorithm. To determine the time complexity of the computation of thresholds we use Theorem 3.5, noticing that Lemma 6.1 gives a polynomial bound on the value of h, and that in a feasible solution x, there are at most m agents for which, since $w_e(x) > 1$, it is necessary to compute payments.

We have proved the following theorem.

Theorem 6.4. *There exists an $O(m|A|(\mathcal{T}_{\mathrm{MCF}} + n^3))$-time 2-approximate truthful mechanism for the MCPP in which each edge is owned by a distinct selfish agent.*

7. Conclusions

In this paper we have developed a general composition technique between algorithms satisfying certain monotonicity properties that are relatively easy to check. The usefulness of our technique is twofold: on the one hand, it allows us to simplify the question of designing monotone algorithms, and on the other hand, it provides a way to compute efficiently the payments returned to the agents. Furthermore, our technique is practical, as witnessed by its application to a set of graph-traversal problems in an adversarial setting in which each agent owns a single edge of the underlying graph.

For future work, a first goal is that of closing the gap between the approximation of the canonical MCPP and that in which each edge is owned by a selfish agent. Moreover, putting our result into perspective, a major issue that needs to be addressed is to what extent, if any, our technique is amenable to additional boundary constraints on the problem (e.g., frugality, multiple-edge ownership). Finally, it would be interesting to extend the focus to other graph (traversal) problems for which the currently known solution algorithms can be naturally decomposed as our technique suggests. Potential candidates in this direction are, for instance, MST-based approximation algorithms for several graph connectivity problems.

Acknowledgments. This work was partially supported by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research. A preliminary version of this paper was presented at the *3rd International Workshop on Internet and Network Economics (WINE)*, San Diego, CA, USA, December 12–14, 2007.

References

[Archer and Tardos 01] A. Archer and E. Tardos. "Truthful Mechanisms for One-Parameter Agents." In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pp. 482–491. Los Alamitos, CA: IEEE Press, 2001.

[Böckenhauer et al. 02] H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. "Towards the Notion of Stability of Approximation for Hard Optimization Tasks and the Traveling Salesman Problem." *Theoretical Computer Science* 285:1 (2002), 3–24.

[Brotcorne et al. 01] L. Brotcorne, M. Labbé, P. Marcotte, and G. Savard. "A Bilevel Model for Toll Optimization on a Multicommodity Transportation Network." *Transportation Science* 35 (2001), 1–14.

[Christofides 76] N. Christofides. "Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem." Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[Clarke 71] E. H. Clarke. "Multipart Pricing of Public Goods." *Public Choice* 11:1 (1971), 17–33.

[Edmonds 65] J. Edmonds. "The Chinese Postman Problem." *Operations Research* 13 (1965), 73–77.

[Edmonds and Johnson 73] J. Edmonds and E. L. Johnson. "Matching, Euler Tours and the Chinese Postman." *Math. Programming* 5 (1973), 88–124.

[Frederickson 79] G. N. Frederickson. "Approximation Algorithms for Some Postman Problems." *J. ACM* 26:3 (1979), 538–554.

[Groves 73] T. Groves. "Incentive in Teams." *Econometrica* 41 (1973), 617–631.

[Gualà and Proietti 05a] L. Gualà and G. Proietti. "Efficient Truthful Mechanisms for the Single-Source Shortest Paths Tree Problem." In *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lisbon, Portugal, August 30–September 2, 2005, Proceedings*, Lecture Notes in Computer Science 3648, pp. 941–951. New York: Springer, 2005.

[Gualà and Proietti 05b] L. Gualà and G. Proietti. "A Truthful (2-2/k)-Approximation Mechanism for the Steiner Tree Problem with k Terminals." In *Computing and Combinatorics: 11th Annual International Conference, COCOON 2005, Kunming, China, August 16–19, 2005, Proceedings*, Lecture Notes in Computer Science 3595, pp. 90–400. New York: Springer, 2005.

[Hershberger and Suri 01] J. Hershberger and S. Suri. "Vickrey Prices and Shortest Paths: What Is an Edge Worth?" In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pp. 252–259. Los Alamitos, CA: IEEE Press, 2001.

[Kao et al. 05] M.-Y. Kao, X.-Y. Li, and W. Wang. "Towards Truthful Mechanisms for Binary Demand Games: A General Framework." In *Proceedings of the 6th ACM Conference on Electronic Commerce*, pp. 213–222. New York: ACM Press, 2005.

[Lenstra and Rinnooy Kan 76] J. K. Lenstra and A. G. H. Rinnooy Kan. "On General Routing Problems." *Networks* 6 (1976), 273–280.

[Nisan and Ronen 01] N. Nisan and A. Ronen. "Algorithmic Mechanism Design." *Games and Economic Behaviour* 35 (2001), 166–196.

[Papadimitriou 76] C. Papadimitriou. "On the Complexity of Edge Traversing." *J. ACM* 23:3 (1976), 544–554.

[Penna et al. 06] P. Penna, G. Proietti, and P. Widmayer. "Strongly Polynomial-Time Truthful Mechanisms in One Shot." In *Internet and Network Economics: Second International Workshop, WINE 2006, Patras, Greece, December 15–17, 2006, Proceedings*, Lecture Notes in Computer Science 4286, pp. 377–388. New York: Springer, 2006.

[Pettie and Ramachandran 02] S. Pettie and V. Ramachandran. "Computing Shortest Paths with Comparisons and Additions." In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 267–276. Philadelphia: SIAM, 2002.

[Proietti and Widmayer 05] G. Proietti and P. Widmayer. "A Truthful Mechanism for the Non-utilitarian Minimum Radius Spanning Tree Problem." In *Proceedings of the Seventeenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 195–202. New York: ACM Press, 2005.

[Raghavachari and Veerasamy 99] B. Raghavachari and J. Veerasamy. "A 3/2-Approximation Algorithm for the Mixed Postman Problem." *SIAM J. Discrete Math.* 12:4 (1999), 425–433.

[Schrijver 03] A. Schrijver. *Combinatorial Optimization—Polyhedra and Efficiency.* New York: Springer, 2003.

[Vickrey 61] W. Vickrey. "Counterspeculation, Auctions and Competitive Sealed Tenders." *J. of Finance* 16:1 (1961), 8–37.

Davide Bilò, Institut für Theoretische Informatik, ETH, Zurich, Universitätstr. 6, CH 8092 Zurich, Switzerland (davide.bilo@di.univaq.it)

Luca Forlizzi, Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy (forlizzi@di.univaq.it)

Luciano Gualà, Dipartimento di Matematica, Università di Tor Vergata, Via della Ricerca Scientifica 1 I-00133 Rome, Italy (guala@di.univaq.it)

Guido Proietti, Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, 67010 L'Aquila, Italy, and Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti," CNR, Viale Manzoni 30, 00185 Rome, Italy (proietti@di.univaq.it)

Received March 31, 2008; accepted January 30, 2009.