Achieving Allocatively-Efficient and Strongly Budget-Balanced Mechanisms in the Network Flow Domain for Bounded-Rational Agents

Yoram Bachrach and Jeffrey S. Rosenschein

School of Engineering and Computer Science
Hebrew University, Jerusalem, Israel
{yori, jeff}@cs.huji.ac.il
http://www.cs.huji.ac.il/

Abstract. Vickrey-Clarke-Groves (VCG) mechanisms are a well-known framework for finding a solution to a distributed optimization problem in systems of self-interested agents. VCG mechanisms have received wide attention in the AI community because they are efficient and strategy-proof; a special case of the Groves family of mechanisms, VCG mechanisms are the *only* direct-revelation mechanisms that are allocatively efficient and strategy-proof. Unfortunately, VCG mechanisms are only weakly budget-balanced.
We consider self-interested agents in a network flow domain, and show that in this domain, it *is* possible to design a mechanism that is both allocatively-efficient and almost completely budget-balanced. This is done by choosing a mechanism that is not *strategy-proof* but rather *strategy-resistant*. Instead of using the VCG mechanism, we propose a mechanism in which finding the most beneficial manipulation is an NP-complete problem, and the payments from the agents to the mechanism may be minimized as much as desired. This way, the mechanism is virtually strongly budget-balanced: for any $\epsilon > 0$, we find a mechanism that is ϵ-budget-balanced.

1 Introduction

Mechanisms face the problem of finding a system-wide solution to an optimization problem based on private information given by self-interested agents. As mechanism designers, we want to build a mechanism that would encourage agents to report their information truthfully, so that we can implement a desirable social choice function and maximize social welfare. A well-known solution to this problem in the case of quasi-linear preferences is that of Groves mechanisms. A special case of the Groves family of mechanisms are VCG mechanisms, which are budget-balanced, allocatively-efficient and strategy-proof.

Thus, in many cases, by using VCG we get a mechanism that operates with no outside subsidy (weakly budget-balanced) and maximizes the agents' utility. We maximize the agents' utility by choosing the outcome that maximizes the total utility according to the agents' reported types. Since VCG mechanisms are strategy-proof, the agents report their true preferences, and the mechanism maximizes their true utility. Also, VCG mechanisms are individually rational. Although in VCG mechanisms the agents pay the

mechanism, they never pay more than they value the chosen outcome; the agents always have positive utility, and voluntarily participate.

A significant disadvantage of VCG mechanisms is that they are only *weakly* budget-balanced. We would in principle prefer a *strongly* budget-balanced mechanism, where the total sum of payments to the mechanism is zero: $\sum_i t_i(\theta) = 0$. Impossibility results ([1] and [2]) show that in a quasi-linear environment, it is impossible to achieve a mechanism that is strategy-proof, allocatively-efficient, and strongly budget-balanced. Given this fact, we are faced with a grim future. Giving up allocation-efficiency means we would no longer maximize the sum of agents' utilities, which was our goal in the first place. Giving up strategy-proofness means agents may have an incentive to try and manipulate the mechanism by not reporting their true preferences, which may lead us to a sub-optimal result. However, without sacrificing one of the two, we will not be able to achieve strong budget-balance.

We propose addressing this problem by relaxing the strategy-proof requirement, replacing it with *strategy-resistance*. A mechanism is strategy-proof if the dominant strategy of each agent is to reveal its true type to the mechanism. Strategy-resistance only requires that even if an agent is given the reported types of the other agents, it still faces a computationally intractable problem to solve if it wishes to find a beneficial manipulation (i.e., report a false type to the mechanism in order to gain higher utility). We here consider a scenario in which it is an NP-hard problem for an agent to find a useful manipulation. NP-hardness is a worst case notion of computational difficulty, in the sense that it only indicates that a certain problem has *some* hard instances. A stronger notion of strategy-resistance could also require the manipulation problem to have no approximation methods, or require it to be in some harder computational complexity class. Also, a stronger sense of strategy resistance could require it to be hard to find *any* beneficial manipulation, not just the *optimal* manipulation. This paper constitutes a first step in establishing the notion of strategy-resistance.

In this paper we consider the network flows domain. In this domain, the edges of a network flow belong to several self-interested agents. Each agent reports its edges to the mechanism. The mechanism is then required to choose a flow from the source vertex to the target vertex. Agents gain utility from flow units on their edges. A reasonable choice for the mechanism would be selecting the flow that maximizes the total flow on all of the agents' edges. In the case of a layer graph, this can easily be done by finding the maximal flow, and we get a simple and tractable algorithm for implementing the mechanism, assuming each of the agents *truthfully* reports its subset of edges. However, in some cases it is beneficial for these agents to hide some of their edges. A VCG mechanism to overcome this problem would be strategy-proof, but only weakly budget-balanced.

We show that in the domain of network flows, it *is* possible to achieve a mechanism that is strategy-resistant (and thus agents have an incentive to be truthfull), efficient when agents are truthfull, and as budget-balanced as we want it to be (i.e., we can minimize the sum of agent payments, $\sum_i t_i(\theta')$, as much as we want). A mechanism is ϵ-budget-balanced if $0 \leq \sum_i t_i(\theta') < \epsilon$. We analyze a general multiagent flow problem, and show that for every $\epsilon > 0$ we can create a strategy-resistant, allocatively efficient, and ϵ-budget-balanced mechanism. This result indicates that at least for some

domains, it is possible to use the fact that agents have computational limitations and are not unboundedly rational, so as to construct mechanisms with beneficial properties.

2 Related Work

The main focus of research on bounded-rational mechanism design is on the problems that computational complexity poses for mechanism designers. This has led to attempts to use approximation algorithms for mechanisms whose standard implementation is intractable. It has also tended to highlight the advantages of strategy-proof mechanisms. In such mechanisms, an agent does not need to reason about what other agents might do, thus eliminating the need for speculation, counter-speculation, and so on.

Relatively little research has been dedicated to *using* the bounded-rationality of realistic agents to build better mechanisms, with more useful properties. This approach was taken in [3] (building on the work in [4]), which used computational complexity to show that common voting protocols are hard to manipulate. A similar approach was taken in [5], where coalition games were analyzed. It was shown there that manipulating a marginal-contribution based value distribution scheme, similar to the standard solution of the Shapley value [6], is an NP-complete problem. This indicates that such value distributions can be used even though they are manipulable, since agents would find it intractable to decide *how* to manipulate the mechanism that decides on the value distribution.

[7] considered coalitions among computationally bounded agents. It suggested some bounded rational concepts for coalition games, and indicated that computational complexity considerations may lead us to extend the set of acceptable stable solutions. This work argued that some coalitions are stable even if there exists a subset of the coalition that can do better by itself, since it is *intractable* for the agents in the large coalition to find such a subset.

[8] analyzed VCG auctions, and showed that manipulating VCG auctions using false name bids is NP-hard; it also analyzed approximate VCG auctions. [9] showed that using an approximation method to find the optimal allocation in combinatorial auctions can lead to the loss of strategy-proofness. However, in [8] it was shown that it is possible to take any tractable approximation algorithm and produce a hard-to-manipulate mechanism. In this case, the bounded rational nature of the *agents* has been used to show that an approximate mechanism is hard to manipulate, thus overcoming the computational complexity problems of the *mechanism*.

3 Preliminaries

In this article, we propose an alternative to VCG mechanisms in quasi-linear domains. In such domains, we have a set I of agents. The mechanism needs to choose one of a set of possible alternatives K. Each agent reports a type $\theta_i \in \Theta_i$ to the mechanism. This type represents the agent's preferences over the different alternatives in K. Each agent has a different valuation of the mechanism's chosen alternative $k \in K$, $v_i(k, \theta_i)$. The mechanism chooses the outcome according to a choice rule $k : \Theta_1 \times ... \times \Theta_I \rightarrow K$. Each agent is also required to make a payment p_i to the mechanism. The mechanism chooses

the payment of each agent according to a payment rule $t_i : \Theta_1 \times ... \times \Theta_I \rightarrow \mathbb{R}$. If the agents have quasi-linear utility functions, then the agents have utility $u_i(k, p_i, \theta_i) = v_i(k, \theta_i) - p_i$. An agent might not report its true type, but can choose a type to report to the mechanism. Thus, agent i (A_i) reports a type $\theta'_i = s_i(\theta_i)$, according to its own strategy.

3.1 Groves and VCG Mechanisms

In Groves mechanisms, the mechanism's choice rule given the reported types $\theta' = (\theta'_1, ..., \theta'_I)$ maximizes the sum of the agents' utilities, according to their reported types:

$$k^*(\theta') = arg\max_{k \in K} \sum_i v_i(k, \theta'_i)$$

The payment rule in Groves mechanisms is

$$t_i(\theta') = h_i(\theta'_{-i}) - \sum_{j \neq i} v_j(k^*, \theta'_j)$$

where $h_i : \Theta_{-i} \rightarrow \mathbb{R}$ may be any function that only depends on the reported types of agents other than i. Groves mechanisms are allocatively-efficient, and maximize the total utility of the agents. They are also strategy-proof, and for each agent the dominant strategy is to reveal its true type (or preferences) to the mechanism, no matter what the other agents report. Groves mechanisms are known to be the only direct revelation mechanisms that are allocatively-efficient and strategy-proof. Another advantage of Groves mechanisms is that in many cases they are weakly budget-balanced: $\sum_i t_i(\theta) \geq 0$.

A special case of Groves mechanisms is that of the VCG mechanism, when

$$h_i(\theta'_{-i}) = \sum_{j \neq i} v_j(k^*_{-i}(\theta'_{-i}), \theta'_j)$$

Under quite general settings, agents would voluntarily participate in VCG mechanisms, and we say that under these conditions the mechanism is individual-rational. The VCG mechanism also achieves weak budget-balance in quite general settings.

3.2 Main Contribution of the Paper

We approach the problem of designing a mechanism for bounded rational agents by building a mechanism for a distributed flow problem. We will demonstrate that for this domain, we can find a mechanism that is allocatively-efficient, ϵ-budget-balanced, and strategy-resistant. This means that if we assume the agents are bounded-rational and would not try to manipulate the mechanism if such manipulation is an NP-complete problem, they would all truthfully report their preferences. Once the mechanism gets their true preferences, it chooses the outcome that maximizes total utility of the agents. To achieve this truthfulness, the mechanism requires side payments; however, the total sum of these payments can be minimized as much as required. In other words, for every

$\epsilon > 0$ we can build such a strategy-resistant, allocatively-efficient mechanism, that would also be ϵ-budget-balanced.

We restrict ourselves to the case where maximizing the graph's flow also maximizes the agents' total utility, since this allows us to choose a tractable mechanism. However, although the mechanism itself performs a polynomial calculation, finding the optimal manipulation for an agent remains NP-complete. The payments we demand from the agents to the mechanism makes finding this manipulation hard, while leaving the mechanism's calculation simple and tractable.

The mechanism we suggest for the self-interested layered-graph network flow problem indicates that for some problems we can devise *tractable* allocatively-efficient, strategy-resistant, and ϵ-budget-balanced solutions. It remains an open problem to characterize the domains in which such a solution is achievable. Also, as explained above, this paper considers a domain in which finding a beneficial manipulation is NP-hard to be a strategy-resistant domain. It also remains an open problem to find domains in which we can achieve a stronger notion of strategy-resistance.

4 Self-Interested Network Flow

We now present the self-interested layered-graph network flow problem. Consider a flow network on a layered graph. We have a graph $G =< V, E >$, with source vertex s and target vertex t. The vertices of the graph are partitioned into $n+1$ layers, $L_0 = \{s\}, L_1, ..., L_n = \{t\}$. The edges only run between consecutive layers. We have a capacity function $c : E \rightarrow \mathbb{R}$ which is the maximal flow allowed on the edges. We also have a set I of agents. Each agent controls a subset $E_i \subset E$ of the graph's edges. No two agents control the same edge: $\forall_{i \neq j} E_i \cap E_j = \phi$.

The mechanism chooses a valid flow from s to t. A valid flow is a function $f : E \rightarrow R$ such that the following hold: $\forall_{(u,v)\in E} f(u,v) \leq c(u,v)$, $\forall_{(u,v)\in E} f(u,v) = -f(v,u)$, and $\forall_{u \in V-\{s,t\}} \sum_{v \in V} f(u,v) = 0$. We denote the positive flow as follows: if $f(u,v) > 0$ then $f^+(u,v) = f(u,v)$, otherwise $f^+(u,v) = 0$. We denote the size of the flow $|f| = \sum_{v \in V} f(s,v)$. The flow the mechanism chooses may only go through edges that belong to some agent. The mechanism knows the capacity constraints of the edges, but must treat edges not reported by an agent as edges whose capacity is 0. Each agent values the flow chosen by the mechanism according to the total flow going through its edges. Let f be the valid flow chosen by the mechanism, and E_f the set of edges in f through which there is a positive flow: $E_f = \{e \in E \mid f(e) > 0\}$. We denote the set of A_i's edges used in the flow f by: $E_{f,i} = E_f \cap E_i$. The agent's valuation of the flow is

$$v_i(f) = \sum_{e \in E_{f,i}} f(e)$$

A direct revelation implementation for the self-interested network flow problem would require each agent to state its valuation of all possible flows, which is not tractable. An alternative tractable implementation is to simply make the type of an agent the set of its declared edges $E_i' \in E_i$. Given this information, the mechanism could compute the agents' valuations of any possible flow. We will assume agents can only declare edges they actually own.

When the mechanism is given the agents' true types, $\theta = E_1, E_2, ..., E_I$, we want it to choose the flow that maximizes the total utility of the agents. The mechanism would be allocatively-efficient if it chooses

$$f^*(\theta) = arg\max_f \sum_i \sum_{e \in E_{f,i}} f(e)$$

4.1 Layered Graphs and Mechanisms for Network Flow

Consider a self-interested network flow problem in a *layered* graph. If each agent truthfully declares its subset of edges, the mechanism can easily compute $f^*(\theta)$ by running a maximal flow algorithm, such as the Edmonds-Karp algorithm.

Proof. Suppose the mechanism chooses a flow f. The total flow exiting s ends up in vertices in L_1. All the flow from L_1 ends up in vertices in L_2, and so on. Since flow may only go through edges owned by some agent, the total utility obtained by the flow f is

$$\sum_{e \in E} f^+(e) = \sum_{u \in L_1, v \in L_2} f^+(u,v) + ... + \sum_{u \in L_{n-1}, v \in L_n} f^+(u,v) = |f| + ... + |f| = (n-1)|f|$$

A naive mechanism for the self-interested flow problem, with no payments to the mechanism, is not strategy-proof. An agent may declare only a subset of the edges it controls, to change the flow that the mechanism chooses to a flow that the agent values more. Figure 1 shows two agents on a certain network flow (A_1 and A_2). A_1's edges are marked with dashed lines, and A_2's edges are marked with full lines. A_2 truthfully declares all its edges. Assuming the mechanism favors A_1 and chooses the specific maximal that maximizes A_1's utility among all maximal flows, A_1 can do better by not declaring its topmost edge (v_1, v_4), gaining a utility of 2 instead of 1.

5 A Mechanism for the Self-Interested Network Flow Problem

We assume quasi-linear utility. Each agent pays the mechanism a payment p_i, and its utility is $u_i(f) = v_i(f) - p_i$. We now show that by using a straightforward payment rule, we make finding a beneficial manipulation NP-hard. The payment rule we use is simple: each agent pays the mechanism a constant of c for each edge it declares it owns. Let $E'_i \subset E_i$ be the subset of edges an agent declares it owns. Then $p_i(E'_i) = c|E'_i|$. To make sure the mechanism is individual rational, the payment p_i should give the agent a utility of 0 when the agent's valuation of the given flow is less than $c|E'_i|$. Thus, the payment rule is:

$$p_i = \begin{cases} c|E'_i| & \text{if } v_i(f^*) > c|E'_i|; \\ v_i(f^*) & \text{otherwise;} \end{cases}$$

Assume that A_i knows E'_j for all $j \neq i$. It can easily calculate the utility it would get by truthfully declaring all its edges. How hard is it for i to find a subset of edges it could declare to the mechanism so as to gain a higher utility? First, note that the

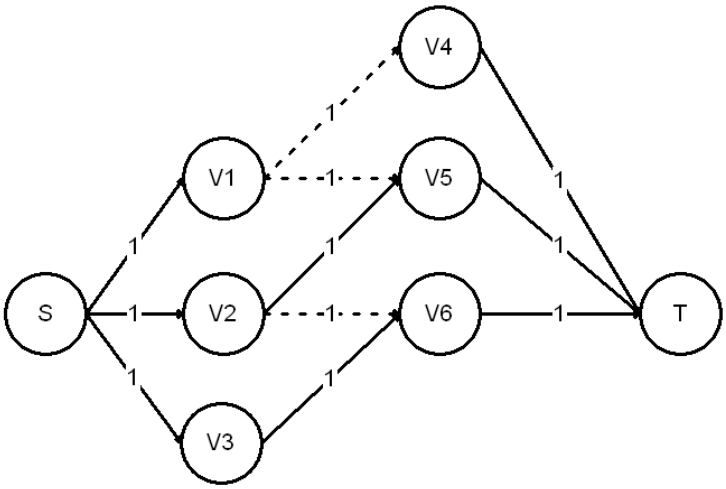

Fig. 1. Manipulations in self-interested flow problems

question itself is under-constrained. Even given E'_j for all j, including i, there may be several maximal flows; the mechanism is free to choose any of them. However, we will show that even if A_i can decide *which* maximal flow the mechanism chooses, it would still remain an NP-hard problem for that agent to find a better subset of edges.

We now define the problem of *finding* the optimal manipulation in the self-interested network flow domain.

FLOW-EDGE-SUBSET: We are given a layered graph flow network, with the capacity function $c : E \to \mathbb{R}$, E_{-i} the declared edges of the other agents, and E_i, the set of our edges. We are also given the constant c of the payment, and we know that if we declare that we have k edges, our payment to the mechanism would be $p_i = ck$. We assume the mechanism prefers a maximal flow that maximizes *our* utility: if we report a subset of edges $E'_i \subset E_i$ the mechanism would choose the maximal flow f^* to be the flow that maximizes $\sum_{e \in E_{f^*,i'}} f(e)$ from among all the possible maximal flows. We are also given a constant k, the target utility for A_i. We are asked if there is a subset of A_i's edges $E'_i \subset E_i$, such that the maximal flow chosen by the mechanism, $f^*(E_1, ..., E_{i-1}, E'_i, E_{i+1}, ..., E_I)$ gives A_i a utility of at least k:

$$u_i(f^*, p_i) = v_i(f^*) - p_i = \sum_{e \in E_{f^*,i}} f(e) - c|E'_i| \geq k$$

5.1 NP-Completeness of FLOW-EDGE-SUBSET

First, we note that FLOW-EDGE-SUBSET is in NP, because given a subset of edges $E'_i \subset E_i$ we can easily compute the maximal flow. We show that FLOW-EDGE-SUBSET is NP-complete by reducing a general VERTEX-COVER problem to a FLOW-EDGE-SUBSET problem. The reduction shows that FLOW-EDGE-SUBSET is NP-

complete even if the inputs are restricted to problems where there are only two agents, and the graph has only 5 layers.

VERTEX-COVER: We are given a graph $G =< V, E >$ and a constant n and are asked if there is a subset of n vertices $V' \subset V, |V'| = n$ that covers all the edges $\forall_{(u,v)\in E}$ either $u \in V'$ or $v \in V'$.

The reduction is done as follows. From the VERTEX-COVER input, we build inputs for the FLOW-EDGE-SUBSET problem. Given the original VERTEX-COVER graph G, we build a layer graph G'. All the inputs to FLOW-EDGE-SUBSET are built with this layer graph G', and in all of them there are two agents, and we are asked about the utility of A_1. In all of these inputs we have the same set of A_1's edges E_1, the same list of declared edges of the other agent, and the same payment constant, c. This payment constant is chosen such that the payment from A_1 to the mechanism is always less than 1, even if A_1 declares all its edges.

The only difference between the inputs is the target utility k. These inputs are constructed such that A_1 has $|V|$ edges, where $|V|$ is the number of vertices in G. The inputs are constructed so that the maximal utility A_1 can achieve is obtained by declaring some set of edges, E_1^*, and in this case, A_1's utility is $u_1(E_1^*) = v_1(E_1^*) - p_1(E_1^*) = |E| - c|E_1^*|$, where $|E|$ is the number of edges in the VERTEX-COVER graph G, and $|E_1^*|$ is the number of vertices in the *minimal vertex-cover* of G. We abuse notation a bit here, and denote $u_1(E_1')$ and $v_1(E_1')$ as the utility and valuation A_1 has when declaring the E_1' subset of edges, since the declared edges of all the other agents are known. Thus the flow chosen by the mechanism only depends on A_1's chosen subset of edges, E_1'.

The Process of the Reduction Since the payment from A_1 to the mechanism is always a multiple of c, we can easily check how many vertices are used in the minimal vertex-cover of G. We construct G' from G, and use FLOW-EDGE-SUBSET to check if we can achieve a utility of at least $|E| - |V|c$, then check the possibility of achieving $|E| - (|V| - 1)c$, then $|E| - (|V| - 2)c$, and so on. The answer would initially be 'yes', since due to the construction, A_1 can achieve a utility of $|E| - |V|c$ by declaring all its edges. A_1 can decide to declare any number of edges between 0 and $|V|$. The questions are asked regarding higher and higher requested utilities, so eventually, for some $x \in \mathbb{N}, 0 \leq x \leq |V|$, the answer for $|E| - xc$ would be 'no'. We would then know the best utility that A_1 can achieve is $|E| - (x+1)c$, and thus the minimal vertex-cover of G is of size $x + 1$. This process involves running FLOW-EDGE-SUBSET $|V|$ times, so if FLOW-EDGE-SUBSET can be done in polynomial time, then this process can also be performed in polynomial time.

Constructing the FLOW-EDGE-SUBSET Inputs We now describe how the inputs for FLOW-EDGE-SUBSET are constructed from the VERTEX-COVER input. We build a 5-layer network flow graph, G'. The L_0 layer contains the single source vertex s, and the L_4 layer contains the single target vertex t. Layer L_1 contains a vertex v_{e_i} for each edge $e_i \in E$ in the original VERTEX-COVER graph. Layer L_2 contains a vertex $v_{i,b}$ for each vertex $v_i \in V$ in the original VERTEX-COVER graph. Layer L_3 contains a single vertex $v_{i,a}$ for each vertex $v_i \in V$ in the original VERTEX-COVER graph.

The edges between the layers are constructed as follows. The source vertex s is connected to all the vertices in L_1, and then we mark the edge (s, v_{e_i}) as e_{e_i}. Every vertex v_{e_i} in L_1 is connected to exactly two vertices in L_2. If edge $e_i \in E$ in G connects vertices v_i and v_j in it, then v_{e_i} is connected to $v_{v_i,b}$ and $v_{v_j,b}$ in the constructed graph G'. Every vertex in L_2 is connected to exactly one vertex in L_3. Vertex $v_{v_i,b}$ in L_2 is connected to $v_{v_i,a}$ in L_3. All the vertices in L_3 are connected to the sink vertex t in L_4. All the edges between L_0 and L_1 and all the edges between L_1 and L_2 have a capacity of 1. All the other edges have capacity of $|E|$.

As explained above, all inputs for FLOW-EDGE-SUBSET are given with regard to A_1. The set of A_1's edges E_1 is $(v_{v_i,b}, v_{v_i,a})$, for all possible i. All of the other edges belong to A_2, and in the input given to FLOW-EDGE-SUBSET, A_2 declares all its edges. The payment constant c is chosen such that $c < \frac{1}{|V|+|E|}$. We demonstrate building the layer graph in Figure 2. The graph on the left of Figure 2 is the input for the VERTEX-COVER, while the graph on the right is the generated FLOW-EDGE-SUBSET layer graph.

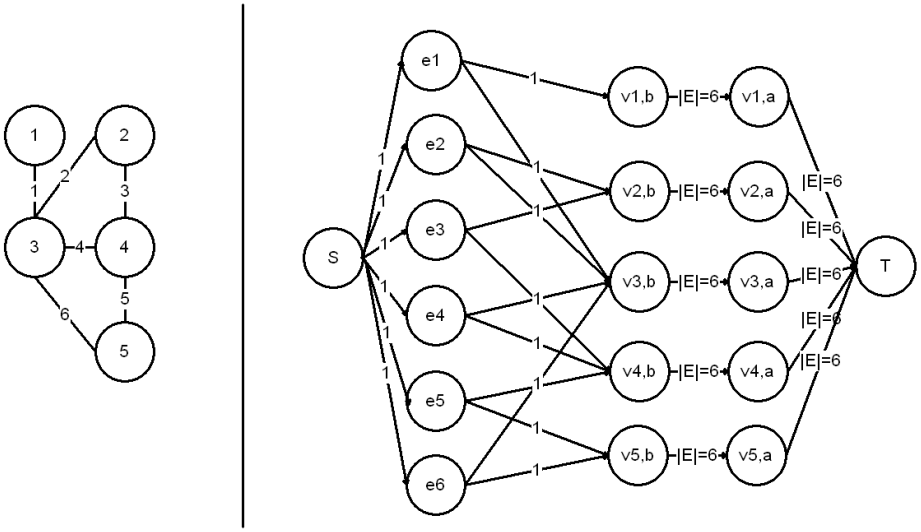

Fig. 2. Reducing VERTEX-COVER to FLOW-EDGE-SUBSET

The intuition behind this construction is simple. A_1's edges in the constructed graph represent vertices in the original graph G. A_1 must choose a subset of edges to report to the mechanism. Let $E'_1 \subset E_1$ be the subset of edges that A_1 chooses to declare. Each such choice can also be seen as a choice of a subset of vertices in G. These vertices cover certain edges in the original graph. We later refer to these edges as EC_{G,E'_1}. The v_{e_i} vertices in L_1 represent the edges in G.

The construction makes sure that the mechanism can only send flow from s to v_{e_i} if e_i is an edge covered by E'_1 ($e_i \in EC_{G,E'_1}$). In fact, a flow going through one of A_1's edges $(v_{v_i,b}, v_{v_i,a})$ can only originate from a v_{e_j} vertex that represents an edge e_j that covers v_i.

Thus A_1's valuation is limited by the number of edges covered by his chosen set of vertices, or in other words by $|EC_{G,E'_1}|$. Therefor, A_1 would choose E'_1 to be edges representing a set of vertices which covers all the edges in G; it would choose E'_1 such that $E = EC_{G,E'_1}$.

However, A_1 also sees that payments it must give the mechanism for declaring his edges. Since he pays a constant c per each edge he declares, A_1 would want to minimize the number of edges he declares. This conflicts with A_1 wish to choose E'_1 so that $E = EC_{G,E'_1}$, since fewer vertices cover fewer edges.

By choosing a low enough payment constant c, we make sure that A_1's top priority is to cover all the edges in G. It is only his second priority to minimize the number if edges he declares. Thus, A_1 actually wishes to choose E'_1 so that the set of vertices E'_1 represent is the minimal vertex cover of G.

The following sections formally prove the intuitive claims above.

Properties of the Constructed Inputs

Lemma 1 *If G had $|E|$ edges, then in the generated network flow graph, A_1's valuation cannot exceed $|E|$. If A_1 can get a valuation of $|E|$, its utility is in the range $|E| - 1 \leq u_1(f) \leq |E|$.*

Proof. The maximal flow cannot exceed $|E|$, because there are only $|E|$ edges between L_0 and L_1, each with a capacity of 1. All of A_1's edges are between L_2 and L_3, so the maximal flow through them also cannot exceed $|E|$. Thus, A_1's valuation of any flow f, $v_1(f)$, cannot exceed $|E|$. A_1's utility when a flow f is chosen is $u_1(f, p_1) = v_1(f) - p_1 = v_1(f) - c|E'_1|$. Due to the choice of c, $p_1 = |E'_1| \cdot \frac{1}{|V|+|E|} < 1$, so $0 \leq p_i \leq 1$.

Let $E'_1 \subset E_1$ be the subset of edges that A_1 chooses to declare. We denote $EC_{G,E'_1} = \{e_i \in E \mid e_i = (v_x, v_y)$ and at least one of the following holds:
$(v_{v_x,b}, v_{v_x,a}) \in E'_1$ or $(v_{v_y,b}, v_{v_y,a}) \in E'_1\}$. Similarly, we denote $EC_{G',E'_1} = \{e_{e_i} \mid e_i \in EC_G\}$. Intuitively, we identify the edge e_{e_i} with the edge e_i in the original graph. We identify the edge $(v_{v_x,b}, v_{v_x,a}) \in E_1$ with *vertex* v_x in the original graph G, and a subset of edges $E'_1 \subset E_1$ with a subset of vertices $V_{E'_1} \subset V$ in G. Such a set of vertices in G covers a subset of the edges in it. EC_G is the set of edges covered by these vertices $V_{E'_1}$, and $EC_{G'}$ is the set of edges in G' corresponding to the covered edges.

Lemma 2 *Let $E^*_1 \subset E_1$ be an optimal choice of edges for A_1 to declare. Then $EC_{G,E^*_1} = E$.*

Proof. Let E^*_1 be an optimal choice of A_1's edges, f be the flow chosen by the mechanism in this case, and EC_{G,E^*_1} be the subset of E, as explained above. Assume by negation that there *is* some $e = (x, y) \in E$ that $e \notin EC_{G,E^*_1}$. There cannot be any flow on e_e in G', since having such a flow $f(s, v_e) > 0$ would require having

either flow $f(v_e, v_{x,b}) > 0$ or $f(v_e, v_{y,b}) > 0$, since v_e is connected only to these two vertices. However, this would require having either a flow $f(v_{x,b}, v_{x,a}) > 0$ or $f(v_{y,b}, v_{y,a}) > 0$, both of which cannot occur, since $e \notin EC_{G,E_1^*}$, and $(v_{x,b}, v_{x,a}) \notin E_1^*$, and $(v_{y,b}, v_{y,a}) \notin E_1^*$. Thus, there cannot be any flow on e_e, and the maximal flow between L_0 and L_1 cannot exceed $|E| - 1$.

Suppose we add to E_1^* the edge $(v_{x,b}, v_{x,a})$. This would allow the mechanism to increase the flow in the following path by exactly 1: $s, v_e, v_{x,b}, v_{x,a}, t$, resulting in a flow f'. The flow through A_1's edges is exactly the same as before, except there is now a flow of 1 through $(v_{x,b}, v_{x,a})$, so $v_1(f') = v_1(f) + 1$. Since the payment to the mechanism is always less than 1, the total utility of A_1 has increased, so E_1^* was not an optimal choice for A_1 to begin with.

Lemma 3 *If A_1 declares $E_1' \subset E_1$ such that $EC_{G,E_1'} = E$, the mechanism can choose a flow f such that $|f| = |E|$.*

Proof. We can fill the capacities of all the edges between L_0 and L_1, having the vertices of L_1 with a total incoming flow of $|E|$. Since $EC_{G,E_1'} = E$, we also have $EC_{G',E_1'} = \{e_{e_i} \mid e_i \in E\}$, and every vertex v_{e_i} in L_1 is connected to at least one vertex $v_{x,b}$ in layer L_2 that is connected (in turn) to $v_{x,a}$ in layer L_3 by an edge $e \in E_1'$. We choose the flow $f(v_{e_i}, v_{x,b}) = 1$. We then continue the flow by sending the incoming flow of vertex $v_{x,b}$ to $v_{x,a}$, by choosing

$$f(v_{x,b}, v_{x,a}) = \sum_{v_{e_i} \in L_1} f(v_{e_i}, v_{x,b})$$

We can do this since the capacity of the edges between L_2 and L_3 is $|E|$, so $c(v_{x,b}, v_{x,a}) = |E|$, and there is no danger of having a flow coming into a vertex $v_{x,b}$ higher than the total capacity of its outgoing edges. The flow is then continued by sending all the incoming flow of vertex $v_{x,a}$ to the target vertex: $f(v_{x,a}, t) = f(v_{x,b}, v_{x,a})$. Again, this is possible since the capacity of the edges between L_3 and L_4 is $|E|$.

Therefore, the optimal subset of edges that A_1 can declare, $E_1^* \subset E_1$, allows the mechanism to achieve the maximal possible flow f^*, of size $|f^*| = |E|$.

Lemma 4 *The optimal subset of edges for A_1, $E_1^* \subset E_1$, gives A_1 a valuation of $v_1(f^*) = |E|$.*

Proof. From Lemma 2 and Lemma 3 we know that if A_1 declares E_1' in the optimal solution, the mechanism can choose a flow f such that $|f| = |E|$. By Lemma 1 this is a maximal flow that maximizes the utility of A_1. Since A_1 controls all the edges between L_2 and L_3 and has no other edges, we have a total flow of $|E|$ through A_1's edges.

As a private case of Lemma 4, we get that A_1 can get a valuation of $|E|$ by declaring all of its edges, since $EC_{G,E_1} = E$. This case gives A_1 a utility of $|E| - c|E_1|$, since we have declared $|E_1|$ edges.

Lemma 4 shows that the optimal subset of edges for A_1, $E_1^* \subset E_1$, gives A_1 a valuation of $v_1(f^*) = |E|$. However, to calculate A_1's utility in this case, we must also know the payment that A_1 gives the mechanism. This payment only depends on the number of edges in E_1^*.

Proof of the Reduction We now prove the validity of the reduction, by showing that the the maximal utility A_1 can achieve in the constructed network flow graph is determined by the size of the minimal vertex cover in the original graph.

Theorem 5 *The size of the minimal vertex-cover of G is k if and only if the maximal utility of A_1 in the constructed inputs to FLOW-EDGE-SUBSET is $|E| - kc$.*

Proof. Assume the maximal utility A_1 can achieve is $|E| - kc$. Due to Lemma 4, in order to obtain this optimal utility A_1 has to declare $E_1' \subset E_1$, a subset of edges with size $|E_1'| = k$. Consider the set $V_{E_1'} = \{v_x \in V \mid (v_{x,a}, v_{x,b}) \in E_1'\}$. From Lemma 2 we have $EC_{G,E_1'} = E$, so this set is a vertex-cover of G. Its size is $|V_{E_1'}| = k$, since the payment A_1 made to the mechanism is kc. Assume by negation that this is not the *minimal* vertex-cover of G. Then there exists a vertex cover VC' with a smaller size of $|VC'| = k'$. Consider $E_{VC'} = \{(v_{x,b}, v_{x,a}) \in E_1 \mid v_x \in VC'\}$. This is a subset of A_1's edges that (by definition of $EC_{G,X}$), $EC_{G,E_{VC'}} = E$. Thus, $v_1(E_{VC'}) = |E|$. However, since $|VC'| = k' < k = |V_{E_1'}|$, the payment from A_1 to the mechanism for declaring $E_{VC'}$ is only $p_1(E_{VC'}) = k' < k$. Thus the utility of A_1 when using $E_{VC'}$ is $u_1(E_{VC'}) = v_1(E_{VC'}) - p_1(E_{VC'}) = |E| - k'c > |E| - kc = u_1(E_1')$, and we would have a subset of edges giving a better utility than the optimal solution.

On the other hand, if we have a vertex-cover VC for G with size $|VC| = k$, consider $E_{VC} = \{(v_{x,b}, v_{x,a}) \in E_1 \mid v_x \in VC\}$. Again, this is a subset of A_1's edges that (by definition of $EC_{G,X}$) makes $EC_{G,E_{VC}} = E$. Thus, $v_1(E_{VC'}) = |E|$. The utility of A_1 when using E_{VC} is $u_1(E_{VC}) = v_1(E_{VC}) - p_1(E_{VC}) = |E| - kc$, so a utility of $|E| - kc$ is achievable.

It remains to show that this is the *maximal* utility achievable. Suppose, by negation, that we have a choice of edges $E_1' \subset E_1$ that gives A_1 a higher utility. The valuation of E_1' must also be $v_1(E_1') = |E|$, since a higher valuation is not possible, and a lower valuation would result in a utility that is below $u_1(E_{VC})$ (since the payment to the mechanism is less than 1). This means that the payment for E_1' is less than the payment for E_{VC}, or $|E_1'| < |E_{VC}|$. As explained above, in order to achieve a utility of $|E|$, E_1' must be a set such that $EC_{G,E_1'} = E$, so $V_{E_1'} = \{v_x \in V \mid (v_{x,b}, v_{x,a}) \in E_1'\}$ must be a vertex-cover of G. However, since $|V_{E_1'}| = |E_1'| < |E_{VC}|$, this would be a vertex-cover of a size smaller than the size of the minimal vertex-cover.

Due to Theorem 5, the process of the reduction as described above is valid, and FLOW-EDGE-SUBSET is NP-complete.

6 Conclusions and Future Directions

We have presented a mechanism for the distributed network flow problem with self-interested agents. With a proper choice of the payment constant c, finding a beneficial manipulation is an NP-complete problem. If most instances of the manipulation problem are indeed computationally intractable, we expect agents would truthfully report their preferences. In this case, the mechanism would choose the result maximizing the sum of agents' utilities, and we have an allocatively-efficient mechanism.

Given some $\epsilon > 0$, we can make the mechanism ϵ-budget-balanced, by choosing a constant $c = \frac{\epsilon}{n(|E|+|V|)}$, so that all of the agents together pay less than ϵ. The mechanism we have described is also individual rational. The mechanism's calculation is tractable, and only involves a polynomial algorithm for finding the maximal network flow. This indicates that the agents' difficulty in finding a beneficial manipulation is not caused by any difficulty in simulating the mechanism, but is instead caused by the difficulty of trying exponentially many options of untruthful declarations to the mechanism.

Therefore, in the domain of network flows, it is possible to achieve an individually-rational, allocatively-efficient, ϵ-budget-balanced, and strategy-resistant mechanism. The standard VCG solution in this domain would be only weakly budget-balanced, but strategy-proof. Impossibility results [1] and [2] indicate that no direct-revelation mechanism can achieve strong budget-balance without sacrificing either allocative-efficiency or strategy-proofness. We believe that in many cases, trading strategy-proofness for strategy-resistance is a fair price to pay for achieving strong budget-balance.

There has been much work dedicated to overcoming the intractability of mechanisms, since in building a real-world mechanism we cannot assume unbounded computation. However, if we are not willing to accept unbounded-rationality on the *mechanism's* part, we must also consider the implications of the bounded-rational nature of the *agents*.

We believe that strategy-proofness should not be the only criteria when considering the susceptibility of a mechanism to manipulations. In fact, we believe it is found on one end of a *scale* of susceptibility. On the other end of this scale are mechanisms where there exists a poly-time algorithm for finding the optimal manipulation. Such mechanisms probably cannot be used in practice, since they are so easy to manipulate. Between these two extremes is the region of strategy-resistant mechanisms. In this paper we have implicitly defined a strategy-resistant mechanism as one in which it is NP-hard to find the optimal manipulation. As we have commented above, this is a rather weak notion of strategy resistance. A preferable solution would be one in which it is computationally intractable to find *any* manipulation. NP-hardness is not sufficient for a problem to be computationally intractable. For example, we can require the manipulation problem to have no approximation methods, or show that most instances of the manipulation problem are indeed hard.

In this paper we have shown that in the network flow domain, we can gain budget-balance by giving up strategy proofness, and replacing it with our notion of strategy-resistance. Assuming we are willing to accept strategy-resistance as a sufficient guarantee that agents would truthfully declare their types, we have improved the results obtained by VCG for this problem.

We have chosen the self-interested network flow domain, because in this case we were able to find a mechanism that was tractable in its computational properties, and also had good results in the sense of being budget-balanced. This domain demonstrates that by using a very simple payment scheme we can create a significant gap between the amount of work the mechanism performs (in this case a simple poly-time algorithm) and the amount of work an agent is required to perform in order to find a beneficial manipulation (in this case solving an NP-hard problem). We believe further research can find domains in which the mechanism is required to perform harder work (e.g., solving an

NP-hard problem by approximation), and manipulations are completely intractable. It may be possible to achieve budget-balance while retaining a stronger notion of strategy-resistance, even in this domain. Also, it may be possible to find other valuable trade offs in other domains. It remains an open problem to characterize the domains in which using computational complexity in this way is possible, and to find domains in which a stronger sense of strategy-resistance can be achieved.

References

1. Green, J., Laffont, J.J.: Characterization of satisfactory mechanisms for the revelation of preferences for public goods. Econometrica **45** (1977) 427–38
2. Hurwicz, L.: On the existence of allocation systems whose manipulative Nash equilibria are pareto-optimal. In: 3rd World Congress of the Econometric Society (Unpublished). (1975)
3. Conitzer, V., Sandholm, T.: Universal voting protocol tweaks to make manipulation hard. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico (2003)
4. Bartholdi, J.J.: The computational difficulty of manipulating an election. Social Choice and Welfare **6** (1989) 227–241
5. Conitzer, V., Sandholm, T.: Computing shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI), San Jose, California, USA (2004) 219–225
6. Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games (1953) 31–40
7. Sandholm, T., Lesser, V.R.: Coalitions among computationally bounded agents. Artificial Intelligence **94** (1997) 99–137
8. Sanghvi, S., Parkes, D.C.: Hard-to-manipulate combinatorial auctions. Technical report, Harvard University (2004)
9. Lavi, R., Mu'alem, A., Nisan, N.: Towards a characterization of truthful combinatorial auctions (extended abstract). In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS). (2003)