Published in final edited form as:
J Comb Optim. 2010 August 1; 20(2): 180–193. doi:10.1007/s10878-008-9202-9.

Algorithms and Time Complexity of the Request-Service Problem

Chunmei Liu[*], **Legand Burge**[*], and **Ajoni Blake**[*]
Chunmei Liu: chunmei@scs.howard.edu; Legand Burge: blegand@scs.howard.edu; Ajoni Blake: ajonib@gmail.com
[*]Department of Systems and Computer Science, Howard University, Washington, DC 20059, USA

Abstract

Given a number of users each of which provides a set of services with a cost for each service and has a set of requests to be satisfied, the goal of the request-service problem is to find a feasible solution that satisfies all requests of each user with minimum cost. In addition, a feasible solution must satisfy an additional constraint. Specifically, if user A provides a service to user B, B should provide a service back to A either directly or indirectly through other users. In this paper, we studied the complexity of this problem. We show that there exists a polynomial time algorithm that can compute a feasible solution with minimum cost if such a solution exists. However, if a feasible solution does not exist, the problem of maximizing the number of satisfied users (i.e., all requests of the users are satisfied) is NP-hard.

Keywords

Request-service; Disjoint cycle cover; MAX2SAT

1 Introduction

The goal of request-service problem is to find the minimum cost to provide services for a group of users such that each user's requests are satisfied and his services are returned to himself. The simple version of this problem is that each user has only one service and one request while the general version is that each users has multiple services to provide and multiple requests to be satisfied. The problem has extensive applications in research related to barter economy, electrical marketing, and operating systems. However, it remains unknown whether the problem can be solved with a polynomial time algorithm.

There are similar problems that have been extensively studied. For example, barter exchange problem [7, 12, 13], specifically kidney exchange problem [1, 4, 6, 14], and fair sharing of peer-to-peer resources problem [2, 3, 5, 10], etc. In particular, the kidney exchange problem is that people who need to have kidney transplanted but have incompatible donors want to obtain compatible kidneys by exchanging their donors. The problem is the same as the simple version of our request-service problem. Abraham et. al. [1] studied the complexity of the kidney exchange problem with an added restriction on the lengths of exchange cycles. They showed that the kidney exchange problem is NP-hard if the lengths of the exchange cycles are at most L, $L \geq 3$, and developed an integer linear programming algorithm for the problem [1]. The fair sharing of peer-to-peer resources problem is similar to our general version of the request-service problem with a slight difference that in the fair sharing of peer-to-peer resources problem, some users may not provide its resources to the system if they get free resources from the system [10]. In this paper, we first show that the simple version of the request-service problem can be reduced to finding the minimum cost vertex disjoint cycle cover in a directed graph. We then show that the general version of the problem can be reduced to finding the minimum cost strongly connected component cover in a directed graph *G*, which can be further reduced to finding the minimum cost vertex disjoint cycle cover in another directed graph *H* constructed from *G*.

A vertex disjoint cycle cover (the size of each cycle is greater than or equal to 2) in a directed graph corresponds to a perfect matching in a bipartite graph. The cycle cover with the minimum weight can thus be found by computing the minimum weighted perfect matching in the bipartite graph. This matching can be found with the Hungarian Method [8, 9] in time $O(n^3)$, where n is the number of vertices in the graph. Based on this fact, we show that both the simple and general case request-service problems can be solved in polynomial time. In particular, we show the simplified version of the problem for N users can be solved in time $O(N^3)$ and the general version of the problem can be solved in time $O(N^{12})$. In cases where a valid solution does not exist, it might be more desirable to minimize the cost while serving as many users as possible. For the simple version of the problem, because each user has only one request, the complexity of the problem of maximizing the number of satisfied users is the same. We show that the problem of maximizing the number of satisfied users is NP-hard through a reduction from MAX2SAT, which is a well-known NP-hard problem [11]. The rest of the paper is organized as follows. In section 2, we present the notations and preliminaries needed for the paper. In sections 3 and 4, we present the solutions for both the simple and general version of the request-service problem and the proof of the correctness of the algorithms. In section 5, we show the NP-hardness of the problem maximizing the number of satisfied users through a reduction constructed from MAX2SAT. To simplify the notation, we use *cycle cover* to refer to vertex disjoint cycle cover in the remaining part of the paper. We conclude the paper in section 6.

2 Preliminaries

The graphs in this paper are directed graphs. Given a graph $G = (V, E)$, where V is the set of vertices in G and E is the set of directed edges. Given a directed edge $(u, v) \in E$, u is the *tail* of the edge and v is the *head* of the edge. we associate a weight value with each edge $(u, v) \in E$ and we use $w(u, v)$ to denote it. A *directed cycle* in G is a sequence of vertices $v_1, v_2, \cdots, v_l$ in V such that (v_i, v_{i+1}) is a directed edge in G for $1 \le i \le l$(here $i + 1$ is 1 if $i = l$). A *cycle cover* of G is a set of directed cycles in G such that each vertex in V is contained in one and only one directed cycle in the set. The *weight* of a cycle cover is the sum of weights of all edges in the cycle cover.

3 Finding a Minimum Weighted Vertex Disjoint Cycle Cover in a Weighted Directed Graph

Given a directed graph $G = (V, E)$ and the weight $w(u, v)$ of each directed edge (u, v) of G, the problem is to find in G a set of disjoint cycles such that each vertex appears in exactly one of the cycles and the total weight of edges in the cycles is the minimum. A well known fact is that the problem can be reduced to computing the maximum weighted perfect matching in a bipartite graph. The bipartite graph H can be constructed from G as follows. The vertex set of H is composed of two sets U and T such that $|U| = |T| = |V|$. We label each vertex in U and T with integers from 1 to $|V|$. Vertex $u \in U$ is connected to vertex $v \in T$ if (u, v) is a directed edge in G. In addition, edge (u, v) in H is associated with a weight $W - w(u, v)$, where W is a number larger than the maximum of all edges weights in G. It is not difficult to see that a minimum weighted cycle cover corresponds to a maximum weighted perfect matching in H. This matching can be efficiently computed with Hungarian Method[8] in time $O(n^3)$, where n is the number of vertices in the directed graph G.

For the simple version of the request-service problem where each user has only one request and one service, we can construct a weighted directed graph G with each vertex of G represents a user. If user A provides a service that user B requests, we connect A to B with a directed edge and the weight of the edge is assigned to be the charge user A asks for the service. Because each user can have at most one incoming edge for its only request and the restriction of the problem that each service each user provides needs to be returned by the service consumer either directly or indirectly, the problem is reduced to finding the minimum weighted cycle cover in the graph G. Therefore, the simple version of the request-service problem is solvable in $O(N^3)$ where N is the number of users in the problem.

4 Solution to the General Version of the Request-Service Problem

Now, we show that the general version of the request-service problem can also be reduced to the problem of finding a minimum weighted cycle cover in a directed graph. Assume that user u has r_u requests and can provide s_u services, we can construct a directed graph G as follows. For each request of user u, we use a vertex to represent it. Such vertices form a set of size r_u and we use R_u to denote the set. Similarly, s_u vertices are used to represent the services provided by u and we use S_u to denote the set. For each user u, we create a directed edge from every vertex in R_u to every vertex in S_u. Each of such edges is assigned a weight of 0. If user u can provide a service to fulfill a request from user v, we create a directed edge from the corresponding vertex in S_u to the corresponding vertex in R_v. The weight of the edge is the price u charges v for that service. For each user u, we create an additional vertex s' for each vertex s in S_u and connect s' with s with two directed edges that form a loop. The edges in each loop are assigned a weight of 0. These additional vertices are needed since some services may not be needed in a solution. Figure 1 provides an example on the construction of graph G.

Now, since each vertex that represents a request should have in-degree one while vertices that represent services may have out-degree more than one in a feasible solution, the general version of the request-service problem corresponds to a minimum weighted strongly connected component cover (SCC cover) G' such that G' is a subgraph of G covering every vertex of G and G' is a set of vertex disjoint strongly connected components. For convenience, we call such G' a minimum SCC (Strongly Connected Component) cover of graph G.

Now, we construct a new directed graph H from G and show that a minimum weighted SCC cover in a directed graph G corresponds to a minimum weighted cycle cover in H. We have the following observations.

Lemma 4.1

A SCC cover in a directed graph $G = (V, E)$ can be further decomposed into at most $|E|$ cycles $C_1, C_2, C_3, \cdots, C_k$ where $k \leq |E|$ such that each cycle C_i contains at least one edge that does not appear in other cycles.

Proof—We initialize the set of cycles C to be empty and we then construct a cycle by arbitrarily picking from the SCC cover one edge that has not been contained in any cycles in set C. Since the edge is in a SCC cover, we are able to construct a cycle from the SCC cover to contain this edge. We add this cycle to set C. We repeat this procedure until every edge in the SCC cover is covered by at least one cycle in set C. Since every cycle in set C contains an edge that is not covered by other cycles, the number of cycles in set C is at most $|E|$.

Based on the decomposition proposed in Lemma 4.1, we know that each edge in G is contained in at most $|E|$ cycles in the decomposition and we thus can split each edge into $|E|$

edges and the cycles in the decomposition then become disjoint. We call such a decomposition a *cycle decomposition* of a SCC cover of G and we use H to denote it. We now show how we can reduce the problem of finding a minimum SCC cover in G to computing a minimum weighted cycle cover in H.

To construct the new graph H, we replace each vertex u in G with $|E|$ vertices. We denote them with $u(1), u(2), \cdots, u(|E|)$. If (u, v) is an edge in G from u to v, we connect $u(i)$ and $v(i)$ with a directed edge from $u(i)$ to $v(i)$ for each $1 \le i \le |E|$. An edge in G is thus replaced by a set of $|E|$ parallel edges. Now, the cycles in the cycle decomposition of a SCC cover of G can be decomposed into $|E|$ disjoint cycles. Figure 2 (a)(b) show an example of a simple graph G and its corresponding cycle decomposition graph H. However, three issues must be resolved before we can claim that computing a minimum weighted cycle cover in H corresponds to a minimum weighted SCC cover in G. First, the weights of these duplicated parallel edges need to be handled in an elegant way such that the weight of a cycle cover in H is equal to that of a SCC cover in G. Second, the constraint that every vertex that represents a request can only have in-degree one in the SCC cover must be satisfied in a cycle cover of H. Third, since $|E|$ is only an upper bound of the number of cycles that can pass an edge in the cycle decomposition of a SCC cover, a mechanism that can cover those "unused" vertices in H is needed. We thus need to design additional graph gadgets to guarantee that these three issues are solved.

For the first and third issues, we need to add additional auxiliary vertices to each set of duplicated $|E|$ edges. In particular, each duplicated edge $(u(i), v(i))$ is added two intermediate vertices $i(u, v, i, 1)$ and $i(u, v, i, 2)$ and the weight of edge $(u(i), i(u, v, i, 1))$ is assigned to be $\frac{w(u,v)}{|E|}$ while edges $(i(u, v, i, 1), i(u, v, i, 2))$ and $(i(u, v, i, 2), v)$ are assigned weight 0. We create an additional directed edge of weight 0 from $i(u, v, 1, 2)$ to $i(u, v, 1, 1)$. We also create edges from $i(u, v, i + 1, 2)$ to $i(u, v, i, 2)$ and from $i(u, v, i, 1)$ to $i(u, v, i + 1, 1)$, where $1 \le i < |E|$. All are assigned weight 0. Now we add an additional vertex $p(u, v)$ and create two directed edges of weight 0 from $i(u, v, |E|, 1)$ to $p(u, v)$ and from $p(u, v)$ to $i(u, v, |E|, 2)$.

Now, we add $|E|$ helper vertices $h(u, v, 1), h(u, v, 2), \cdots, h(u, v, |E|)$ and for each $1 \le i \le |E|$, we create four directed edges: $(h(u, v, i), u(i))$ of weight 0, $(v(i), h(u, v, i))$ of weight 0, $(h(u, v, i), i(u, v, i, 1))$ of weight $\frac{w(u,v)}{|E|}$ and $(i(u, v, i, 2), h(u, v, i))$ of weight 0. Now, we connect all the helper vertices and $p(u, v)$ into a bi-directional clique of weight 0. In other words, every pair of the $|E| + 1$ vertices are connected with two directed edges of weight 0. For the convenience of notation, we also denote this new directed graph with H. Figure 3 shows an example of the gadgets created for each set of parallel edges. We have the following lemma.

Lemma 4.2

A SCC cover in G corresponds to a vertex disjoint cycle cover in H of the same weight and vice versa.

Proof—First, we show that given a SCC cover G' in G, we are able to find a corresponding vertex disjoint cycle cover in H. From lemma 4.1, we can decompose G' into a set C of $|E|$ cycles, $C = \{C_1, C_2, \cdots, C_{|E|}\}$. We can repeat the following procedure to "embed" these cycles into H.

1. Mark every cycle in C with 0.
2. Set a variable m to be 0.
3. Arbitrarily pick one cycle in C with mark 0 and call it C_i.

4. Find in C all cycles that are disjoint with C_i and have mark 0.
5. For each cycle found in step 4, we replace each edge (u, v) in G by edge $(u(m), v(m))$ in H. This can "embed" the cycle in graph H.
6. Increase m by 1 and mark every just embedded cycle with 1.
7. Go to step 3 if C contains at least one cycle with mark 0.

Since C contains up to $|E|$ cycles, m can never be larger than $|E|$. The procedure thus can "embed" every cycle into graph H and these cycles are now vertex disjoint. Next, we show that the remaining vertices in H can also be covered with cycles disjoint from these embedded cycles. We can cover the remaining vertices with the following procedure.

1. Divide the edges not incident on helper or intermediate vertices in graph H into two sets D and F such that an edge $(u(k), v(k))$ is in D if and only if (u, v) is an edge in G' and $(u(k), v(k))$ is not covered by the embedded cycles. An edge $(u(k), v(k))$ is in F if and only if (u, v) is an edge in G but not an edge in G' and $(u(k), v(k))$ is not covered by the embedded cycles.
2. Mark each edge in D with 0.
3. Arbitrarily pick an edge $(u(k), v(k))$ in D with mark 0 and check whether vertices $u(k)$ and $v(k)$ have been covered. If neither of them is covered, choose the cycle $h(u, v, k), u(k), i(u, v, k, 1), i(u, v, k, 2), v(k), h(u, v, k)$ to cover both $u(k)$ and $v(k)$. If $u(k)$ is not covered but $v(k)$ is, choose the cycle $h(u, v, k), u(k), i(u, v, k, 1), i(u, v, k, 2), h(u, v, k)$ to cover $u(k)$. Similarly, if $u(k)$ is covered but $v(k)$ is not, choose the cycle $h(u, v, k), i(u, v, k, 1), i(u, v, k, 2), v(k), h(u, v, k)$ to cover $v(k)$. If both vertices are covered, we choose the cycle $h(u, v, k), i(u, v, k, 1), i(u, v, k, 2), h(u, v, k)$ to cover vertices $i(u, v, k, 1)$ and $i(u, v, k, 2)$. We then mark the edge $(u(k), v(k))$ with 1.
4. If D still has an edge with mark 0, go to step 3.
5. For each edge $(u(k), v(k))$ in F, because (u, v) is not in G', all edges $(u(t), v(t))$ where $1 \leq t \leq |E|$ are in F and we thus can consider them as a whole. To this end, we use the cycle $p(u, v), i(u, v, |E|, 2), i(u, v, |E| - 1, 2), \cdots, i(u, v, 1, 2), i(u, v, 1, 1), i(u, v, 2, 1), \cdots, i(u, v, |E| - 1, 1), p(u, v)$ to cover all intermediate vertices. We then use the bidirectional clique between helper vertices to cover all helper vertices $h(u, v, t), 1 \leq t \leq |E|$.
6. For edge $(u(k), v(k))$ that is covered by the embedded cycles, at least one helper vertex $h(u, v, t)$ for edge (u, v) is not covered by the cycle constructed in above steps. For those uncovered helper vertices, we use the bi-directional clique among them and the vertex $p(u, v)$ to cover all of them and $p(u, v)$.

It is easy to check that this procedure can create a set of vertex disjoint cycles that can cover every vertex in H that are not covered by embedded cycles. In addition, the weight of this cycle cover is the same as that of G' in G since for every $(u, v) \in G'$, the associated cycles that cover corresponding vertices in H have a total weight of $w(u, v)$ and if (u, v) is not in G', the associated cycles that cover the corresponding vertices have weight 0.

Now, we show that we can construct a SCC cover from a vertex disjoint cycle cover of H. In particular, for vertex $p(u, v)$ in H, there are only two ways to cover it in a cycle: either the cycle formed by itself with some helper vertices of (u, v) or the cycle formed by itself with all intermediate vertices. We pick edge (u, v) if the corresponding vertex $p(u, v)$ in H is covered in the former way, i.e., $p(u, v)$ is covered with a cycle formed by itself with some helper vertices. Therefore, to cover vertices $i(u, v, k, 1)$ and $i(u, v, k, 2)$ for some $1 \leq k \leq |E|$, a cycle must pass through vertices $u(k)$, $i(u, v, k, 1)$, $i(u, v, k, 2)$ and $v(k)$. Thus all selected edges in G form a SCC cover. We denote this SCC cover with G'.

Next we show that every vertex in G is contained in G'. Assume on the contrary that there exists a vertex v in G that is not included in G', then for each edge (u, v) incident on v in G, $(u(k), v(k))$ is not an edge in G' and $(u(k), v(k))$ is not covered by the embedded cycles in H. Therefore, vertex $p(u, v)$ is covered by a cycle formed by all the intermediate vertices associated with edge (u, v). It thus is impossible to cover the vertex $v(k)$ in the disjoint cycle cover of H, which is contradictory to the assumption that G' is constructed from a cycle cover of H.

Now we have shown that G' is a SCC cover of G. It is easy to see that the SCC cover constructed this way has the same weight as that of the cycle cover of H, since for each edge (u, v) in G', the cycle that cover $p(u, v)$ and some helper vertices are of weight 0. In addition, the cycles that cover vertices $u(k)$ and $v(k)$ for all $1 \le k \le |E|$ in H have a total weight of $w(u, v)$.

Now, we construct graph gadgets to guarantee the constraint that each vertex for a request in G can only have in-degree one in a vertex disjoint cycle cover. To guarantee this, consider any parallel edge $(u(i), v(i))$ where v represents a request. We use $T(v, i)$ to denote the set of parallel edges $(s(j), v(j))$ where $s \ne u$ and $j \ne i$. For edge $(u(i), v(i))$, we create two additional auxiliary vertices $x(u, v, i)$ and $y(u, v, i)$. On $(u(i), v(i))$, we create an intermediate vertex $t(u, v, i)$. Similarly on each edge in $T(v, i)$, we create an intermediate vertex and denote them with $t(u, v, i, 1), \cdots, t(u, v, i, |T(v, i)|)$. Now, we create a directed cycle by connecting $x(u, v, i)$ to $y(u, v, i)$, $t(u, v, i)$ to $x(u, v, i)$, and $y(u, v, i)$ to $t(u, v, i)$ with three directed edges of weight 0. Then for all the intermediate vertices other than $t(u, v, i)$, we form a second directed cycle by arbitrarily ordering the vertices with $x(u, v, i)$ and $y(u, v, i)$ such that the vertices that are associated with the same edge in G are consecutive in the order with $y(u, v, i)$ and $x(u, v, i)$ being the first and the last vertices, respectively. Now, we connect each pair of consecutive vertices in the order with a directed edge of weight zero. In addition to these two cycles, we form a third cycle that can bypass some of the intermediate vertices for the edges in $T(v, i)$. In particular, we connect $x(u, v, i)$ to $t(u, v, i)$ and then follow the reversed order of the second cycle we just constructed. In addition, we create some directed edges such that the intermediate vertices for some edges (possibly all edges) for a particular edge in G can be bypassed in the third cycle. All these edges are also of weight 0. Figure 4 shows an example of this gadget. It is not difficult to see that, to cover $x(u, v, i)$ and $y(u, v, i)$, if edge $(u(i), v(i))$ is chosen in a cycle cover, $t(u, v, i)$ cannot be used to cover $x(u, v, i)$ and $y(u, v, i)$. Therefore, we can only pick the second cycle, which will force other edges in $T(v, i)$ not to be selected in the cycle cover. We denote this new directed graph with H_f and we have the following lemma.

Lemma 4.3

A SCC cover in G where each vertex for a request has in-degree one corresponds to a cycle cover in H_f and vice versa.

Proof—First, we show that we can construct a cycle cover in H_f from a SCC cover G' in G where each vertex for a request has in-degree one. From Lemma 4.2, we can construct from G' a cycle cover in H with the same weight. Now, we find cycles that can cover those additional vertices in H_f. Since each vertex for a request in G' has in-degree one, for edge $(u(i), v(i))$ in the cycle cover of H, we can pick the second cycle of vertices $x(u, v, i)$ and $y(u, v, i)$ to cover the two vertices and the intermediate vertices in other edges in $T(v, i)$. For any edge $(s(k), v(k))$ in $T(v, i)$, we can use the third cycle of vertices $x(s, v, k)$ and $y(s, v, k)$ to bypass the intermediate vertices on edges $(u(i), v(i))$ that are in the cycle cover, where $1 \le i$

$\leq |E|$, since these intermediate vertices have been covered. It is not difficult to see that these cycles can cover all vertices in H_f without increasing the total weight of the cycle cover. For the reversed direction, notice that to cover two additional vertices $x(u, v, i)$ and $y(u, v, i)$ for each edge $(u(i), v(i))$ in H_f, we only have three options and two of them need the vertex $t(u, v, i)$. Therefore, if $(u(i), v(i))$ is in a cycle cover of H_f, we can only pick the second cycle to cover the two vertices and this cycle will disable the inclusion of other edges in $T(v, i)$ in the cycle cover. This thus guarantees the satisfaction of the constraint that each vertex for request only has in-degree one. Since H is a subgraph of H_f, the same argument of Lemma 4.2 applies and we eventually obtain a SCC cover as claimed.

Now, by putting all these together, we obtain the following theorem.

Theorem 4.1

The general version of the request-service problem can be solved in time $O(n^{12})$ if the number of requests and services provided by each user is bounded by a constant.

Proof—The directed graph $G = (V, E)$ has $O(n)$ vertices since each user has a constant number of requests and services. Graph H may have up to $O(n|E|)$ vertices and graph H_f may have up to $O(n^2|E|)$ since the in-degree of every vertex in G is bounded by $O(n)$. Based on lemmas 4.2 and 4.3, we can apply the Hungarian method on H_f to compute the minimum weighted cycle cover problem and it needs up to $O((n^2|E|)^3)$ time. Since $E = O(n^2)$, we get the time complexity as claimed.

5 To satisfy as many users as possible

In this section, we consider the problem that a solution that can satisfy all users' requests does not exist. In this case, we want to satisfy the requests of as many users as possible, i.e., to maximize the number of satisfied users. We construct a reduction from the MAX2SAT problem to show the NP-hardness of this problem. In particular, given a set of boolean variables and a set of clauses of these variables, and each clause contains two literals, the objective of MAX2SAT problem is to find an assignment of variables to maximize the number of satisfied clauses. This problem is a well known NP-hard problem.

Theorem 5.1

Minimizing the total weight while maximizing the number of satisfied users is NP-hard.

Proof. Given an instance of MAX2SAT, we assume the boolean formula contains n variables $x_1, x_2, \cdots, x_n$ and m clauses $c_1, c_2, \cdots, c_m$ in total. Each clause contains two literals. As the first step of the reduction, we construct a directed graph G from the instance of MAX2SAT and each vertex in G is associated with a weight value. We show that a set of vertex disjoint cycles that cover a set of vertices with maximum total weight in G corresponds to an assignment that maximizes the number of satisfied clauses. We then further show that the weighted instance can be reduced to an unweighted instance, where each vertex has the same weight.

G contains two types of graph gadgets: variable gadgets and clause gadgets. In particular, each variable is represented with a variable gadget. Figure 5(a) shows a variable gadget for variable x_i. The gadget contains an *input vertex* $a(i)$ and an *output vertex* $b(i)$ and four additional vertices $c(i, 1, 0)$, $c(i, 2, 0)$, $c(i, 1, m)$, $c(i, 2, m)$. Each of the four vertices is associated with a weight value of m^2. In addition, we create $m - 1$ intermediate vertices $c(i, 1, t)$ where $0 < t < m$ on the directed path from $c(i, 1, 0)$ to $c(i, 1, m)$ and $m - 1$ intermediate vertices $c(i, 2, t)$ where $0 < t < m$ on the directed path from $c(i, 2, 0)$ to $c(i, 2, m)$. These intermediate vertices create m partitions and each occurrence of variable x_i will need one of them to construct a clause gadget. For each $0 < t < m$, we connect $c(i, 1, t)$ with $c(i, 2, t)$ with a bidirectional gadget and this bidirectional gadget is shown in Figure 5(b). There are m^2 vertices in a directed path and each of these vertices is of weight 1. We add two additional vertices $e(i, 1, t - 1)$ and $e(i, 2, t - 1)$ on the directed edges $(c(i, 1, t - 1), c(i, 1, t))$ and $(c(i, 2, t - 1), c(i, 2, t))$, respectively. The weight of the two additional vertices is m^2 and they are used to further "restrict" the vertex disjoint cycles that can cover all vertices with weight m^2 in a variable gadget. It is not difficult to see from Figure 5(a)(b) that there are two directed paths from $a(i)$ to $b(i)$ that can cover all vertices in the gadget. They are $a(i)$, $c(i, 1, 0)$, ⋯, $c(i, 2, 0)$, $c(i, 2, 1)$, ⋯, $c(i, 1, 1)$, $c(i, 1, 2)$, ⋯, $b(i)$ and $a(i)$, $c(i, 2, 0)$, ⋯, $c(i, 1, 0)$, $c(i, 1, 1)$, ⋯, $c(i, 2, 1)$, $c(i, 2, 2)$, ⋯, $b(i)$. Therefore, each of the two paths "encodes" a truth value for variable x_i. In particular, x_i is assigned to be true if the first directed path is used and false otherwise. The first directed path is thus the *true path* for x_i and the second one is the *false path* for x_i. Now, we connect $b(i)$ to $a(i)$ with a directed edge. Such an edge is called an *intermediate edge*. We create m^2 additional vertices on the intermediate edge and each additional vertex is assigned a weight of 1.

Figure 6 (a) shows the gadget for a clause. Each clause is represented by a clause gadget. Without loss of generality, we assume that a clause c_k contains two literals l_i and l_j. If l_i is x_i we create two additional vertices $l(i, k, 1)$ and $l(i, k, 2)$ in the kth partition on its false path $(c(i, 2, 0), c(i, 1, 0), \cdots, c(i, 2, t), \cdots)$. Otherwise, $l(i, k, 1)$ and $l(i, k, 2)$ are created on the kth partition on its true path $(c(i, 1, 0), c(i, 2, 0), \cdots, c(i, 1, t), \cdots)$. Now, we add two additional directed edges to form a directed cycle that connects vertices $l(i, k, 1)$, $l(i, k, 2)$, $l(j, k, 1)$, $l(j, k, 2)$. Without loss of generality we assume the cycle is $l(i, k, 1)$, $l(i, k, 2)$, $l(j, k, 1)$, $l(j, k, 2)$, $l(i, k, 1)$. We then create two additional vertices on each edge in this cycle. In particular, $d(i, k, 1)$, $d(i, k, 2)$ are on the edge $(l(i, k, 1), l(i, k, 2))$, $e(i, k, 2)$, $e(j, k, 1)$ on the edge $(l(i, k, 2), l(j, k, 1))$, $d(j, k, 1)$, $d(j, k, 2)$ on the edge $(l(j, k, 1), l(j, k, 2))$ and $e(j, k, 2)$, $e(i, k, 1)$ on the edge $(l(j, k, 2), l(i, k, 1))$.

As the last step to create a clause gadget, we create two vertices $m(i, j, 1)$ and $m(i, j, 2)$. For vertex $m(i, j, 1)$, we create edges $(m(i, j, 1), e(i, k, 1))$, $(m(i, j, 1), d(j, k, 2))$, $(d(i, k, 1), m(i, k, 1))$, $(e(j, k, 2), m(i, j, 1))$. For vertex $m(i, j, 2)$, we create edges $(m(i, j, 2), e(j, k, 1))$, $(m(i, j, 2), d(i, k, 2))$, $(e(i, k, 2), m(i, j, 2))$, and $(d(j, k, 1), m(i, j, 2))$. We then add an additional vertex on each of the edges $(e(j, k, 2), m(i, j, 1))$, $(m(i, j, 1), e(i, k, 1))$, $(e(i, k, 2), m(i, j, 2))$ and $(m(i, j, 2), e(j, k, 1))$. These vertices are called *crucial vertices*. All vertices in a clause gadget are of weight 1.

Now, we show that if k of the m clauses can be satisfied by an assignment, $|W| - 10m + 6k$ of G can be covered by a cycle cover in G, where $|W|$ is the total weight of vertices of G. In particular, given the assignment that can satisfy k clauses, we choose the path in each variable gadget based on the truth value of that variable in the assignment and its intermediate edges. All intermediate vertices are thus covered. For clause gadgets, it is not difficult to see that, as shown in Figure 6(b), for each satisfied clause, we can create a cycle such that 4 vertices are left uncovered in a clause gadget. However, for a clause that is not satisfied, 10 vertices are not covered in total in its clause gadget. Therefore, in total, we have $4k + 10(m - k) = 10m - 6k$ vertices uncovered by this cycle cover. Therefore the total weight of covered vertices are $|W| - 10m + 6k$.

Now we prove the reverse direction: if G has a set of disjoint cycles that covers a total number of $|W| - 10m + 6k$ vertices in G, there exists an assignment that satisfies k clauses. First, we notice that each variable gadget should choose a truth value for the corresponding variable since otherwise at least one vertex of weight m^2 or m^2 additional vertices on an intermediate edge or an edge that connects two intermediate vertices in a variable gadget will not be covered by these cycles. However $m^2 > 10m - 6k$ for sufficiently large m, which is a contradiction. Therefore, this cycle cover chooses a truth value for each variable. Now we show that this assignment satisfies at least k clauses. We assume that k' clauses are satisfied by this cycle cover. Then each of the gadgets of these clauses contain at least 4 uncovered vertices. In the gadget of an unsatisfied clause, the number of uncovered vertices is at least 10. Therefore, we have at least $4k' + 10(m - k') = 10m - 6k'$ vertices uncovered in G by this cycle cover. Therefore, we have $|V| - 10m + 6k \le |V| - 10m + 6k'$. This is equivalent to $k' \ge k$. The case where vertices are weighted has been proved.

Now, we show that the weighted case with a directed graph G of weighted vertices can be easily reduced to an unweighted case with a directed graph of unweighted vertices. In fact, each weighted vertex in a weighted case can be replaced by a directed path with a few vertices on it. The number of such vertices is equal to its weight. We then replace the head of all incoming edges with the left end of the path and the tail of all outgoing edges with the right end of the path. It's not difficult to see that the resulting graph H is an unweighted instance and an unweighted solution in H corresponds to a solution in G and vice versa. This simple reduction thus shows the NP-hardness of the unweighted case of the problem.

Eventually, since we have shown that it is NP-hard to minimize the number of uncovered vertices by a cycle cover in a directed graph and there is only a special case for the minimization problem in the request-service problem, where each user has only one request and can provide only one service, the general version of the problem is thus NP-hard as claimed.

6 Conclusions and Future Work

In this paper, we investigated the request-service problem. Based on an algorithm that can find the minimum weighted cycle cover in a directed graph in time $O(n^3)$, where n is the number of vertices in the graph, we show that both the simple and general versions of the problem can be reduced to finding the minimum weighted cycle cover in a directed graph, thus both of them can be solved in polynomial time, in particular, in $O(N^3)$ and $O(N^{12})$, respectively. If a feasible solution (i.e. all requests of each user can be satisfied) does not exist, we want to satisfy as many users as possible. For the simple version of the problem, because the number of requests of each user is only one, the same algorithm for the problem can be used. However, for the general version of the problem, if a feasible solution does not exist, we show that finding a minimum weighted solution that maximizes the number of satisfied users is NP-hard.

Our future work consists of two important directions. For the case that a feasible solution does not exist for the general version of the request-service problem, we will work on developing an approximation algorithm for the problem. The other is to explore the possibility of developing a more efficient exact algorithm for the general version of the request-service problem and will apply the algorithm to the kidney exchange problem.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. This work was supported in part by grant 2 G12 RR003048 from the RCMI Program, Division of Research Infrastructure, National Center for Research Resources, NIH, and Howard University Mordecai Wyatt Johnson program.

References

1. Abraham, David; Blum, Avrim; Sandholm, Tuomas. Clearing Algorithms for Barter-Exchange Markets: Enabling Nationwide Kidney Exchanges. Proceedings of ACM-EC 2007: the Eighth ACM Conference on Electronic Commerce;

2. Castro M, Druschel P, Ganesh A, Rowstron A, Wallach DS. Security for structured peer-to-peer overlay networks. Proceedings of OSDI 2002, Boston, MA. 2002

3. Cooper, BF.; Garcia-Molina, H. Bidding for storage space in a peer-to-peer data preservation system. Proceedings of 22nd International Conference on Distributed Computing Systems; Vienna, Austria. July 2002;

4. Delmonico FL. Exchanging kidneys-advances in living-donor transplantation. New England Journal of Medicine. 2004; 350:1812–1814. [PubMed: 15115826]

5. Druschel, P.; Rowstron, A. PAST: A large-scale, persistent peer-to-peer storage utility. Proceedings of 8th Workshop on Hot Topics in Operating Systems; Schoss Elmau, Germany. May 2001;

6. Gentry SE, Segev DL, Montgomery RA. A comparison of populations served by kidney paired donation and list paired donation. American Journal of Transplantation. 2005; 5(8):1914–1921. [PubMed: 15996239]

7. Intervac. http://intervac-online.com

8. Kuhn, Harold W. The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly. 1955; 2:83–97.

9. Manthey, Bodo. Dissertation, Universitat zu Lbeck, Technisch-Naturwissenschaftliche Fakultat, Institut fr Theoretische Informatik. December. 2005 Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees.

10. Ngan, Tsuen Wan; Wallach, Dan S.; Druschel, Peter. Enforcing Fair Sharing of Peer-to-Peer Resources. 2nd International Workshop on Peer-to-Peer Systems (IPTPS);

11. Papadimitriou, CH. Computational Complexity. Addison-Wesley; USA: 1994.

12. Peerflix. http://www.peerflix.com

13. Read it swap it. http://www.readitswapit.co.uk

14. Segev DL, Gentry SE, Warren DS, Reeb B, Montgomery RA. Kidney paired donation and optimizing the use of live donor organs. Journal of the AMA. 2005; 293(15):1883–1890.

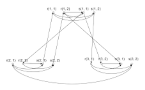

Figure 1.
A graph constructed for three users. Each user has two requests and two services. For clarity, we omit the weights of the edges and the additional vertices for each user.

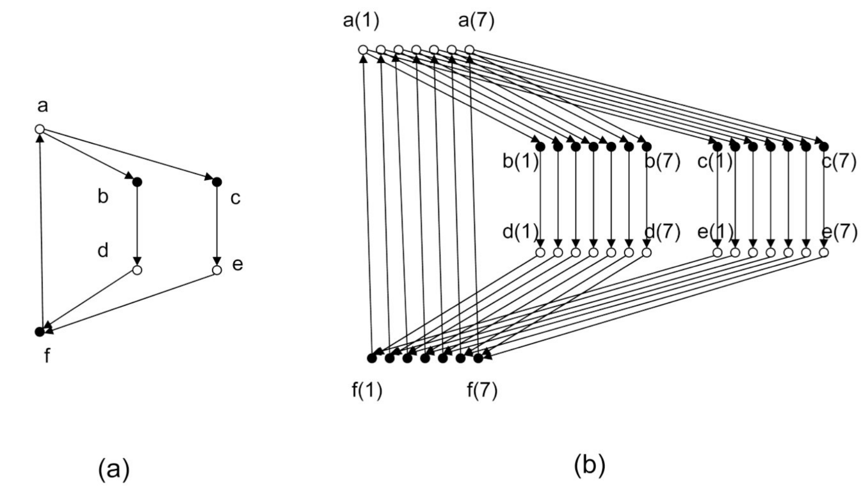

Figure 2.
(a) An example of a simple request-service graph *G*, where the filled vertices are for services and unfilled vertices are for requests. (b) Graph *H* constructed from *G*. It can be seen from the graph that the two overlapped cycles in (a) become two disjoint cycles in *H*.

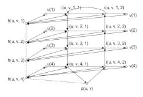

Figure 3.
An example of the gadgets created for each set of parallel edges. For simplicity, we assume there are four parallel edges (*u*(1), *v*(1)), (*u*(2), *v*(2)), (*u*(3), *v*(3)), and (*u*(4), *v*(4)) in the set, and the vertices connected with dashed lines form a bi-directional clique.

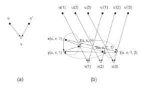

Figure 4.
(a) Vertex *v* represents a request and could be served by two different services *u* and *u*' respectively. (b) To guarantee the constraint that *v* is of in-degree one in the cycle cover, we create a pair of vertices *x*(*u*, *v*, 1) and *y*(*u*, *v*, 1) and create three cycles as described. The dashed lines represent the third cycle, where some vertices could be bypassed.

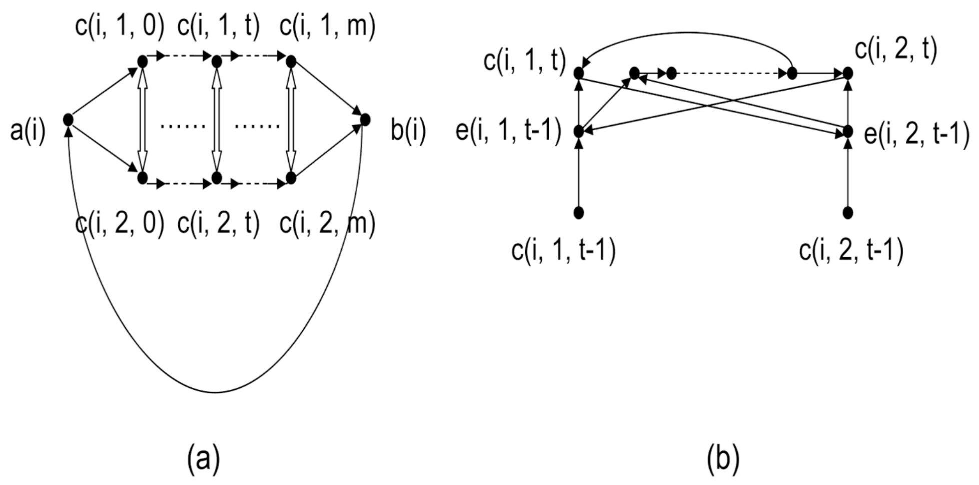

Figure 5.

(a) The graph gadget for variable x_i. (b) All variable gadgets are connected into a "loop".

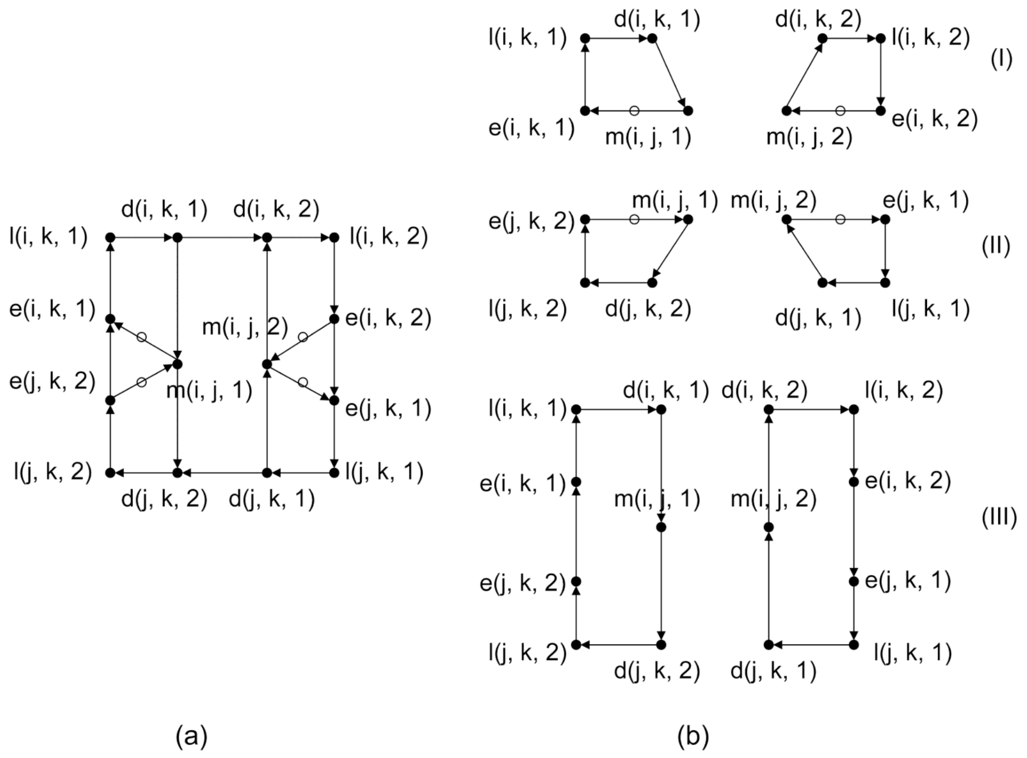

Figure 6.

(a) The graph gadget for a clause that contains literals l_i and l_j. (b) Disjoint cycles that can cover the maximum number of vertices in a clause gadget: (I) when literal l_i is true but l_j is false; (II) when literal l_i is false but l_j is true; (III) when both literals are true.