STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIV**, Number 2, 2009

ON THE DEBTS' CLEARING PROBLEM

CSABA PĂTCAŞ

Abstract. The debts' clearing problem is about clearing all the debts in a group of n entities (eg. persons, companies) using a minimal number of money transaction operations. First we will model the problem using graph theoretical concepts and we will reduce the problem to a minimum cost maximum flow problem in a bipartite graph. Because our cost function is not the usual one, a classical minimum cost maximum flow algorithm cannot be applied in this case. We propose a solution using the dynamic programming method with $\Theta(2^n)$ space and exponential time complexity. We will also examine other solving possibilities.

1. Introduction

In this article we will discuss an original problem proposed in 2008 by the author at the qualification contest of the Romanian national team of informatics for the Central European Olympiad of Informatics and Balkan Olympiad of Informatics. The problem of debts' clearing is one, that arises in real life situations as well. In a group of persons that know each other it is not uncommon to borrow some amount of money to an acquaintance for a period of time. This process is also happening among different banks, or even countries. As money transactions are time and money sensitive operations, it is desirable to clear the debts in a minimal number of money transaction operations.

Our article studies some methods to solve this task, conjectured to be NP-complete.

Received by the editors: November 1, 2008.

2010 *Mathematics Subject Classification.* 05C85, 90C39.

1998 *CR Categories and Descriptors.* G.2.2 [**Graph theory**]: Network problems – *Network flow*.

Key words and phrases. NP-completeness, dynamic programming, network flow.

This paper has been presented at the 7th Joint Conference on Mathematics and Computer Science (7th MaCS), Cluj-Napoca, Romania, July 3-6, 2008.

2. The task

In the following we will give the problem statement.

Let us consider a number of n entities (eg. persons, companies), and a list of m borrowings among these entities. A borrowing can be described by three parameters: the index of the borrower entity, the index of the lender entity and the amount of money that was lent. The task is to find a minimal list of money transactions that clears the debts formed among these n entities as a result of the m borrowings made.

Let us clarify this by the following example:

Let $n = 6$, $m = 5$

List of borrowings:

Borrower	Lender	Amount of money
1	2	10
2	3	10
4	5	5
5	6	5
6	4	5

Solution:

Sender	Reciever	Amount of money
1	3	10

Explanation: The circular borrowings among entities 4, 5 and 6 cancel out each other, and the two borrowings made among 1, 2 and 3 can be cleared using just one money transaction.

3. Reformulation using graph theory

We can reformulate the problem using graph theoretical concepts. For this purpose we need to define some new terms first.

Definition 1. Let $G(V, A, W)$ be a directed, weighted multigraph without loops, $|V| = n$, $|A| = m$, $W : A \to \mathbb{Z}$, where V is the set of vertices, A is the set of arcs and W is the weight function. G represents the borrowings made, so we will call it the **borrowing graph**.

Definition 2. Let us define for each vertex $v \in V$ the **absolute amount of debt** over the graph G: $D_G(v) = \sum_{\substack{v' \in V \\ (v,v') \in A}} W(v, v') - \sum_{\substack{v'' \in V \\ (v'',v) \in A}} W(v'', v)$

Definition 3. $G(V, A, W) \sim G'(V, A', W')$ if and only if:

$D_G(v_i) = D_{G'}(v_i), \forall i = \overline{1,n}$, where $V = \{v_1, v_2, \dots, v_n\}$

Theorem 4. *The "$\sim$" relation defined above is an equivalence relation.*

Proof.

(1) Reflexivity can be proved trivially: $D_G(v_i) = D_G(v_i), \forall i = \overline{1,n}$
(2) Symmetry is also trivial to prove: $D_G(v_i) = D_{G'}(v_i) \Rightarrow D_{G'}(v_i) = D_G(v_i), \forall i = \overline{1,n}$
(3) Transitivity: $D_G(v_i) = D_{G'}(v_i)$, $D_{G'}(v_i) = D_{G''}(v_i) \Rightarrow D_G(v_i) = D_{G''}(v_i), \forall i = \overline{1,n}$

□

Definition 5. There is an infinite number of G' graphs, that are in "$\sim$" relation with the G borrowing graph. As these G' graphs represent the transactions that are needed to clear the borrowings, we will call them **transaction graphs**.

In the following we will state the problem using the terms defined above:

We are looking for a minimal transaction graph $G_{min}(V, A_{min}, W_{min})$, for which $\forall G'(V, A', W') : G \sim G', |A_{min}| \leq |A'|$ holds.

Figure 1 shows the borrowing graph, that can be associated with the example given in Section 2, while Figure 2 shows the respective minimum transaction graph.

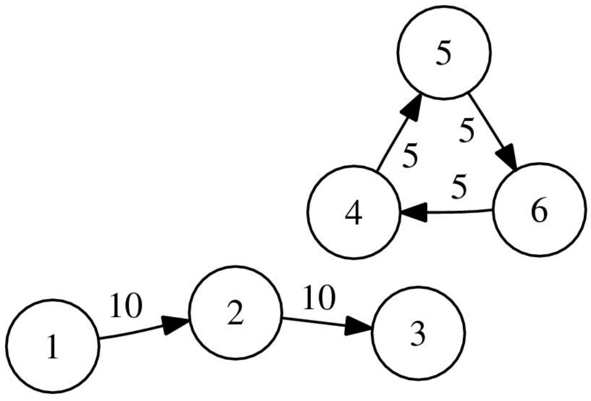

FIGURE 1. The borrowing graph associated with the given example. An arc from node i to node j with weight w means, that entity i must pay w amount of money to entity j.

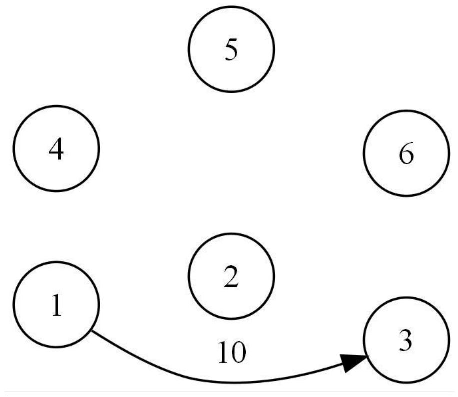

Figure 2. The respective minimum transaction graph. An arc from node i to node j with weight w means, that entity i pays w amount of money to entity j.

4. Possible solutions

In this Section we will propose some methods to solve the problem.

4.1. **Graph transformations.** It is clear that the original borrowing graph is also a transaction graph, because of the reflexivity of the "$\sim$" relation ($G \sim G$). This yields the following idea: Is it possible to find a sequence of graphs, such that $G = G_1, G_2, \ldots, G_k = G_{min}$, $G_1 \sim G_2$, $G_2 \sim G_3$,…,$G_{k-1} \sim G_k$ and G_j is obtained from G_{j-1} using some "elementary transformation operation" for every $j = \overline{2,k}$? Because the "$\sim$" relation is also transitive, it would immediately result, that $G = G_1 \sim G_k = G_{min}$.

Let us enumerate some transformation operations we could use:

- All arcs having weight zero can be discarded.
- It is clear, that if we have multiple arcs between two vertices, these can be united in a single one, having a weight equal to the sum of the original weights.
- It is also easy to see, that if for any two vertices a and b we have $(a, b) \in A$ and also $(b, a) \in A$, the arc having the smaller weight out of those two can be deleted, and its weight can be substracted from the another arc.

These findings suggest the existence of an elementary transformation operation, that can be given in the most general way as follows:

(1) Let $v_{i_1}, v_{i_2}, \ldots, v_{i_p}$ be a sequence of vertices.

(2) Let x be any integer number ($x \in \mathbb{Z}$).
(3) Substract x from all the weights of two consecutive vertices in the sequence ($W(v_{i_{j-1}}, v_{i_j}) := W(v_{i_{j-1}}, v_{i_j}) - x$, $j = \overline{2,p}$). If a weight becomes negative, an edge in the opposite direction should be added. As a result some edges could disappear and new edges could appear.
(4) Add an arc between v_{i_1} and v_{i_p} having weight x.

Theorem 6. *After applying the transformation described above, the resulting graph will be in "$\sim$" relation with the original one.*

Proof.
Let $G(V, A, W)$ be the graph before the transformation and let $G'(V, A', W')$ be the graph after the transformation. Thus we must prove, that $G \sim G'$.

Let us note with d the vector of absolute amount of debts of the graph before the transformation and with d' the vector of absolute amount of debts of the graph after the transformation ($d_i = D_G(v_i), d'_i = D_{G'}(v_i), \forall i = \overline{1,n}$). Thus we must prove, that $d = d'$.

The first two steps of the transformation does not alter the graph. During the third step, the absolute amount of debts do change:
$W(v_{i_1}, v_{i_2}) = W(v_{i_1}, v_{i_2}) - x \to d'_{i_1} = d_{i_1} - x, d'_{i_2} = d_{i_2} + x$
$W(v_{i_2}, v_{i_3}) = W(v_{i_2}, v_{i_3}) - x \to d'_{i_2} = d'_{i_2} - x = d_{i_2} + x - x = d_{i_2}, d'_{i_3} = d_{i_3} + x$
$\vdots$
$W(v_{i_{p-1}}, v_{i_p}) = W(v_{i_{p-1}}, v_{i_p}) - x \to d'_{i_{p-1}} = d'_{i_p} - x = d_{i_{p-1}} + x - x = d_{i_{p-1}}, d'_{i_p} = d_{i_p} + x$

So, when step 3 is completed, the only d values that are changed are d_{i_1} and d_{i_p}, that is $d'_{i_1} = d_{i_1} - x$ and $d'_{i_p} = d_{i_p} + x$. As a result of the fourth step these elements also get back to their original value:
$W(v_{i_1}, v_{i_p}) = W(v_{i_1}, v_{i_p}) + x \to d'_{i_1} = d'_{i_1} + x = d_{i_1} - x + x = d_{i_1}, d'_{i_p} = d'_{i_p} - x = d_{i_p} + x - x = d_{i_p}$ □

An attempt of using two concrete versions of this transformation was the following:
(1) Reduce the number of arcs in all the cycles of the graph, by using the minimal weight of the cycle's arcs as the x value.
(2) The resulting graph has no cycles, thus can be sorted topologically. Try to find a strategy to reduce the number of arcs in the paths of this graph, using the topological order. For instance simplify the longest paths first, in a similar way as the cycles, by using the weights' minimum as the x value.

Unfortunately this strategy doesn't work for all graphs. The order in which we eliminate the cycles and paths does matter, and it is not clear what this order should be. In Figure 3 a borrowing graph is illustrated. If we first apply the transformation to path 2-4-5-7 no more transformations can be made, and we get a graph with 5 arcs, that can be seen in Figure 4. The optimal solution is to apply the transformation to path 1-4-5-6, then to path 3-4-5-8, thus obtaining the graph from Figure 5, which has only 4 arcs.

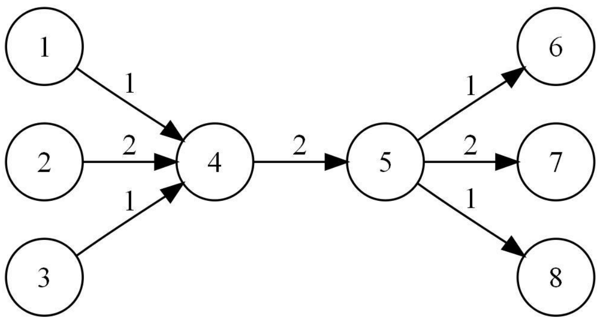

FIGURE 3. Example for a graph in which the order of transformations does matter

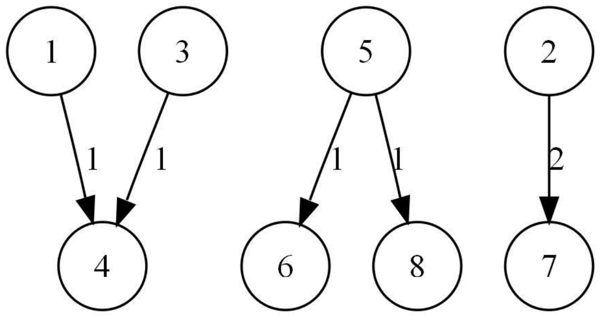

FIGURE 4. Non-optimal solution gained using graph transformations

The general problem with this approach is, that it is not clear in which order to choose which sequences, and what x values to use. The author could not find an algorithm based on this approach, that works for any borrowing graph.

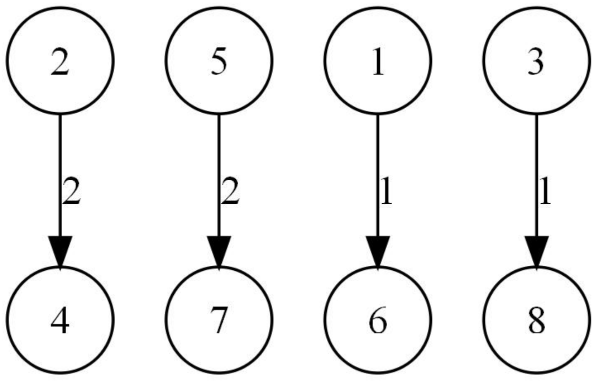

FIGURE 5. Optimal solution gained using graph transformations

4.2. **Reducing the debts' clearing to a flow problem.** A better approach is to try to solve the problem using network flows. Let us construct a bipartite graph $G_b(V_b, A_b)$ from the initial borrowing graph, as follows:

(1) $V_b = V_{left} \cup V_{right}$
(2) $V_{left} = \{v|v \in V, D_G(v) > 0\}$
(3) $V_{right} = \{v|v \in V, D_G(v) < 0\}$
(4) $A_b = \{(a,b)|a \in V_{left}, b \in V_{right}\}$

Using this graph let us construct a flow network $G_f(V_f, A_f, c)$, where $c : A_f \to \mathbb{Z}$ is the capacity function.

(1) We add a source and a sink: $V_f = V_b \cup \{s,t\}$
(2) We add arcs from the source to all the nodes from V_{left} and from all nodes from V_{right} to the sink: $A_f = A_b \cup \{(s,v)|v \in V_{left}\} \cup \{(v,t)|v \in V_{right}\}$
(3) We set the capacities of the arcs as follows:
$$c(i,j) = \begin{cases} D_G(j), & \text{if } i = s, j \in V_{left} \\ -D_G(i), & \text{if } i \in V_{right}, j = t \\ \infty, & \text{if } i \in V_{left}, j \in V_{right} \end{cases}$$

Theorem 7. *Finding the minimal transaction graph of the borrowing graph G is equivalent to finding a maximal flow in G_f with a minimal number of arcs, for which the flow is strictly positive.*

Proof.

Let F be a maximal flow with a minimal number of arcs, for which the flow is strictly positive. From this flow network we can construct the transaction graph $G'(V, A', W')$ in the following way:

(1) $V = V_f / \{s, t\} \cup \{k \in V | D_G(k) = 0\}$
(2) $A' = \{(i, j) | F(i, j) > 0, i \neq s, j \neq t\}$
(3) $W'(i, j) = F(i, j)$

Lemma 8. *The maximal flow F will saturate all arcs outgoing from s and all arcs incoming in t.*

Proof. Each arc $(i, j) \in A$ will increase $D_G(i)$ by $W(i, j)$ and decrease $D_G(j)$ by the same amount $\Rightarrow \sum\limits_{D_G(k)>0} D_G(k) = \sum\limits_{D_G(l)<0} -D_G(l) =: S$. In other words the sum of capacities of all arcs outgoing from s equals to the sum of capacities of all arcs incoming in t. We will prove that the maximal flow in the network has cost S. Let us suppose that the cost of the maximal flow is $S' \neq S$. As the sum of capacities of all arcs outgoing from s is $S \Rightarrow S' < S$ (the maximal flow cannot be greater than S). This means, that we have (at least) two unsaturated arcs (s, i) and (j, t). But the structure of the flow network ($c(i, j) = \infty$) leads to the existence of the augmenting path $s - i - j - t$, which contradicts the assumption, that S' was the cost of the maximal flow. We have proved, that the cost of the maximal flow F is S, which immediately yields the proof of the lemma (we cannot have this maximal flow, unless we saturate the respective arcs). □

Lemma 9. *Let G be the borrowing graph and G' be the graph constructed in the beginning of the proof of Theorem 7. Then $G' \sim G$.*

Proof.
We must prove, that $D_{G'}(v_i) = D_G(v_i) \forall i = \overline{1, n}$. We have two cases:

(1) If $D_{G'}(v_i) > 0$, there is an arc (s, v_i') in the flow network. It results from Lemma 8, that this arc is saturated. From the flow conservation rule ([3]) it results that the sum of the costs of all arcs outgoing from vertex v_i' will be $c(s, v_i') = D_G(v_i)$. From the construction of G' it follows, that there are no arcs incoming in v_i', so $D_{G'}(v_i) = D_G(v_i)$
(2) If $D_{G'}(v_i) < 0$, there is an arc (v_i', t) in the flow network, which is saturated. The rest of the proof, that $D_{G'}(v_i) = D_G(v_i)$ is done in the same manner as in the first case. □

From the fact, that we chose the maximal flow F in a way that minimizes $|A'|$, and from Lemma 9 it results, that we reduced the original problem to a maximal flow problem. □

5. Solving the flow problem

5.1. **Minimal cost maximal flow.** The first idea would be to associate costs $z(e), e \in A_f$ to the arcs of the flow network. Let the arcs between V_{left} and V_{right} have cost 1, and the other arcs have cost 0, and let us use a classical minimum cost maximum flow algorithm to solve the problem:

$$z(i,j) = \begin{cases} 1, & \text{if } i \in V_{left}, j \in V_{right} \\ 0, & \text{otherwise} \end{cases}$$

Figure 6 shows the flow network, that can be associated to the borrowing graph shown in Figure 1. The first number on each arc represents the capacity of the arc, and the second number represents the cost of the arc.

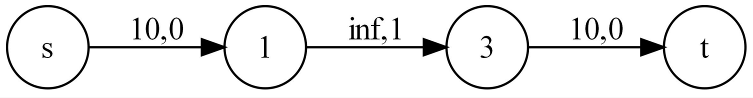

Figure 6. The flow network associated with the example

This approach doesn't work, because the cost of an arc is multiplied by the flow, we don't have fixed costs on the arcs. In a classical minimum cost maximum flow algorithm the cost function looks like: $Cost(e) = f(e) \cdot z(e)$, but in our case this is:

$$Cost(e) = \begin{cases} z(e), & \text{if } f(e) > 0 \\ 0, & \text{otherwise} \end{cases}$$

It can be rewritten to a more simple form, as: $Cost(e) = ind(f(e)) \cdot z(e)$, where *ind* is the indicator function, defined as follows:

$$ind(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{cases}$$

In the literature this problem is called Minimum Edge-Cost Flow and is known to be NP-complete([4], problem [ND32]). Even if the problem is constrained so that the capacity of each arc is 2, and the cost of each arc can be 0 or 1, it remains NP-complete. This fact leads us to the following conjecture:

Conjecture 10. *The debts' clearing problem is NP-complete.*

The fact that we have a special case, with all the costs equaling to 0 or 1, and having a complete bipartite graph, gives us a hope that a polynomial solution does exist.

5.2. **Convex flow.** Efficient minimum cost maximum flow algorithms do exist, when the cost function is convex ([1]). Unfortunately this is not our case. Indeed, a function to be convex must satisfy:

$f(t\cdot x+(1-t)\cdot y) \leq t\cdot f(x)+(1-t)\cdot f(y)$ for any x and y from the function domain, and $\forall t \in [0,1]$. For $t=\frac{1}{2}, x=0, y=1$ we get $f(\frac{1}{2}) \leq \frac{f(0)}{2}+\frac{f(1)}{2}$, which is obviously not true in our case, so our cost function is not convex.

5.3. **Nonlinear programming.** We can formulate the problem as a nonlinear programming (NLP) problem:

Minimize $p = \sum\limits_{e\in A_f} ind(f(e))$ subject to

$\sum\limits_{k\in V_{right}} f(i,k) = c(s,i)$

$\sum\limits_{k\in V_{left}} f(k,j) = c(j,t)$

Unfortunately no polinomial algorithm is known for solving any NLP instance. However good approximation algorithms do exist ([6]).

5.4. **Dynamic programming.** We will give a solution using the dynamic programming method. It uses similar techniques to the algorithm discovered independently by Bellman ([2]), respectively Held and Karp ([5]) for solving the Traveling Salesman Problem.

Let us note $n_1 = |V_{left}|$ and $n_2 = |V_{right}|$. Let us define the subproblems of the dynamic programming problem with two parameters i and j, where i is a binary representation of n_1 bits, and j is a binary representation of n_2 bits ($i=\overline{0,2^{n_1}-1}, j=\overline{0,2^{n_2}-1}$). A subproblem will have the following meaning:

$dp_{i,j}$ = the minimal number of arcs having strictly positive flow, such that the arcs between s and the nodes determined by the bits of i, and the arcs between the nodes determined by the bits of j and t are all saturated.

The recursive formula to determine the values of the subproblems is the following[1]:

$dp_{i,j} = \min(dp_{i \texttt{ XOR } i', j \texttt{ XOR } j'} + bitcount(i') + bitcount(j') - 1)$, where

(1) $i \texttt{ AND } i' = i'$

[1]We note by `AND` the bitwise and operation and by `XOR` the bitwise exclusive or operation

(2) j AND $j' = j'$

(3) $\sum\limits_{i' \text{ AND } 2^k \neq 0} c(s,k) = \sum\limits_{j' \text{ AND } 2^k \neq 0} c(k,t)$

(4) *bitcount*(x) returns the number of bits of x equal to 1.

It can be easily seen, that in the worst case we will need $n_1 + n_2 - 1$ arcs, and in the best case $\max(n_1, n_2)$ arcs. We choose all the possible subsets i' of i and j' of j. The nodes determined by i' and j' can form an "independent subnetwork" only if the respective sums of capacities are equal. In this case we consider the worst case scenario, so we add $bitcount(i') + bitcount(j') - 1$ to the solution not containing these nodes. If we didn't find an independent subnetwork, it means, that the chosen arcs must form a connected graph, thus the minimal number of arcs is $n_1 + n_2 - 1$, so we can't get any better than the worst case scenario.

Let us analyze the performance of the proposed algorithm. The number of subproblems is $2^{n_1} \cdot 2^{n_2} = 2^{n_1+n_2}$, which in the worst case is 2^n. Thus the space complexity of our algorithm is $\Theta(2^n)$. To solve a subproblem (i,j) we need all the pairs (i',j'), such that i' is a subset of i and j' is a subset of j. We can codify any pair (i,i') with a sequence of length n_1 of ternary digits. A digit will be 0, if the respective node is not in i, 1 if it is in i but not in i' and 2 if it is in i' (and thus also in i). The same codification can be done for any (j,j') pair. Thus the number of steps performed by our algorithm is proportional to $3^{n_1} \cdot 3^{n_2} = 3^n$

6. Conclusions and future work

In this article we stated the debt's clearing problem, and analyzed some solving possibilities. An algorithm, that finds the optimal solution using the dynamic programming method was given, and its exponential running time and space complexity was proven.

We didn't analyze deeply the solving possibilities using nonlinear programming strategies. Also, the presented non optimal solutions, such as convex flow, could work in a high percentage of the cases, this possibility must be studied in the future. Approximation algorithms were not considered.

The biggest concern regarding this problem is its NP-completeness. The problem is a special instance of the known NP-complete problem of Minimum Edge-Cost Flow, and seems to be NP-complete too.

References

[1] Ravindra K. Ahuja and Thomas L. Magnanti and James B. Orlin, *Network flows*, Prentice-Hall, 1993.

[2] Bellman, Richard, *Dynamic Programming Treatment of the Travelling Salesman Problem*, Journal of the ACM, 1962.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms (2nd edition)*, 2001.

[4] Michael R. Garey, David S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[5] Held, Michael and Karp, Richard M, *A dynamic programming approach to sequencing problems*, Proceedings of the 1961 16th ACM national meeting, 1961.

[6] D. Luenberger, *Linear and nonlinear programming*, Addison-Wesley, New York, 1989.

Department of Computer Science, Babeş-Bolyai University, M. Kogălniceanu 1, 400084 Cluj-Napoca, Romania

E-mail address: `patcas@cs.ubbcluj.ro`