

# Settling Multiple Debts Efficiently: An Invitation to Computing Science

Tom VERHOEFF

*Faculty of Mathematics and Computing Science, Eindhoven University of Technology  
Den Dolech 2, NL-5612 AZ, Eindhoven, The Netherlands  
e-mail: t.verhoeff@tue.nl*

Received: June 2003

**Abstract.** I present and solve several problems related to the settling of multiple debts. The solutions are documented in much detail, with (bright) high-school students in mind. One of the variants has a simple solution, though it is not so easy to code concisely. Another variant is an elegant NP-hard problem.

The problem leads into important areas of mathematics and computing science, making it suitable as an invitation to these subjects.

**Key words:** computing science education, combinatorial optimization, balanced transportation problem, uncapacitated fixed-charge network flow, NP-completeness.

## 1. Introduction

Many informatics curricula for secondary education suffer from the ‘encyclopedia syndrome’. They try to cover every topic, but in view of the limited time can offer almost nothing about each topic. The author believes that it is much better to select a few topics and treat them enthusiastically in more depth. This article presents such a topic.

The problem I am about to present looks pretty innocent. It can be explained to anyone with some common (money) sense. Do not be misled, however. I found it embarrassingly instructive to fill in all details. I suggest that you work on the problem (Section 2) yourself before reading my analysis (Section 3) and solutions (Section 4 to Section 6).

## 2. The Problem

A group of friends lend each other money throughout the year. They carefully record each transaction and at the end of the year wish to settle their debts. How should they transfer money so as to settle all debts? How difficult is it to find an appropriate settling scheme? How efficient is that scheme? Try to minimize the number of transfers and the total amount transferred.

To be a bit more precise, let us number the  $N$  friends from 0 to  $N-1$ . At the end of the year, the record shows for each pair  $i, j$  with  $0 \leq i, j < N$  and  $i \neq j$  how much, in total, friend  $i$  owes to friend  $j$ , say amount  $a_{i,j} \geq 0$ .

For example, consider three friends 0, 1, and 2 with the following debts:

$$\begin{array}{c|c}
 i, j & a_{i,j} \\
 \hline
 0, 1 & 10 \\
 0, 2 & 10 \\
 1, 2 & 10 \\
 2, 0 & 20
 \end{array} \tag{1}$$

Only non-zero debts are listed. How to settle these debts efficiently?

The context where I encountered this problem is a joint practice for physical therapy. Each therapist treats her patients in a number of sessions. Afterwards she sends them an invoice and collects the fee. Occasionally, therapist  $i$  calls on therapist  $j$  to take over a session for a patient. The patient pays to therapist  $i$  the entire bill, for all sessions, including those sessions handled by therapist  $j$ . Thus, a debt is created from  $i$  to  $j$ . These debts are settled at the end of the year. In this context, there are amounts  $a_{i,j} \geq 0$  that therapist  $i$  collected for therapist  $j$ , with possibly  $i = j$ . The amounts  $a_{i,i}$ , however, are irrelevant for settling of the debts.

Read no further if you want to have a go at this problem yourself.

### 3. Analysis

For  $N \leq 2$ , the problem is trivial. In fact, for  $N = 0$  and  $N = 1$  there is no problem, because there are no debts to settle; thus, zero transfers is the optimal solution. For  $N = 2$ , the optimal solution is well known:

$$\begin{array}{ll}
 \text{Friend 0 pays } a_{0,1} - a_{1,0} \text{ to friend 1,} & \text{if } a_{0,1} > a_{1,0}. \\
 \text{No transfers are needed,} & \text{if } a_{0,1} = a_{1,0}. \\
 \text{Friend 1 pays } a_{1,0} - a_{0,1} \text{ to friend 0,} & \text{if } a_{0,1} < a_{1,0}.
 \end{array} \tag{2}$$

That is, the key is  $a_{0,1} - a_{1,0}$ .

How about  $N = 3$ ? Maybe you see how to do that right away. I didn't. The debts can be conveniently presented in a matrix, which I will call the **debt matrix**:

$$\begin{array}{c}
 \text{col.} \quad \text{col.} \quad \text{col.} \\
 \quad \quad 0 \quad 1 \quad 2 \\
 \text{row 0} \left[ \begin{array}{ccc} \cdot & a_{0,1} & a_{0,2} \\
 \text{row 1} & a_{1,0} & \cdot & a_{1,2} \\
 \text{row 2} & a_{2,0} & a_{2,1} & \cdot \end{array} \right] \tag{3}
 \end{array}$$

Amount  $a_{i,j}$  appears in row  $i$  and column  $j$ . Irrelevant entries  $a_{i,i}$  contain a dot ' $\cdot$ '. These debts can obviously be settled by at most six transfers and, in general, by  $N^2 - N = N(N-1)$  transfers:  $i$  pays  $a_{i,j}$  to  $j$  for all relevant pairs  $i, j$ . I call this the **trivial solution**. But one can do better, as already shown for  $N = 2$ . For  $i \neq j$ , the debts  $a_{i,j}$  and  $a_{j,i}$

are mirror images along the main (dotted) diagonal in the debt matrix. These two mutual debts can be settled by a single transfer involving the amount  $|a_{i,j} - a_{j,i}|$ . This way, all debts can be settled by at most three transfers and, in general, by  $\frac{1}{2}N(N-1)$  transfers. This I call the **paired solution**.

Is the paired solution optimal? No! Consider the matrix

$$\begin{array}{|ccc|} \hline \cdot & 1 & 1 \\ 0 & \cdot & 1 \\ 2 & 0 & \cdot \\ \hline \end{array} \quad (4)$$

which corresponds to example (1) above, disregarding a factor ten. The paired solution involves three 1-unit transfers in a cycle: friend 0 pays to friend 1, who pays to friend 2, who pays again to friend 0. Observe that each friend also receives 1 unit. Consequently, there is no need to transfer any money at all.<sup>1</sup>

Apparently, the total amounts  $p_i$  to be paid by  $i$  are of importance:

$$p_i = \sum_k a_{i,k}, \quad (5)$$

and the total amounts  $r_j$  to be received by  $j$ :

$$r_j = \sum_k a_{k,j}. \quad (6)$$

In terms of the debt matrix,  $p_i$  is the sum of the amounts in row  $i$ , and  $r_j$  is the sum of the amounts in column  $j$ :

$$\begin{array}{|ccc|} \hline \cdot & a_{0,1} & a_{0,2} \\ a_{1,0} & \cdot & a_{1,2} \\ a_{2,0} & a_{2,1} & \cdot \\ \hline \end{array} \begin{array}{l} p_0 \\ p_1 \\ p_2 \end{array} \quad (7)$$

$$\begin{array}{ccc} r_0 & r_1 & r_2 \end{array}$$

All that matters for friend  $i$  is her **balance**  $b_i$  at the end of the year:

$$b_i = p_i - r_i. \quad (8)$$

If  $b_i > 0$ , then she borrowed more than she lent out, and she needs to pay some others. The question remains whom to pay how much. If  $b_i = 0$ , then she borrowed as much as she lent out, and she is even. If  $b_i < 0$ , then she lent out more than she borrowed, and

---

<sup>1</sup>Provided these friends are just interested in the total amount of money and not, for instance, in the finger prints of their debtors.

she needs to receive from some others. In example (4), we have the following row and column sums:

$$\begin{array}{ccc|c} \cdot & 1 & 1 & 2 \\ 0 & \cdot & 1 & 1 \\ 2 & 0 & \cdot & 2 \\ \hline 2 & 1 & 2 & \end{array} \quad (9)$$

and, hence, all  $b_i = 0$ , that is, there effectively are no debts to settle.

Before giving an optimal pay-back scheme for  $N = 3$ , I make the following observations. The grand total amount to be paid equals the grand total amount to be received:

$$\sum_i p_i = \sum_i \sum_k a_{i,k} = \sum_j \sum_k a_{k,j} = \sum_j r_j. \quad (10)$$

Consequently, the grand total balance equals zero:

$$\sum_i b_i = \sum_i (p_i - r_i) = \sum_i p_i - \sum_i r_i = 0. \quad (11)$$

Hence, there exists a friend with a positive balance, if and only if there exists a friend with a negative balance:

$$(\exists_i b_i > 0) \equiv (\exists_i b_i < 0). \quad (12)$$

For  $N = 3$ , there are only four possible combinations of the three  $b_i$ -signs (+, 0, or -), if we abstract from friend identity, that is, from permutations of friends:

$$\begin{array}{cccc} 0 & 0 & + & + \\ 0 & + & + & - \\ 0 & - & - & - \end{array} \quad (13)$$

The first combination (all 0, the leftmost column in (13)) is equivalent to  $N = 0$  and  $N = 1$ , requiring zero transfers. The second combination (one 0) is equivalent to  $N = 2$ , requiring one transfer. The third combination (two +) is settled by two transfers: from each of the two positive  $b_i$  to the negative  $b_i$ . The fourth combination (two -) is also settled by two transfers: from the positive  $b_i$  to each of the two negative  $b_i$ . Note that the latter two cases are related by sign reversal. In general, reversing the directions of all transfers in a solution for some case, yields a solution to the case with all balance signs reversed. Thus, we can also abstract from sign reversal.

The optimal transfers for each of the four sign combinations are depicted in Fig. 1. Each dot represents a friend, each arrow a transfer. For  $N = 3$ , the worst case apparently involves two transfers.

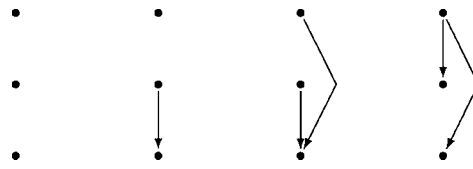


Fig. 1. The four optimal transfer patterns for  $N = 3$ .

Next, consider case  $N = 4$ . These seven combinations of balance signs can occur:

$$\begin{array}{ccccccc}
 0 & 0 & 0 & 0 & + & + & + \\
 0 & 0 & + & + & + & + & - \\
 0 & + & + & - & + & - & - \\
 0 & - & - & - & - & - & -
 \end{array} \tag{14}$$

The four leftmost combinations reduce to cases with  $N < 4$ . Of the remaining three combinations, the two with a unique positive or negative balance are optimally solved by three transfers, as depicted in Fig. 2. (Why?) They are related by sign reversal.

Finally, the case with two positive and two negative balances requires further analysis. Without loss of generality, assume

$$b_0 \geq b_1 > 0 > b_2 \geq b_3. \tag{15}$$

Distinguish three cases depending on the sign of  $b_0 + b_3$  (also see Fig. 3):

- $b_0 = -b_3$ : hence  $b_1 = -b_2$ , on account of (11); the optimal solution involves just two transfers, from 0 to 3, and from 1 to 2;
- $b_0 > -b_3$ : hence  $b_1 < -b_2$ ; the optimal solution involves three transfers, from 0 to 2 and 3, and from 1 to 2;
- $b_0 < -b_3$ : hence  $b_1 > -b_2$ ; the optimal solution involves three transfers, from 0 to 3, and from 1 to 2 and 3.

For  $N = 4$ , the worst case apparently involves three transfers.

How about  $N > 4$ ? Can the debts always be settled in at most  $N - 1$  transfers?

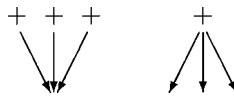


Fig. 2. Two of the optimal transfer patterns for  $N = 4$ .

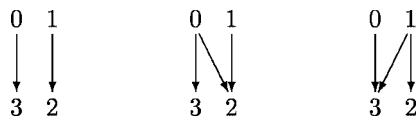


Fig. 3. Three other optimal transfer patterns for  $N = 4$ .

**4. Practical Solutions**

There is a practical solution that requires at most  $N$  transfers. It is used in the partnership of therapists mentioned at the end of Section 2: All fees are collected in a **central account**, and at the end of the year, therapist  $i$  collects  $r_i$  from the central account (this includes  $a_{i,i}$ ).

The  $N$  friends can do this as follows. First, all friends  $i$  with  $b_i > 0$  put  $b_i$  on the table (the central account). Next, all friends  $i$  with  $b_i < 0$  take  $-b_i$  from the table. There is no surplus or deficit on the table, on account of (11):

$$(\sum i : b_i > 0 : b_i) = (\sum i : b_i < 0 : -b_i). \tag{16}$$

A slight modification of this scheme uses a **friend as central account**, say friend  $j$ . All others with  $b_i > 0$  transfer  $b_i$  to  $j$ , and then those with  $b_i < 0$  collect  $-b_i$  from  $j$ . Friend  $j$  has to provide  $b_j$  herself if  $b_j > 0$ , or gains  $-b_j$  if  $b_j < 0$ . This scheme requires no more than  $N-1$  transfers.

Is the scheme with a friend as central account optimal? No, not in general! It might involve unnecessary transfers, or unnecessarily large amounts transferred. For example, if someone with  $b_i = 0$  plays the role of central account, both inefficiencies will be the case. Consider  $N = 3$  with  $b_0 = 0 < b_1 = -b_2$ . Using friend 0 as central account involves two transfers instead of one, and, in total, twice as much money changes hands as necessary. Even if the central account is with someone whose  $b_i \neq 0$ , it need not be optimal. Consider again  $N = 3$ , now with  $b_0, b_1 > 0$  and  $b_2 < 0$ . If the central account is with friend 0, then the number of transfers is indeed optimal (two), but the total amount transferred would be unnecessarily large ( $b_0 + 2b_1$  instead of  $b_0 + b_1$ , see left in Fig. 4). In this case, using friend 2 as center works optimally.

For  $N = 4$  with  $0 < b_0 = -b_3$  and  $0 < b_1 = -b_2$ , any center is suboptimal. The optimal solution (see left in Fig. 3) involves two ‘independent’ transfers. Introduction of a center makes all transfers ‘dependent’. Thus, one of the independent transfers of the optimal solution is then handled indirectly: the amount is transferred twice by way of the center. This gives rise to an unnecessary transfer and an unnecessarily large total amount transferred. How can one construct better solutions?

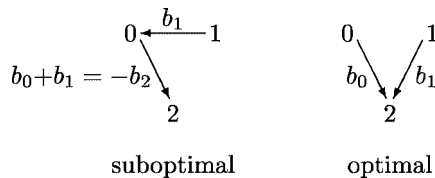


Fig. 4. Two related transfer patterns for  $N = 3$ .

## 5. Transformed Solutions

There is another way to see that no more than  $N-1$  transfers are required; a way that eventually leads to optimal solutions. I have already been drawing pictures with friends and transfers. Let us formalize such a picture as a mathematical graph, more precisely a **labeled directed graph**. A directed graph is a pair  $(P, A)$ , where  $P$  is a set of **points** and  $A$  is a set of **arrows**. Each arrow starts in a point and ends in a point. The points and the arrows can be labeled, for instance, with numbers. The statement ‘there is an arrow from point  $i$  to point  $j$  labeled with amount  $a$ ’ is denoted by

$$i \xrightarrow{a} j. \quad (17)$$

Our problem can be rephrased as the following graph problem. We are given  $N$  points (friends), namely the numbers 0 to  $N-1$ . Each point  $i$  is labeled by a number  $b_i$  (balance), such that (also see (11)):

$$\sum_i b_i = 0. \quad (18)$$

We are to find a set of arrows (transfers) labeled by positive numbers (amounts), such that all balances are evened out:

$$\text{For every point } i, \text{ the sum of the labels on its outgoing arrows} \\ \text{minus the sum of the labels on its incoming arrows equals } b_i. \quad (19)$$

This property, which I call the **balancing relation**, can be formalized as follows. Let  $in_i$  be the sum of the labels on the incoming arrows of node  $i$ , and, similarly,  $out_i$  the sum of the labels on the outgoing arrows, that is,

$$in_i = (\sum j, t : j \xrightarrow{t} i : t), \quad (20)$$

$$out_i = (\sum j, t : i \xrightarrow{t} j : t). \quad (21)$$

The balancing relation (19) is then captured by

$$b_i = out_i - in_i \quad \text{for all } i. \quad (22)$$

A graph with this property is called a **transfer graph** for the numbers  $b_i$ . The sum of the numbers on all arrows (total amount transferred) is called the **weight** of the graph. We are looking for a transfer graph with minimum number of arrows and minimum weight. (Why is this problem not solvable when (18) does not hold?)

The trivial solution, presented in Section 3, consists of all arrows

$$i \xrightarrow{a_{i,j}} j \quad (23)$$

with  $a_{i,j} > 0$ . In general, it is not optimal.

I now investigate some graph transformations to reduce the number of arrows and/or the weight, while preserving the balancing relation (19). In the remainder of this section,  $(P, A)$  is a transfer graph for  $b_i$  ( $0 \leq i < N$ ).

5.1. *Directed-Cycle Transformation*

Consider a directed cycle in the graph  $(P, A)$  involving  $k$  points and  $k$  arrows:

$$p_0 \xrightarrow{t_0} p_1 \xrightarrow{t_1} p_2 \cdots p_{k-1} \xrightarrow{t_{k-1}} p_0. \tag{24}$$

Let  $m$  be the minimum amount on the arrows in the cycle, that is,

$$m = (\downarrow i : 0 \leq i < k : t_i). \tag{25}$$

Decrease each  $t_i$  in the cycle by  $m$ , yielding

$$p_0 \xrightarrow{t_0-m} p_1 \xrightarrow{t_1-m} p_2 \cdots p_{k-1} \xrightarrow{t_{k-1}-m} p_0. \tag{26}$$

Arrows whose amount has become 0 are omitted. There is at least one such arrow, hence the cycle is ‘broken’. This transformation preserves the balancing relation (19), because for each of the affected points  $i$  both the outgoing sum  $out_i$  and incoming sum  $in_i$  are decreased by  $m$ , leaving their difference invariant. Consequently, the resulting graph is a transfer graph for the same problem, and it has fewer arrows and its weight has been decreased by  $k * m$ .

As an example, consider the problem given by (1). Its trivial solution is depicted by the transfer graph on the left in Fig. 5. I have omitted the  $b_i$ -labels, to avoid cluttering the picture.

First consider the directed cycle  $0 \xrightarrow{1} 2 \xrightarrow{2} 0$ , whose reduction by 1 yields  $2 \xrightarrow{1} 0$ , as depicted in the middle of Fig. 5. This corresponds to the paired solution. In general, reducing all cycles of length two in the trivial solution yields the paired solution. Next, observe that what remains is a directed cycle of length three, which disappears altogether under the directed-cycle transformation, because all amounts equal the minimum amount. We already knew the resulting solution shown on the right.

Repeated application of this transformation yields a transfer graph without directed cycles, a so-called **directed acyclic graph**, or **dag**.

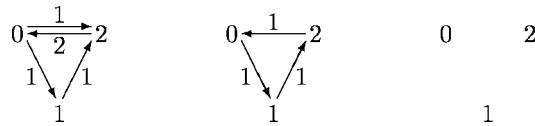


Fig. 5. A transfer graph and two transformations for  $N = 3$ .



The resulting dag depends on the order in which cycles are broken, as illustrated by the following example with  $N = 4$  and debt matrix

$i \setminus j$	0	1	2	3	$p_i$	$i$	$p_i - r_i = b_i$
0	·	3	0	6	9	0	$9 - 3 = +6$
1	1	·	5	0	6	1	$6 - 7 = -1$
2	2	4	·	0	6	2	$6 - 8 = -2$
3	0	0	3	·	3	3	$3 - 6 = -3$
$r_j$	3	7	8	6			

(27)

The transfer graph for the paired solution is shown on the left in Fig. 6. Again we have omitted the  $b_i$ . It has five arrows and weight 14. There are two directed cycles of length three. The one allowing the biggest reduction is at the bottom:  $0 \xrightarrow{6} 3 \xrightarrow{3} 2 \xrightarrow{2} 0$ . Reduction by 2 yields a dag, shown second from the left in the figure. It has four arrows and weight 8.

The cycle at the top is  $0 \xrightarrow{2} 1 \xrightarrow{1} 2 \xrightarrow{2} 0$ . Reduction by 1 yields the transfer graph in the middle of Fig. 6. There is still a cycle at the bottom:  $0 \xrightarrow{6} 3 \xrightarrow{3} 2 \xrightarrow{1} 0$ . Reduction by 1 yields a dag, shown second from the right. It has three arrows and weight 8.

We already know the optimal transfer graph for  $N = 4$  and type +--- from Section 3. It is shown on the right, having three arrows and weight 6. Note that a **lower bound on the weight** of a transfer graph is

$$(\sum i : b_i > 0 : b_i), \tag{28}$$

because all these balances need to be evened out. On account of (16), it equals

$$\frac{1}{2} \sum_i |b_i|. \tag{29}$$

Concerning the number of arrows, the following can be remarked. Each positive balance must have at least one outgoing arrow and each negative balance must have at least one incoming arrow. Therefore, a **lower bound on the number of arrows** is

$$(\#i :: b_i > 0) \uparrow (\#i :: b_i < 0), \tag{30}$$

where  $(\#i :: B.i)$  is the number of  $i$ 's for which  $B.i$  holds, and  $a \uparrow b$  is the maximum of  $a$  and  $b$ . The optimal graph in Fig. 6 attains these lower bounds.

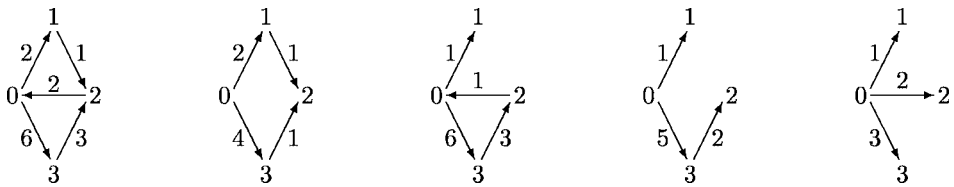


Fig. 6. A transfer graph and four transformations for  $N = 4$ .

5.2. Directed-Path Transformation

The directed-cycle transformation is not enough to reduce the transfer graph to at most  $N-1$  arrows. Here is another transformation.

Consider two ‘parallel’ directed paths with the same start point  $p_0$  and end point  $p_k$ :

$$p_0 \xrightarrow{t_0} p_1 \xrightarrow{t_1} p_2 \cdots p_{k-1} \xrightarrow{t_{k-1}} p_k, \tag{31}$$

involving  $k$  arrows and

$$p_0 \xrightarrow{t_{k+\ell-1}} p_{k+\ell-1} \cdots p_{k+2} \xrightarrow{t_{k+1}} p_{k+1} \xrightarrow{t_k} p_k, \tag{32}$$

involving  $\ell$  arrows. Let  $m$  be the minimum amount on the arrows in the first path, that is,

$$m = (\downarrow i : 0 \leq i < k : t_i). \tag{33}$$

Decrease each  $t_i$  in the first path by  $m$ , and increase each  $t_i$  in the second path by  $m$ . Arrows whose amount has become 0 are omitted (there is at least one such arrow). This transformation preserves the balancing relation (why?) and the resulting graph is a transfer graph for the same problem. It has fewer arrows and its weight has been ‘decreased’ by  $(k - \ell) * m$ .

Note that the directed-cycle transformation discussed earlier is a special case of this transformation by taking  $p_0 = p_k$  and  $\ell = 0$ .

In the example above for  $N = 4$ , the second graph from the left in Fig. 6 contains no directed cycles, but it does contain two directed paths from 0 to 2. It is shown again on the left in Fig. 7. Decreasing the upper path by 1 and increasing the lower path by 1 yields the second graph from the left (also shown in Fig. 6). This graph can be reduced to the optimal graph on the right by first applying the transformation in ‘reverse’. The second graph from the right can be transformed to its left neighbor by decreasing the upper path from 0 to 2 by 1 and increasing the lower path by 1. Thus, these two graphs solve the same problem. That same graph can also be transformed to its right neighbor, the optimal solution, by decreasing the lower path by 1 and increasing the upper path by 1.

5.3. Arbitrary Cycle Transformation

The ‘reverse’ application of transformations hints at a further generalization. First of all, there is no reason to decrease a directed-cycle by exactly the minimum amount that occurs

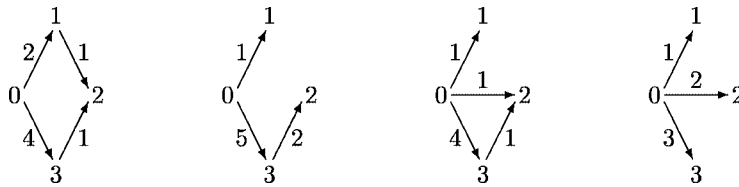


Fig. 7. Four related transfer graphs for  $N = 4$ .

on any of its arrows. Decreasing by less also works fine: it reduces the weight but not the number of arrows. However, decreasing by more than the minimum can also be made to work. This still preserves the balancing relation (19). The only trouble is the introduction of arrows labeled with negative amounts. Such arrows can be ‘eliminated’ by reversing their direction and sign:

$$p \xrightarrow{t} q \text{ is equivalent to } p \xleftarrow{-t} q. \tag{34}$$

Furthermore, any pair of points  $p, q$  not connected by an arrow may be connected by an arrow labeled with amount 0:

$$p \quad q \text{ is equivalent to } p \xrightarrow{0} q. \tag{35}$$

In this new light, the directed-path transformation is actually a special case of the directed-cycle transformation: In the second path, reverse the arrows and the signs on their amounts. Together with the first path this yields a *directed* cycle. Next, decrease all amounts by  $m$ , and, finally, reverse arrows and amounts in the second path again. You do not explicitly have to create a directed cycle by reversing arrows, as long as you keep in mind that when amounts on an *arbitrary* cycle are *decreased*, the amounts on arrows pointing in the reverse direction need to be *increased*.

As an example consider the transfer graph for  $N = 4$  depicted on the left in Fig. 8. The balances (not shown) are  $+3, +7, -4, -6$ . Neither the directed-cycle nor the directed-path transformation applies, but the graph consists of one (undirected) cycle. Walking around in the direction of the arrow with amount 1, decreasing all forward arrows by 1 and increasing all reverse arrows by 1, yields the graph in the middle. It has one fewer arrow and the same weight. Walking around in the direction of the arrow with amount 2, decreasing all forward arrows by 2 and increasing all reverse arrows by 2, yields the graph shown on the right. Both these resulting graphs are optimal, as we know from Section 3. By the way, this example also illustrates that the optimal solution need not be unique.

This transformation applies to any cycle in the transfer graph, possibly including zero-labeled arrows and arrows in reverse. Consequently, any cycle of arrows with non-zero amounts can be ‘broken’: just decrease along the cycle by an amount that occurs on some forward arrow. Remember to *increase* reverse arrows, to drop out zero-labeled arrows, and to reverse arrows that got negative amounts. This way any transfer graph can be put into a form without cycles. An acyclic graph on  $N$  points has no more than  $N-1$  arrows, because if you walk along  $k$  arrows (ignoring their direction) without encountering the same point twice, you have seen  $k+1$  distinct points. Thus, I have shown, once more, how to settle the debts by no more than  $N-1$  transfers. Note that the solution with a friend as central account has an acyclic transfer graph.

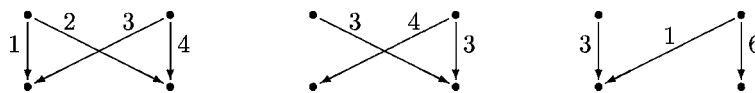


Fig. 8. Three related transfer graphs for  $N = 4$ .

### 6. Optimized Solutions

A transfer graph with no more than  $N-1$  arrows is not a bad solution, certainly when compared to the trivial solution that may have as many as  $N(N-1)$  arrows. But it is not necessarily optimal.

I mentioned two items to minimize: the number of arrows (transfers) and the weight (total amount transferred). Is it the case that minimality of one item implies minimality of the other? No! You can see in Fig. 6 that transfer graphs with the minimum number of arrows are not necessarily of minimum weight, and in Fig. 8 that transfer graphs with minimum weight do not necessarily have the minimum number of arrows.

#### 6.1. Minimum Weight

In a transfer graph, any directed path of length two or more can be shortened. Consider a directed path of length two, as shown on the left in Fig. 9, from  $p$  via  $q$  to  $r$  with amounts  $t$  and  $u$ . Let  $m$  be the minimum of  $t$  and  $u$ , that is,  $m = t \downarrow u$ . Imagine a zero-labeled arrow from  $p$  to  $r$ , and decrease the cycle  $p, q, r$  by  $m$ , as shown in the second graph from the left (the zero-labeled arrow was increased because it is reversed). At least one of the resulting arrows gets a zero label. Which one(s), depends on how  $t$  and  $u$  compare (see the three graphs on the right). In all cases the weight has decreased by  $m$ . In the case of  $t = u$ , the number of arrows has decreased by one, whereas in the other cases it has remained the same.

By applying this transformation repeatedly, a transfer graph is obtained in which each point has either no outgoing arrows or no incoming arrows. Such a graph is known as a (directed) **bipartite graph**, in which the points are partitioned into two groups, and all arrows go from one group to the other and not back or within groups. Points with positive balance ( $b_i > 0$ ) have outgoing arrows only, points with negative balance ( $b_i < 0$ ) have incoming arrows only. Points with zero balance ( $b_i = 0$ ) have no arrows at all. All graphs in Figs. 1, 2, 3, and 8 are bipartite. Observe that a transfer graph with minimum weight must be bipartite, because the weight of a graph that is not bipartite (as defined above) contains a path of length two and can thus be reduced in weight.

The weight of a bipartite transfer graph is

$$(\sum i : b_i > 0 : b_i) \tag{36}$$

because each arrow starts in a point with positive balance and all balances are evened out. On account of lower bound (28), this is the least possible weight. Thus, *all* bipartite

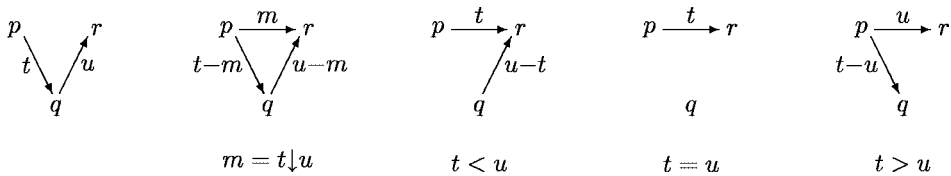


Fig. 9. Shortening a path and decreasing the weight.

transfer graphs have minimum weight. Together with the observation above, it has been shown that the solutions of minimum weight are exactly the bipartite solutions.

Here is an **algorithm to construct a bipartite transfer graph**:

1. Start with the empty graph: just  $N$  points and no arrows.
2. If all balances  $b_i$  are zero, no transfers are needed and the graph is ready.
3. Otherwise, on account of  $\sum_i b_i = 0$ , there exist two points  $i$  and  $j$  with  $b_i > 0 > b_j$ .
4. Let  $m$  be the minimum of  $b_i$  and  $-b_j$ , and add the arrow  $i \xrightarrow{m} j$  to the graph. Decrease  $b_i$  by  $m$  and increase  $b_j$  by  $m$ .
5. Solve the remaining problem recursively by repeating from Step 2.

Upon termination, the resulting transfer graph is bipartite (only arrows from positive to negative balances) and it evens out all balances. Whenever a transfer  $i \xrightarrow{t} j$  is added, the balance at  $i$  or  $j$  is decreased to zero (or both are decreased to zero). Hence, no transfer cycles are created. Because an acyclic graph on  $N$  points has at most  $N-1$  arrows (see end of Section 5), the graph contains no more than  $N-1$  transfers. This also shows that the algorithm indeed terminates. In Section 8, I present a Pascal program for this algorithm.

6.2. *Minimum Number of Arrows*

By now it is clear that points with balance zero can be omitted, because they need not be involved in any transfers. For if they would be then they would have incoming *and* outgoing arrows, which can be removed by path shortening. In the remainder, I assume  $b_i \neq 0$ .

The algorithm presented above produces an acyclic bipartite graph, which minimizes the weight. We have already seen in Fig. 8 that minimum-weight graphs need not have a minimum number of arrows. However, in that example, there is still a cycle that can be broken. But even an acyclic bipartite graph need not be optimal, as illustrated by Fig. 10 for  $N = 4$  and balances  $+5, +3, -5, -3$ .

The algorithm presented above is **greedy**, because the *greatest* possible amount is transferred between every selected pair  $i, j$  with  $b_i > 0 > b_j$ . There are various ways to modify it, in an attempt to minimize the number of transfers. One approach is to make it less greedy. But then how should one choose an appropriate amount to transfer? Another approach – which I analyzed – is to select the pairs  $i, j$  in a careful order.

My first attempt considers pairs  $i, j$  with  $b_i > 0 > b_j$  in decreasing order of absolute value, that is, to **match the greatest balance differences first**. That would indeed avoid the suboptimal graph in Fig. 10, because balances 5 and  $-5$  (difference 10) would be

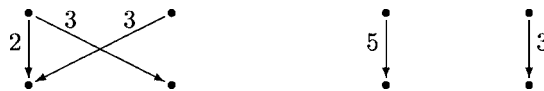


Fig. 10. Related transfer graphs for  $N = 4$ .

matched first. This works well for  $N \leq 4$ , but not in general: Consider the case with  $N = 5$  and balances  $+9, +8, -3, -6, -8$ . My proposal would first match  $+9$  and  $-8$  (difference 17), yielding a graph with four transfers instead of the optimal three (verify!).

Fig. 11 shows the four classes of acyclic bipartite graph for  $N = 5$ , without isolated points, modulo sign reversal. For the type  $++---$ , the three-transfer solution is preferred. It can only occur if there are two opposite balances that can cancel each other in one transfer.

My second attempt considers pairs  $i, j$  with  $b_i > 0 > b_j$  in order of increasing sum  $b_i + b_j$ , that is, to **match closest opposite balances first**. That would indeed avoid suboptimal graphs for the cases with  $N = 5$ . In the example above, balances  $+8$  and  $-8$  (sum 0) would be matched first. This works well for  $N \leq 5$ , but not in general. A counterexample for  $N = 6$  is provided by the balances  $+9, +8, +2, -4, -5, -10$ . Matching  $+9$  and  $-10$  results in a solution with five transfers instead of the optimal four.

Is it always best to **match opposite balances**? Consider a solution where two points  $i$  and  $j$ , having opposite balances  $b_i = -b_j = B > 0$ , are *not* connected by an arrow. There must be some arrows going from  $i$ , say  $p > 0$  of them in total, and some other arrows going to  $j$ , say  $q > 0$  in total. Note that the sum of the amounts on the  $p$  arrows from  $i$  equals the sum of the amounts on the  $q$  arrows to  $j$ , both sums being equal to  $B$ . If the  $p+q$  arrows are removed and an arrow  $i \xrightarrow{-B} j$  is added, then we still have to settle the remaining imbalance at the  $p$  sources and  $q$  sinks. Since these imbalances are opposite, they can be evened out by at most  $p+q-1$  arrows. Thus, the total number of arrows in the new solution involving  $i \xrightarrow{-B} j$  is at most the number of arrows in the original solution.

**Exercise:** Give an example showing that opposite matching is not necessary for optimality.

Matching opposites, however, never prohibits optimality, and an algorithm to construct an optimal transfer graph could start by eliminating all opposites. As remarked before and evident from Fig. 11, this approach takes care of all cases with  $N \leq 5$ . But it leaves the harder part of the problem unsolved: What to do when there are no opposites?

**Exercise:** Give an example showing that it is not always best to ‘match triples’.

Fig. 12 shows the eleven classes of acyclic bipartite graphs for  $N = 6$ , without isolated points, modulo sign reversal. It seems evident from these graphs that acyclic graphs with fewer arrows consist of more ‘disconnected clusters’, and vice versa. There is an easy explanation for this, but it calls for more graph terminology.

A graph is called **connected** if there is a path between every pair of points. The direction of arrows in a path is considered irrelevant. Every graph consists of a number of

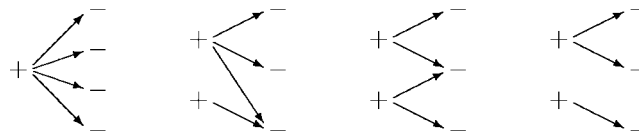


Fig. 11. Acyclic bipartite graphs for  $N = 5$ .

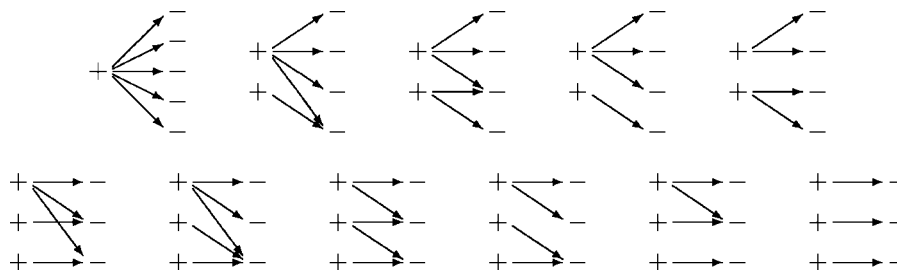


Fig. 12. Acyclic bipartite graphs for  $N = 6$ .

connected subgraphs (possibly one) that are not connected to each other. These connected subgraphs are called the **components** of the graph. For example, in Fig. 12 there are six graphs with one component, four with two components, and one with three.

All components of an acyclic graph are acyclic. A connected acyclic graph is called a **tree**. It has one more point than arrows; that is, if it has  $k$  points, then it has  $k-1$  arrows. Adding an arrow would introduce a cycle, and upon removal of an arrow the graph would no longer be connected. Therefore, an acyclic graph is a collection of trees, also known as a **forest**. The number of arrows in a forest equals the number of points minus the number of components (trees). Consequently, minimizing the number of arrows in an acyclic graph, is equivalent to **maximizing the number of components**.

The sum of the balances in a component of a transfer graph equals zero, because each arrow  $i \xrightarrow{t} j$  adds  $t$  to the balance sum via  $out_i$  and subtracts  $t$  from the balance sum via  $in_j$ . Therefore, to minimize the number of arrows one needs to find a partition of the points such that each part has balance sum zero and the number of parts is maximized.

Consider the special case of this problem, where  $N-2$  distinct balances are positive and two are negative, say  $b_0, b_1 < 0$ . On account of lower bound (30) on the number of arrows in a transfer graph, this special case requires at least  $N-2$  transfers (each of the positive balances needs an outgoing arrow). Since no more than  $N-1$  transfers are needed, the question is whether  $N-2$  is feasible. In terms of partitioning, this boils down to the existence of a partition of the balances into two parts, both with balance sum zero. Obviously, each part must contain a negative balance. Therefore, the problem is to find a subset of the positive balances  $b_i$  (those with  $2 \leq i < N$ ) whose sum equals  $-b_0$ , the complement then automatically sums to  $-b_1$ . This problem is equivalent to the **subset-sum problem**, sometimes also called the (simplified) **knapsack problem** Garey and Johnson, 1979, SP13:

For a given positive integer  $K$  and set  $S$  of items  $x$  with positive integer size  $s(x)$ , does there exist a subset  $R$  of  $S$  whose total size  $\sum_{x \in R} s(x)$  equals  $K$ ?

Here,  $K$  is the size of the knapsack,  $S$  contains the items to pack, and  $s$  gives their sizes. The question is whether the knapsack can be filled exactly with a suitable selection  $R$  of the items.

It is easy to verify that a given subset  $R$  has indeed total size  $K$ . However, deciding whether a suitable subset  $R$  exists is believed to be a hard problem. As far as we know,

one can, in general, not do much better than trying out all possibilities. The number of possibilities is an **exponential** function of the number of items in the input set  $S$ : there are  $2^n$  subsets of an  $n$ -element set. If you discover a shortcut, such as a solution requiring no more work than a **polynomial** function of  $n$ , then you'll be famous instantly.

Finding a transfer graph with minimum number of arrows is at least as difficult as solving the subset-sum problem.

## 7. Variations

I have covered a number of issues when settling multiple debts:

1. generality (arbitrary number of lenders),
2. simplicity and practical feasibility,
3. minimized total amount transferred,
4. minimized total number of transfers, and
5. mathematical complexity of obtaining a solution.

There are, however, many other issues that might be considered, such as

1. charging interest on loans,
2. handling exchange rates for multiple currencies, and
3. dealing with distrust among the lenders.

I leave these matters as challenges to the reader.

## 8. Programs

There are many ways to present the problem of efficiently settling multiple debts as a programming problem. In the preceding section, it turned out that the problem of finding a minimum weight transfer graph is relatively easy, whereas that of finding a transfer graph with minimum number of arrows is hard.

The input to the program can be given in various ways:

1. A list of loans  $a_{i,j}$  from  $i$  to  $j$ . This is the format used in (1). More precisely:

The input to your program consists of a text file. On the first line is an integer  $N$ ,  $N_{\min} \leq N \leq N_{\max}$ , the number of friends. On the second line is an integer  $K$ ,  $K_{\min} \leq K \leq K_{\max}$ , the number of loans. On each of the  $K$  following lines are three integers  $i, j, a$  with  $0 \leq i, j < N$ ,  $i \neq j$ , and  $0 < a = a_{i,j} \leq A_{\max}$ . The pairs  $i, j$  that appear are distinct and ordered lexicographically.

Either of the two conditions on the pairs  $i, j$  (distinctness, lexicographic order) can be dropped. When dropping the distinctness condition, the input can be interpreted to give the raw data of all the loaning transactions that took place over the year and that need to be settled at the end of the year. Zero could be allowed as loan.

2. A debt matrix  $a_{i,j}$ . This is the format used in (4). More precisely:

The input to your program consists of a text file. On the first line is an integer  $N$ ,  $N_{\min} \leq N \leq N_{\max}$ , the number of friends. On each of the  $N$  following lines are  $N-1$  integers.



Line  $i+2$  contains, in order of increasing  $j$ , the debts  $a_{i,j}$  with  $i \neq j$ , and  $0 \leq a_{i,j} \leq A_{\max}$ .

'Debts'  $a_{i,i}$  might be included as well.

### 3. A list of balances $b_i$ . More precisely:

The input to your program consists of a text file. On the first line is an integer  $N$ ,  $2 \leq N \leq N_{\max}$ , the number of friends. On each of the  $N$  following lines is an integer  $b$  with  $0 \leq b = b_i \leq B_{\max}$ .

Zero balances could be excluded.

N.B. Lower bounds  $N_{\min}$  and  $K_{\min}$ , and upper bounds  $N_{\max}$ ,  $K_{\max}$ ,  $A_{\max}$ , and  $B_{\max}$  need to be chosen carefully. Input format 3 takes away half of the problem, because for formats 1 and 2 the relevance of the balances still needs to be discovered.

The output of the program must list all transfers needed to settle the input debts. More precisely:

The output of your program is a text file. Each line contains three numbers  $i, j, t$ , encoding a transfer of amount  $t$  from  $i$  to  $j$ , with  $0 \leq i, j < N$ ,  $i \neq j$ , and  $0 < t$ , such that all pairs  $i, j$  are distinct and all debts in the input file are settled. The order of the lines is irrelevant.

There are various ways to impose extra constraints:

1. Minimize the total amount transferred (weight), that is, the sum of all  $t$ -values in the output.
2. Minimize the number of transfers, that is, the number of lines in the output.
3. Minimize the number of transfers and the total amount transferred. This is the original problem of Section 2.

Furthermore, the score of the program can depend on how close the output approximates the minimum. For the second and third form (minimizing the number of transfers), it is even possible to give several non-secret input files and only ask for corresponding output files.

#### 8.1. Input Processing

It is straightforward how input formats 1 and 2 can be transformed into each other and into input format 3. Here is a piece of Pascal program for reading input format 1 and storing it as an array of balances (format 3):

**program** *GettingEven*;

**const**

$Nmin = 2$ ; { lower bound on number of friends }

$Nmax = 30000$ ; { upper bound on number of friends }

**type**

$NFriendsRange = Nmin .. Nmax$ ; { range for number of friends }

$Friend = 0 .. Nmax-1$ ; { actually  $0 .. N-1$ , where  $N$  is number of friends }  
 $Balances = \mathbf{array} [ Friend ] \mathbf{of} Integer$ ; {  $b[k] = \text{balance of friend } k, 0 \leq k < N$  }

**procedure** *ReadInput* ( **var**  $N: NFriensRange$ ; **var**  $b: Balances$  );

{ **globals:** *input* }  
 { **pre:** *input* contains a recording of loans in format 1 }  
 { **post:**  $N$  and  $b$  are defined according to the input }

**var**

$i, j: Friend$ ; { friend  $i$  owes to  $j$  }  
 $a: Integer$ ; { amount of loan }  
 $K: Integer$ ; { number of loans (input) }  
 $h: Integer$ ; { to traverse the  $K$  loans }

**begin**

$readln(N)$   
 ;  $readln(K)$   
 ; **for**  $i := 0$  **to**  $N-1$  **do**  $b[i] := 0$   
 { **invariant:**  $\sum_{0 \leq k < N} b[k] = 0$  }  
 ; **for**  $h := 1$  **to**  $K$  **do begin**  
    $readln(i, j, a)$   
 ;  $b[i] := b[i] + a$   
 ;  $b[j] := b[j] - a$   
**end** { for  $h$  }  
**end;** { *ReadInput* }

## 8.2. Minimum Total Amount Transferred

In Section 6.1, I have presented a greedy algorithm to construct a bipartite transfer graph that minimizes the total amount transferred. The following procedure expresses it concisely in Pascal, like a poem.

**procedure** *WriteOutput* (  $N: NFriensRange$ ;  $b: Balances$  );

{ **globals:** *output*; **spec.var:**  $B$  }  
 { **pre:**  $b = B, \sum_{0 \leq k < N} B[k] = 0$  }  
 { **post:** *output* contains a list of transfers to settle all input debts  $B$ , }  
 { minimizing the total amount transferred }

**var**

$i, j: 0 .. Nmax$ ; { to traverse friends  $0..N-1$ ,  $N$  acts as sentinel }  
 $m: Integer$ ; { auxiliary to compute minimum }  
 {  $c: Balances$ ; ghost variable, initially  $\forall_k c[k] = 0$  }

**begin**

$i := 0$   
 ;  $j := 0$

```

{invariant:  $\sum_k b[k] = 0, \forall_{k < i} b[k] \leq 0, \forall_{k < j} b[k] \geq 0, b + c = B, \text{ output settles } c$ 
variant function:  $2N - i - j + (\# k :: b[k] \neq 0)$  }
; while ( $i \neq N$ ) and ( $j \neq N$ ) do begin
  if  $b[i] \leq 0$  then
     $i := i + 1$ 
  else if  $b[j] \geq 0$  then
     $j := j + 1$ 
  else begin {  $b[i] > 0 > b[j]$  }
    if  $b[i] < -b[j]$  then  $m := b[i]$  else  $m := -b[j]$ 
    {  $m = b[i] \downarrow -b[j]$  }
    ; writeln ( $i, ' ', j, ' ', m$ )
    ;  $b[i] := b[i] - m$  { ;  $c[i] := c[i] + m$  }
    ;  $b[j] := b[j] + m$  { ;  $c[j] := c[j] - m$  }
  end { else }
end { while }
{  $\forall_k b[k] = 0, \text{ hence } c = B$  }
end; { WriteOutput }

```

The while-loop terminates in at most  $3N$  steps, because in each step either  $i$  increases, or  $j$  increases, or the number of non-zero balances decreases. As was already pointed out, this algorithm produces an acyclic graph, which involves no more than  $N-1$  transfers.

### 8.3. Minimum Number of Transfers

Since minimizing the number of transfers turned out to be a hard problem in Section 6.2, one cannot do much better than try all possibilities. In this case, that means inspecting all partitions whose parts have zero balance sums, and finding the one with maximum number of parts. In view of the lower bounds (28) and (30), there is an opportunity for **branch-and-bound**. There are clear relationships to multiple knapsacks and bin packing.

One could imagine that for small values of the balances, an efficient **dynamic programming** solution exists, as with the knapsack problem. However, in general, even that is not possible, since the well-known problem **3-Partition** (Garey and Johnson, 1979) is **NP-complete in the strong sense**, and it can be transformed to minimizing the number of transfers as follows. First, the definition of the 3-Partition problem:

Given are a positive integer  $K$  and a set  $S$  of  $3m$  items  $x$  with positive integer size  $s(x)$  such that  $K/4 < s(x) < K/2$  and  $\sum_{x \in S} s(x) = mK$ . The question is whether  $S$  can be partitioned into  $m$  disjoint sets  $S_i$  for  $1 \leq i \leq m$  with  $\sum_{x \in S_i} s(x) = K$ . Note that on account of the restrictions on  $s(x)$ , each  $S_i$  contains exactly three items.

This situation can be translated into the following transfer minimization problem:

There are  $m$  negative balances of value  $-K$  each, and  $3m$  positive balances of value  $s(x)$ . These balances take at least  $m$  transfers to settle. If it can be done in exactly  $m$  transfers, then that solution corresponds to a partition of  $S$  into  $m$  sets  $S_i$

with  $\sum_{x \in S_i} s(x) = K$ , which is also a solution to the instance of 3-Partition. If it cannot be done in exactly  $m$  transfers (but requires more transfers), then there is no such partition, and the instance of 3-Partition cannot be solved.

Hence, minimizing the number of transfers is at least as difficult as 3-Partition.

#### 8.4. *Minimum Total Amount in Minimum Number of Transfers*

The combined minimization problem (minimizing the number of transfers and the total amount transferred) can be solved by first finding an appropriate partition (hard), then, for each component, minimizing the amount transferred (easy).

## 9. Conclusion

When I wrote the initial version of this article in 1998, I had not seen the problem of efficiently settling multiple debts in the literature. I must admit that I did not do much research to locate it. Nor have I discussed it with a lot of people, because I wanted to keep it secret in order to use it in a competition. The problem seemed directly related to the **input-output economic model** of Leontief (Leontief, 1951; HET, 2003).

An anonymous reviewer referred me to a series of articles about clearing debts in the Russian economy (Kalitkin, 1995; Anikonov and Tsitsiashvili, 1997; Kalitkin *et al.*, 2001). Here the grand total balance is not zero (open economy; cf. (11), (18)). The objective is to clear as much of the debts as possible, given certain legal restrictions. The number of transactions and total amount transferred are not so important. The algorithmic complexity is not investigated.

Problem 96 in (Steinhaus(1964)) concerns minimization of the number of transfers. A further literature search reveals relationships with the **minimum-cost network flow problem** and special cases like the **transportation problem** and **transshipment problem** (Cook *et al.*, 1998). Our problem of minimizing the number of transfers appears to be known as a special case of the **fixed-charge transportation problem** (Spielberg, 1964) and the **Uncapacitated Fixed-charge Network Flow (UFNF)** problem (Duhamel, 2001), with per-unit costs  $c_{i,j} = 0$  and fixed costs  $f_{i,j} = 1$ .

I have mostly presented the material in the order that I discovered it. Of course, this is not the only approach. Some readers may have wondered at places why I did not give the key argument immediately. Well, I simply did not see it immediately. Therefore, this story reveals more how my problem-solving brain works, than that it teaches how best to solve problems. With hindsight, some of my excursions seem unnecessarily long-winded. Possibly, I could have avoided these by concentrating more on the abstract problem, instead of working with concrete – and sometimes misleading – examples. It is, however, important to realize that most high-school students need details and they should be encouraged to follow their own path of investigation. In secondary education it is good to include many concrete examples, as I have done.

The ancient Greeks, such as Pythagoras and Euclid, could, no doubt, have understood this problem. One can only wonder whether they could have solved it. More likely they

would not have appreciated the problem. Their cultural framework prevented them from asking such questions. Graph theory is a convenient ingredient for fully understanding and tackling the problem. Euler is generally recognized as the father of graph theory with his famous publication in 1736 (Euler, 1736). That may seem long ago but one must realize that Pythagoras lived 500 B.C. and Euclid 300 B.C. Clay tablets with mathematical inscriptions date back to well before 10,000 B.C. Mathematics has grown considerably, especially since Euler, and is still under turbulent development. Only in the second half of this century has graph theory become a part of ‘respectable’ mathematics (instead of ‘just’ dealing with recreational problems and games). There is ample literature on graph theory. The terminology, however, is not uniform and may differ from what I have used.

I hope that this problem can serve as an example of how a simple question leads to many important concepts in mathematics and computing science. Teachers need not use this material directly. I would rather encourage them to find their own questions that they can develop in detail and teach with enthusiasm.

### Answers to the Exercises of Section 6.2

Consider the six distinct balances  $+5, +3, +2, -1, -4, -5$ . Then there are two solutions with a minimum number of transfers. One of them does not match the opposites  $+5$  and  $-5$ .

The twelve distinct balances

$$+900, +203, +100, +99, +98, +1, -2, -202, -297, -299, -300, -301$$

contain no opposites (zero-sum pairs). There is one zero-sum triple:  $+203, +98, -301$ . However, the remainder cannot be split into two or more zero-sum parts, thus yielding a 10-transfer solution. The unique maximal partition into zero-sum parts is

$$\begin{aligned} &+900, -299, -300, -301; \\ &+203, +1, -202, -2; \\ &+100, +99, +98, -297 \end{aligned}$$

giving a 9-transfer solution. Thus, combining zero-sum triples is not always a good idea.

### References

- Anikonov, D.S., and G.Sh. Tsitsiashvili (1997). Reduction of clearing of debts to the transportation problem. *Doklady Mathematics*, **55** (1), 120. Translated from *Doklady Akademii Nauk*, **352** (6), 730.
- Cook, W.J., W.H. Cunningham and W.R. Pulleyblank (1998). *Combinatorial Optimization*. Wiley.
- Duhamel, C. (2001). Solving the uncapacitated fixed-charge network flow with metaheuristics. In *Proceedings of MIC 2001: 4th Metaheuristics International Conference*. Portugal, July 2001, pp. 685–690.
- Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. U. Petrop.*, **8**, 128–140. Reprinted in *Opera Omnia*, Ser. I-7, pp. 1–10, 1766.
- Also see <http://mathworld.wolfram.com/KoenigsbergBridgeProblem.html> (accessed July 2003).

- Garey, M., and D. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman.
- HET (2003). Wassily Leontief, 1906–1999. In G.L. Fonseca and L.J. Ussher (Eds.), *History of Economic Thought Website*. Department of Economics of the New School for Social Research.  
<http://cepa.newschool.edu/het/profiles/leontief.htm> (accessed July 2003).
- Kalitkin, N.N. (1995). Optimal clearing of mutual debts of enterprises. *Mathematical Modelling*, **7** (1), 11–21 (in Russian).
- Kalitkin, N.N., L.V. Kuz'mina and M.V. Chernenkov (2001). Clearing debts by the fractional method. *Doklady Mathematics*, **63** (3), 437–440. Translated from *Doklady Akademii Nauk*, **378** (1), 29–32.
- Leontief, W. (1951). Input-output economics. *Scientific American*, Oct.
- Spielberg, K. (1964). On the fixed charge transportation problem. In *Proceedings of the 1964 19th ACM national conference*, pp. 11.10.1–11.10.13.
- Steinhaus, H. (1964). *One Hundred Problems in Elementary Mathematics*. Dover Publications.

**T. Verhoeff** is an assistant professor in computing science at Eindhoven University of Technology (TUE), the Netherlands. He obtained his PhD (A Theory of Delay-Insensitive Systems, 1994) from TUE. His current research area is software construction. He chaired the Scientific Committee of IOI 1995 in The Netherlands and contributed tasks to various competitions. In 1999, he was Finals Directors of the ACM ICPC World Finals held in Eindhoven, The Netherlands. He chairs the IOI Scientific Committee since 1999.

## Daugybinių skolų sureguliuojimas: kvietimas į kompiuterių mokslą

Tom VERHOEFF

Ne viena mokyklinė informatikos programa kenčia nuo “enciklopedijos sindromo”. Dažnai bandoma apžvelgti kuo daugiau temų, tačiau visos jos išdėstomos gan paviršutiniškai. Straipsnio autorius mano, kad geriau pasirinkti nedaug temų, tačiau jas giliai išanalizuoti.

Straipsnyje išsamiai nagrinėjamas gana paprastas uždavinys: yra grupė draugų, vienas kitam skolinančių pinigų ištisius metus. Jie rūpestingai užrašinėja visus skolinimus ir metų gale nori sureguliuoti tarpusavio skolas.

Kaip jie turėtų pervesti (perduoti) pinigų, kad atsiskaitytų tarpusavyje? Ar sunku surasti reikiamą tarpusavio atsiskaitymo schemą? Ar ta schema efektyvi? Siūloma minimizuoti pinigų pervedimą, t.y., pervedimų ir pinigų skaičių.

Autorius išsamiai išanalizuoja uždavinį ir pateikia įvairius sprendimus. Remiamasi transformacijomis, grafų teorija, kombinatorika. Straipsnyje medžiaga pateikiama daugiausia tokiu būdu, kaip ją tyrinėjo autorius.

Tikimasi, kad straipsnis bus naudingas ne tik mokslininkams, bet ir gabiems studentams. Svarbu suprasti, kad daugumai pradedančiųjų tyrėjų reikia smulkiai išaiškinti ir skatinti juos pačius tyrinėti problemą. Jiems naudingi ir konkretūs pavyzdžiai, kurių nemažai autorius ir įtraukė į straipsnį.

Straipsnyje pagrindžiama, kaip nesunku minimizuoti pervedamų pinigų kiekį. Pateikta nesudėtinga Paskalio programa, sprendžianti šį uždavinį. Kita vertus, pervedimų skaičiaus minimizavimas yra NP sudėtingumo uždavinys.

Straipsnyje aptartas pavyzdys galės būti pavyzdžiu, kaip paprasto uždavinio analizė supažindina su daugeliu svarbių matematikos ir kompiuterių mokslo sąvokomis ir šitaip skatina giliau domėtis šiomis sritimis.