A mechanical model for the Hitchcock problem

M. Hénon
C.N.R.S., Observatoire de Nice,
BP 4229, 06304 Nice Cedex 4, France

January 23, 1992

Abstract

We describe a mechanical device which can be used as an analog computer to solve the Hitchcock optimization problem. In practice this device is simulated by a numerical algorithm. Tests show that this algorithm is 60 times faster than a current subroutine (NAG library) for an average 1000×1000 problem. Its performance is even better for degenerate problems in which the weights take only a small number of integer values.

Key words: Hitchcock problem, transportation problem, analog computer, mechanical model.

Contents

1 Introduction

We describe here an algorithm for the solution of the Hitchcock optimization problem [7] (also known as the transportation problem).

The development of this algorithm had its origin in studies of the lattice gas method for three-dimensional fluid simulations [8]. The optimization of the collision table has generally the form of a Hitchcock problem [5, 12], with large cost matrices. Classical algorithms were found to require prohibitively long computing times. Therefore an attempt was made to devise a method which would take advantage of the peculiarities of the lattice gas problem. This method then turned out to be of general applicability.

The present algorithm was developed independently of the already published studies of the Hitchcock problem and related optimization problems. This was not planned; it only reflects the way things happened, and the ignorance of this author who comes from a rather different field. More will be said about this in Section 9.

The paper is organized as follows. Section 2 defines the problem. In Section 3, we describe a mechanical device which can be used as an analog computer to solve the Hitchcock problem. In Section 4 we develop an appropriate graph representation and a numerical scheme which simulates the mechanical model. This is illustrated by a detailed example in Section 5. In Section 6, we give a rigorous definition and justification of the algorithm. Section 7 describes some aspects of the computer implementation. In Section 8, the algorithm is compared with the NAG library subroutine for the solution of the Hitchcock problem. In the particular case of the assignment problem, comparisons are also made with the algorithm of Burkard and Derigs [2]. A few comments are made in Section 9. Finally, an Appendix derives some bounds on the number of operations.

2 The problem

We are given a matrix c_{ij} and two vectors $a_i \geq 0$, $b_j \geq 0$. The index i runs from 1 to m and the index j runs from 1 to n. There is

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j. \tag{1}$$

The problem is to find coefficients f_{ij} which *maximize* the sum

$$\sum_i \sum_j c_{ij} f_{ij} \tag{2}$$

subject to the constraints

$$\sum_j f_{ij} \quad = \quad a_i \qquad (i = 1, \ldots, m), \tag{3}$$

$$\sum_i f_{ij} = b_j \qquad (j = 1, \ldots, n), \tag{4}$$

$$f_{ij} \geq 0. \tag{5}$$

A set of f_{ij} which satisfies the constraints (3) to (5) will be called a *feasible solution*. A feasible solution which maximizes (2) will be called an *optimal solution*.

Notes: (i) This is essentially the *Hitchcock problem*. It can be reduced to the standard form given for instance in [11, Section 7.4], by defining

$$c^*_{ij} = c_{\text{sup}} - c_{ij}, \tag{6}$$

where c_{sup} is a constant satisfying

$$c_{\text{sup}} > \max_{i,j} c_{ij}. \tag{7}$$

(ii) There is no sign condition on the c_{ij}, which can be positive or negative.

(iii) If $a_i = 0$ for one row i, then from (3) and (5) we have $f_{ij} = 0$ for all j. This row does not contribute to the sum (2); the values of the c_{ij} on that row are irrelevant. Thus this row could be eliminated without changing the problem. The same holds if $b_j = 0$ for some j. We could therefore in principle restrict our attention to the case where the a_i and b_j are strictly positive, as is usually done [4, 11]. In practice, however, it is convenient to be able to include the cases with some $a_i = 0$ and/or $b_j = 0$ into the general treatment. The algorithm to be described works just as well in such cases.

(iv) The a_i, b_j, c_{ij} can be integer or real numbers.

(v) A special case of the Hitchcock problem is

$$a_i = 1 \qquad (i = 1, \ldots, m), \qquad b_j = 1 \qquad (j = 1, \ldots, n). \tag{8}$$

From (1) it follows then that

$$m = n. \tag{9}$$

If we prescribe the additional constraint

$$f_{ij} = 0 \text{ or } 1, \tag{10}$$

we obtain the classical *assignment problem* ([11, chap. 11]).

3 A mechanical model

We show now that it is possible to build a simple mechanical device which solves the Hitchcock problem. This device acts as a *analog computer*: the numbers entering the problem are represented by physical quantities, and the equations are replaced by physical laws.

We define a system of axes x, y, z in physical space (Fig. 1). To every value of

Figure 1: An analog computer for the solution of the Hitchcock problem.

i is associated a rod A_i parallel to the y axis, which we will call a *row* by reference to the c_{ij} matrix. Mechanical constraints (not shown on the figure) ensure that each row can only move in the vertical direction. More precisely, each row remains parallel to the y axis and moves in a fixed vertical plane $x = \text{Const}$. The variable height z of the lower face of row A_i will be designated by α_i. Row A_i has a weight a_i and thus is subjected to a force a_i towards the negative z axis.

Similarly, to every value of j is associated a rod B_j parallel to the x axis, which we will call a *column*. (Notice that these "columns" are horizontal in the three-dimensional physical space !). Each column is constrained to move only vertically: it remains parallel to the x axis and moves in a fixed vertical plane $y = \text{Const}$. It has a *negative* weight $-b_j$ (or, if one prefers, a buoyancy b_j) and thus is subjected to a force b_j towards the positive z axis. (In an actual model, this might be realized with cables and counterweights). We call $z = \beta_j$ the height of its upper face.

Finally, small vertical cylinders or *studs* of height c_{ij} and of negligible weight are placed on the columns, in such a way that each stud enforces a minimal distance between row A_i and column B_j:

$$\alpha_i - \beta_j \geq c_{ij}. \tag{11}$$

(Note: this description seems to imply that $c_{ij} \geq 0$. Actually it is possible, although mechanically more awkward, to have negative values of the c_{ij} by bending the rods. One can also make all c_{ij} positive by adding a sufficiently large constant to all of them. Therefore we continue to consider that the c_{ij} can be arbitrary.)

The potential energy of the system is, within an additive constant:

$$U = \sum_i a_i \alpha_i - \sum_j b_j \beta_j. \tag{12}$$

Initially, all rods are maintained at a fixed position by two additional fixed rods P and Q acting as *stops* (Fig. 1), with the rows A_i well above the columns B_j, so that there is no contact between the rows and the studs. For instance we take $\beta_j = 0$ ($j = 1, \dots, n$) and $\alpha_i = c_{\text{sup}}$ ($i = 1, \dots, m$). Then the rods are released by removing the stops P and Q, and the system starts evolving. Rows go down, columns go up, and contacts are made with the studs. Aggregates of rows and columns are progressively formed. As new contacts are made, these aggregates are modified. Thus a complex evolution may take place.

It will be convenient to imagine that the system is immersed in a viscous fluid, so that the velocity of an object, rather than its acceleration, is proportional to the force to which it is subjected. More specifically, let us define formally an aggregate as a subset S of rows and columns which are in contact (they form a connected set in space) and move with the same velocity. We also assume that it is a maximal connected set: no other row or column touches it. The downward force acting on this aggregate is its total weight, which we call $M(S)$:

$$M(S) = \sum_{A_i \in S} a_i - \sum_{B_j \in S} b_j. \tag{13}$$

We will simply assume that the aggregate moves with a velocity

$$\frac{dz}{dt} = -M(S). \tag{14}$$

In particular an aggregate remains motionless if the force applied to it vanishes.

It is intuitively clear that the system will reach an equilibrium, an in fact we have

Theorem 1 *An equilibrium is reached after a finite time.*

Proof: an aggregate S has a potential energy

$$U(S) = \sum_{A_i \in S} a_i \alpha_i - \sum_{B_j \in S} b_j \beta_j. \tag{15}$$

From (13) and (14) we find that this potential energy decreases with time according to

$$\frac{dU(S)}{dt} = -M^2(S). \tag{16}$$

We call $|M|_{\min}$ the minimum of all non-zero values of $|M(S)|$, over all subsets S of the full set of rods. Since there is only a finite number of subsets, we have $|M|_{\min} > 0$, and:

$$\begin{cases} \text{either } dU(S)/dt = 0, \\ \text{or } dU(S)/dt \leq -|M|^2_{\min}. \end{cases} \tag{17}$$

The total potential energy U is the sum of the potential energies of the aggregates. Therefore we also have

$$\begin{cases} \text{either } dU/dt = 0, \\ \text{or } dU/dt \leq -|M|^2_{\min}. \end{cases} \tag{18}$$

The first case is realized only if $dU(S)/dt = 0$ for every aggregate, i.e. if all aggregates are motionless. Thus: either the system is in equilibrium, or its potential energy decreases at a rate at least equal to $|M|^2_{\min}$.

On the other hand we have

$$\alpha_i - \beta_j \geq \min_{i,j} c_{ij}. \tag{19}$$

Multiplying by $a_i b_j$, summing on i and j, and using (1), we obtain

$$U \sum a_i \geq \min_{i,j} c_{ij} \left(\sum a_i \right)^2. \tag{20}$$

This gives a lower bound for U.

Combining these results, we find that the system reaches an equilibrium after a finite time (for which an upper bound is easily derived). ■

We consider now such an equilibrium state. We will show that

Theorem 2 *If the system is in equilibrium, and if F_{ij} is the force transmitted through stud c_{ij} from row A_i to column B_j, then $f_{ij} = F_{ij}$ is an optimal solution of the Hitchcock problem.*

Proof: (i) Each row is in equilibrium, therefore

$$\sum_j F_{ij} = a_i. \tag{21}$$

(ii) Each column is in equilibrium, therefore

$$\sum_i F_{ij} = b_j. \tag{22}$$

(iii) F_{ij} cannot be negative since it is transmitted by contact:

$$F_{ij} \geq 0. \tag{23}$$

Therefore F_{ij} is a feasible solution.

(iv) If $F_{ij} > 0$, row A_i is in contact with column B_j, and therefore

$$\alpha_i - \beta_j = c_{ij}. \tag{24}$$

It follows that

$$F_{ij}(\alpha_i - \beta_j - c_{ij}) = 0 \qquad \forall i, j. \tag{25}$$

Summing (25) over i and j and using (21) and (22), we obtain

$$\sum_i a_i\alpha_i - \sum_j b_j\beta_j - \sum_i \sum_j F_{ij}c_{ij} = 0. \tag{26}$$

Consider another feasible solution f'_{ij}. From (5) and (11) we have

$$f'_{ij}(\alpha_i - \beta_j - c_{ij}) \geq 0 \qquad \forall i, j \tag{27}$$

and therefore, summing over i and j and using (3) and (4):

$$\sum_i a_i\alpha_i - \sum_j b_j\beta_j - \sum_i \sum_j f'_{ij}c_{ij} \geq 0. \tag{28}$$

Comparing with (26), we have

$$\sum_i \sum_j f'_{ij}c_{ij} \leq \sum_i \sum_j F_{ij}c_{ij} \tag{29}$$

which shows that F_{ij} is optimal. ■

Incidentally, (26) shows that the "cost" $\sum_i \sum_j F_{ij}c_{ij}$ of the optimal solution is equal to the potential energy U of the corresponding equilibrium.

4 Numerical simulation

4.1 Method

The analog computer described in the previous Section could be built in principle, but for large values of m and n this would be impractical. Instead, we will *simulate* on a digital computer the behaviour of the analog computer, as it progressively settles into an equilibrium.

It would be possible to simulate the evolution of the mechanical system as described in the previous Section, i.e. to remove suddenly the two stops P and Q and let the system evolve freely until it has found an equilibrium. In that case, however, all rods would interact more or less simultaneously, and the simulation would be somewhat complex; essentially we would have to solve an N-body problem. It turns out to be simpler and also more efficient to guide the system through a more controlled and orderly evolution. This is permitted because, as shown by Theorem 2, all we need in order to solve the Hitchcock problem is to find an equilibrium; how we arrive at it is irrelevant.

Many algorithms can be imagined. Here we will only describe one of the simplest methods, which was found to work well, although it is not always the most efficient in terms of computing time (see below Section 7). We give here an informal description of the algorithm, based on physical intuition; a rigorous derivation will be presented in Section 6.

The stops P and Q are not removed. Instead, the stop P is held fixed during the whole process, and the stop Q is slowly lowered from its initial position. The velocity of descent is smaller than the minimal non-zero velocity of any aggregate, $|M|_{\min}$, so that the evolution is fully controlled by the motion of the stop Q. At any given time the system is in quasi-equilibrium: if Q stops then nothing moves anymore.

The evolution of the system will be studied in detail below. It ends when the whole system of rows and columns comes to rest. The stop Q, continuing its descent, ceases then to be in contact with any row and can be removed. The force exerted by the stop P on any column still in contact with it is then zero, and that stop can also be removed. Thus an equibrilium has been reached, from which the optimal solution can be read.

4.2 Graph representation

The state of the system at any given time can be conveniently represented by a graph, as follows. Each rod is a node of the graph; we represent rows by squares and columns by circles. An edge always joins a square to a circle: the graph is *bipartite*. An edge is present between row i and column j when the row and the column are in contact through the stud c_{ij}, i.e. when

$$\alpha_i - \beta_j = c_{ij}. \tag{30}$$

The stops are also represented by nodes, and their contacts with rods are similarly represented by edges. In order to preserve the bipartite property, the stop P must then be a square while the stop Q is a circle. As an illustration, Fig. 2 represents the initial state of the system, before any row has been lowered. Note that the graph is a purely topological representation: the position of a symbol in the graph has nothing to do with the position of the corresponding rod in physical space.

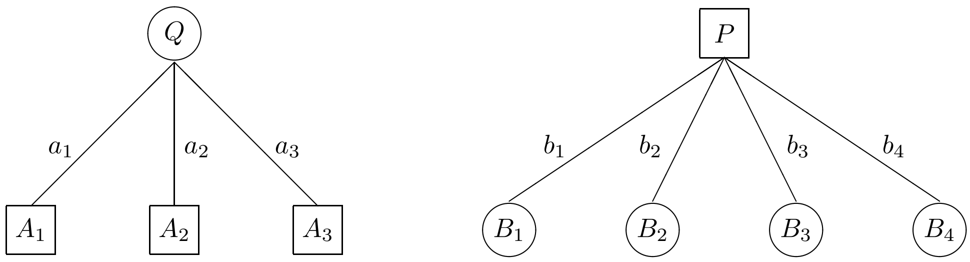


Figure 2: Graph of the initial state.

The force transmitted downwards from a row to a column, or from a row to the Q stop, or from the P stop to a column, can be written beside the corresponding edge. In the initial state, only the last two kinds of forces are present; they have the values indicated on Fig. 2. Note that this force is always positive or zero. At any given time in the procedure, each row is in equilibrium; therefore the sum of the forces emanating from it must equal the absolute value of its weight a_i. Similarly, each column is in equilibrium, and the sum of the forces received by it must equal its buoyancy b_j.

Can this graph have cycles ? A cycle will be an even sequence of alternating rows and columns:

$$i_1, j_1, i_2, j_2, \ldots, i_p, j_p. \tag{31}$$

From (30) we have then

$$c_{i_1 j_1} - c_{i_2 j_1} + c_{i_2 j_2} - c_{i_3 j_2} + \ldots + c_{i_p j_p} - c_{i_1 j_p} = 0. \tag{32}$$

In order to simplify the exposition, we make the following assumption (which will be removed in Section 6):

Assumption 1 *The c_{ij} are such that there are no linear relations of the form (32) between them.*

Then the graph has no cycles and is a *forest*, or a collection of trees. Each tree corresponds to an aggregate as defined in Section 3.

It will be convenient to assume also, for the time being, that no subset of rows and columns can be in equilibrium. This can be expressed as:

Assumption 2 *Let I be a subset of $\{1, \ldots, m\}$ and J a subset of $\{1, \ldots, n\}$. The relation*

$$\sum_{i \in I} a_i = \sum_{j \in J} b_j \tag{33}$$

is true only in two cases: (i) $I = J = \emptyset$*; (ii)* $I = \{1, \ldots, m\}$ *and* $J = \{1, \ldots, n\}$.

Note that this excludes in particular $a_i = 0$ or $b_j = 0$: rows and columns must have positive weights and buoyancies.

We remark that these two assumptions are satisfied in principle in the generic case, when the a_i, b_j, c_{ij} are real numbers with arbitrary values. In practice, however, these numbers are often integers with a restricted range and the assumptions are frequently violated.

It will be proved in Section 6 that the graph always consists of two trees, except at particular instants of time where they fuse into a single tree (see below Section 4.3). One of them contains Q and will be called *moving tree*. The other contains P and will be called *fixed tree*. We will represent the two trees with the usual hierarchical representation of trees ([9], Section 2.3), taking the stop as root for each tree. Fig. 3 shows an example.

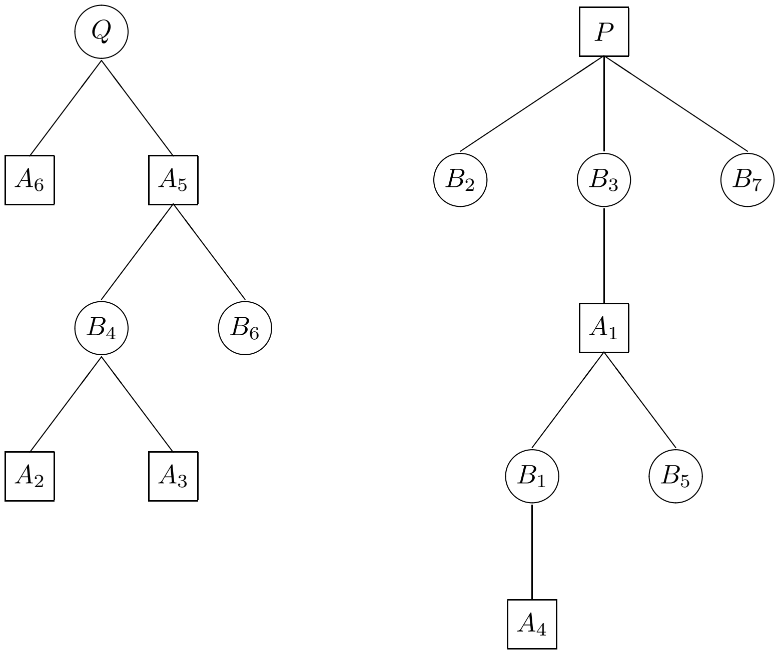


Figure 3: Example of a graph. Left: moving tree. Right: fixed tree.

It will be convenient to extend the usual terminology of parents and children by specifying that *rows are male* and *columns are female*. The stop Q is female and the stop P is male. To recapitulate:

$$\left.\begin{array}{r}\text{row}\\ \text{stop } P\end{array}\right\} = \square = \text{male},$$

$$\left.\begin{array}{r}\text{column}\\ \text{stop } Q\end{array}\right\} = \bigcirc = \text{female}. \qquad (34)$$

In each tree, sex is thus alternating from one generation to the next. A row has one mother and any number of daughters, while a column has one father and any number of sons. The stops themselves have no parents.

4.3 Contact and rearrangement

As the stop Q goes down, the rows and columns which belong to the moving tree move with it. This continues until a contact is made between the two trees. Since rows always remain above columns in physical space, this contact happens necessarily between a moving row A_{i_c} and a fixed column B_{j_c}. A new edge is created and the two trees are temporarily fused into a single tree. A single contact is made, because two simultaneous contacts would imply a cycle, in contradiction to Assumption 1. Fig. 4 shows an example, posterior in time to Fig. 3, where contact is made between the row A_3 and the column B_1. (Note: in the example of Fig. 4, the row and the column which make contact happen to be at the same level in their respective trees; this need not be so in general.)

As a result of the contact, the forces change. To understand what happens, it is convenient to imagine that the studs are slightly elastic, so that the change does not happen all at once, but progressively over a small interval of time. The force λ along the newly created edge, which is initially zero, increases as the moving row moves down. This induces other changes in neighbouring edges. We consider the path from Q to P made by (i) the path from Q to A_{i_c} in the moving tree; (ii) the newly created edge from A_{i_c} to B_{j_c}; (iii) the path from B_{j_c} to P in the fixed tree. We call this the *main path*. In Fig. 4, for instance, the main path is $QA_5B_4A_3B_1A_1B_3P$. The graph can then be viewed as made up of the main path, plus a number of lateral branches. The lateral branches are not involved in the readjustment of the forces; each of them is attached to the main path by a single edge and the force along that edge equals the weight or the buoyancy of the branch, which does not change. Therefore, only forces along the main path can change. Each node must remain in equilibrium; therefore all changes have the same modulus λ and alternate in sign along the main path, as shown by Fig. 4. (Note in particular that the forces of contact with the two stops decrease).

This continues until one of the decreasing forces becomes zero. (Two forces cannot vanish simultaneoulsy, because the intermediate tree would have zero weight, in contradiction to Assumption 2). The chain breaks then at the corresponding edge,

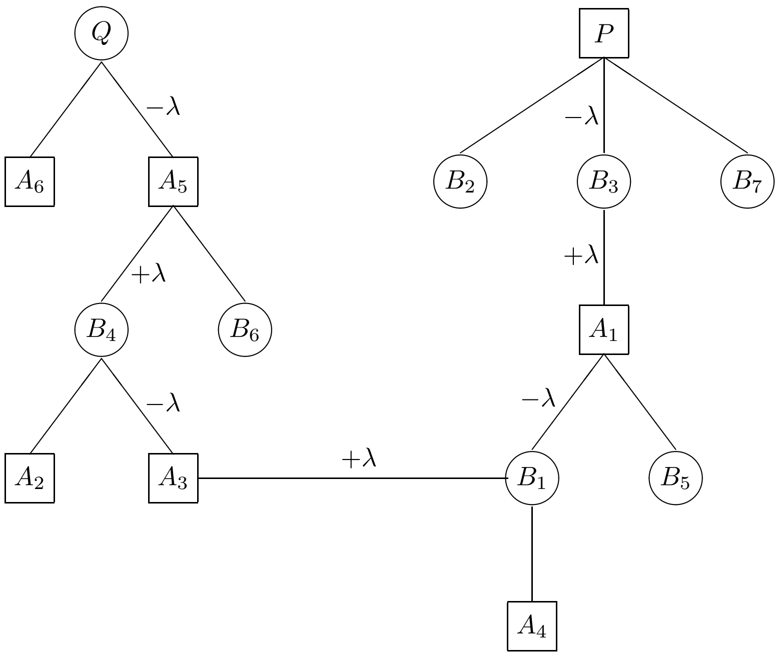


Figure 4: Contact between the moving and the fixed tree, and readjustment of the forces. Here the numbers $+\lambda$ and $-\lambda$ represent the changes in the forces, rather than the forces themselves.

which disappears, and we have again two separate trees. Thus, the whole episode ends in a *capture* of a part of one tree by the other. The capture can occur in either direction, depending on where is the weakest link of the main path. For instance if the weakest link in Fig. 4 is between B_1 and A_1, the branch of the fixed tree with head B_1 is captured by the moving tree and we obtain Fig. 5. B_1 ceases to be in contact with A_1 as the moving tree continues its descent.

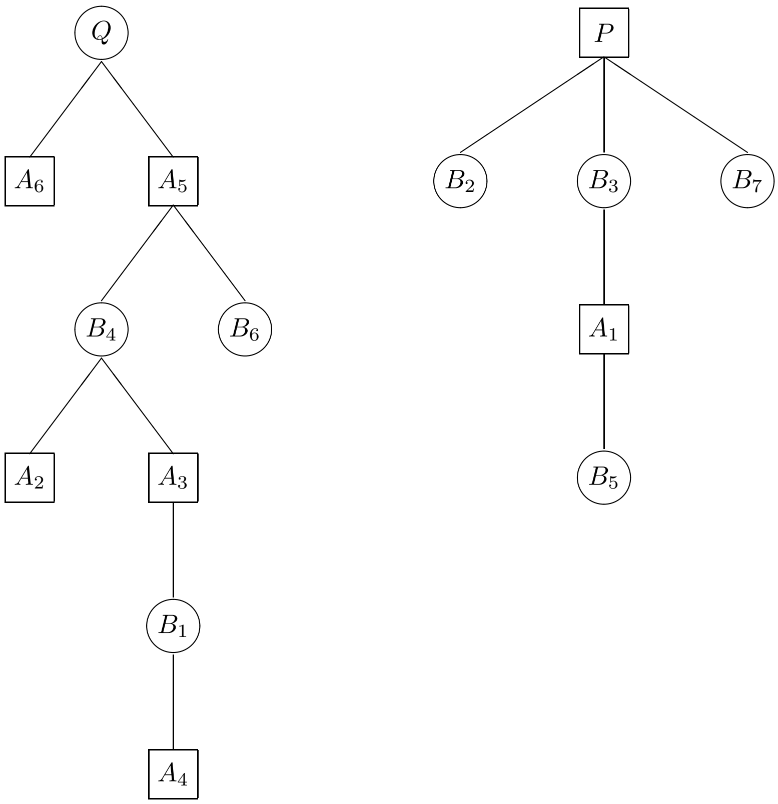


Figure 5: Example of a capture of a part of the fixed tree by the moving tree.

The new moving tree continues to go down with the stop Q. Eventually a new contact is made, and one of the trees captures a part of the other. This goes on until the moving tree is reduced to the stop Q alone. It can be shown that this always happens after a finite number of captures (see Appendix A). Only the fixed tree remains, now containing all rows and all columns, and we have the sought equilibrium.

5 Example

We exhibit here the step-by-step progress of the algorithm on a simple example with $m = 3$, $n = 4$. Table 1 shows the values of the given coefficients a_i, b_j, and c_{ij}. Note that the condition (1) is verified.

Table 1: Values of the parameters a_i (left column), b_j (top row), and c_{ij} for the example problem.

	44	52	13	37
86	26	64	33	62
4	63	27	13	14
56	94	4	4	52

When executing the algorithm by hand, it is convenient to keep track of the distances between the rows and the studs, i.e. the quantities

$$\gamma_{ij} = \alpha_i - \beta_j - c_{ij}. \tag{35}$$

One can then easily determine where the next contact will take place. The distances γ_{ij} are shown on the left in Fig. 6, while the graph (moving tree and fixed tree) is shown on the right. Only the indices i or j of the rows and columns are indicated; the type is indicated by the symbol (square for a row, circle for a column). Evolution proceeds from top to bottom; successive steps are represented in lines labelled a, b, c, ... Lines b, d, f, ..., correspond to a descent of the moving tree; the distances change, while the forces and the trees remain fixed. Conversely, lines c, e, g, ..., correspond to a readjustment of the forces and of the trees, during which the distances do not change.

Initially we set the height of the rows and columns at $\alpha_i = 100$ and $\beta_j = 0$. The corresponding distances γ_{ij} are then obtained by complementing to 100 the values of table 1 and are shown in Fig. 6, line a. The initial moving and fixed trees are set up as indicated in Section 4.2, Fig. 2, and are shown in line b.

All rows are moving and all columns are fixed, therefore all distances γ_{ij} decrease. From line a, we immediately find that the moving tree can descend a distance $d = 6$; a contact is then made between row 3 and column 1. The new distances are shown in line c.

We now readjust the forces and the trees. The main path is: stop Q − row 3 − column 1 − stop P. The weakest link is between the column 1 and the stop P, with a force 44. Therefore the column 1 is captured by the moving tree. The forces along the main path change by $\lambda = \pm 44$. The new trees and the new forces are represented in line d.

All rows are still moving; in addition, column 1 is also moving. Therefore the distances γ_{ij} remain fixed for $j = 1$ (first column of matrix) and decrease for $j \in$

$\{2,3,4\}$. From line c we find then that the distance of descent is $d = 30$. Contact is made between row 1 and column 2. The new distances are shown in line e.

Column 2 is captured by the moving tree and we obtain line f. Now the distances decrease for $j \in \{3,4\}$. After a descent of $d = 2$, contact is made between row 1 and column 4. The new distances are shown in line g.

This time the weakest link is between Q and row 1. Therefore row 1, and its daughter the column 2, are captured by the fixed tree. The new trees are represented in line h.

Now only the distances γ_{ij} with $i \in \{2,3\}$ and $j \in \{2,3,4\}$ are decreasing. Therefore we have a descent of $d = 10$. Note that γ_{11} is *increasing*, since row 1 is fixed and column 1 is moving. The other distances remain fixed. The new distances are shown in line i. Contact is made between row 3 and column 4.

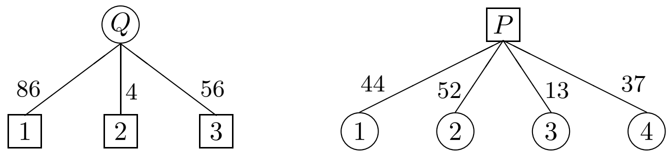
The evolution continues. In line i, the column 4 and its two descendants are captured by the moving tree. In line k, we observe a more complex event, involving the capture of a large piece of the main path and a drastic reorganization of the trees. Finally, in line m the last remnant of the moving tree is captured.

We reach line n, where only the fixed tree remains. The forces f_{ij} along the edges of that tree give the solution of the Hitchcock problem. They can be rewritten in matrix form (Table 2).

a

74	36	67	38
37	73	87	86
6	96	96	48

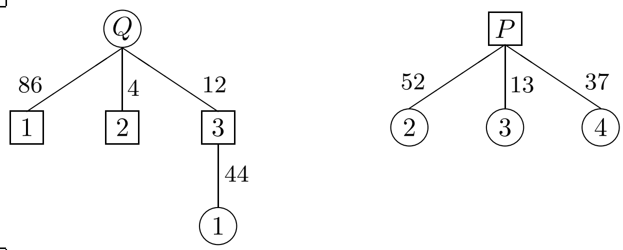
b $d = 6$



c

68	30	61	32
31	67	81	80
0	90	90	42

d $d = 30$



e

68	0	31	2
31	37	51	50
0	60	60	12

f $d = 2$

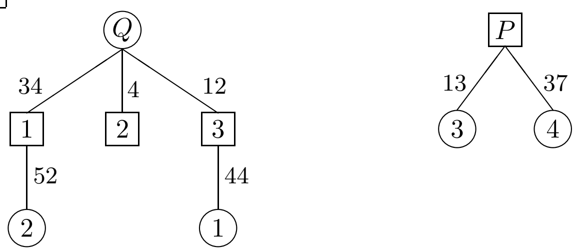
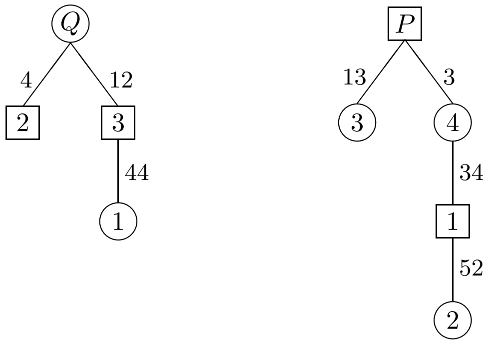


Figure 6: Exemple. Left: the distances $\gamma_{ij} = \alpha_i - \beta_j - c_{ij}$ between the lines and the studs. Right: moving tree and fixed tree.

g

68	0	29	0
31	37	49	48
0	60	58	10

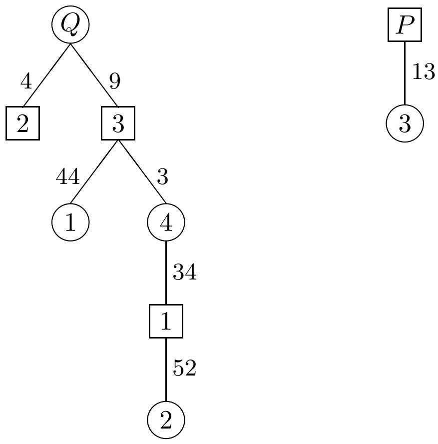
h $d = 10$



i

78	0	29	0
31	27	39	38
0	50	48	0

j $d = 29$

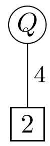


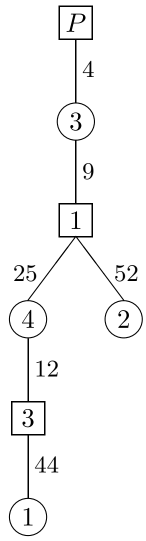
k

78	0	0	0
31	27	10	38
0	50	19	0

Figure 6 (continued).

l $d = 10$





m

78	0	0	0
21	17	0	28
0	50	19	0

n equilibrium

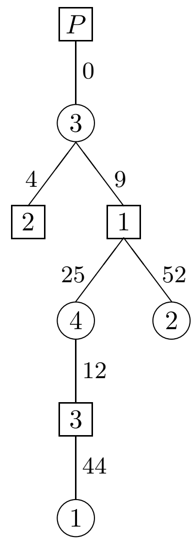


Figure 6 (continued).

Table 2: Solution of the Hitchcock problem defined by Table 1.

0	52	9	25
0	0	4	0
44	0	0	12

6 Formal definition and justification of the algorithm

6.1 Definitions

In the present Section, we give a more rigorous definition of the numerical algorithm, and we prove that it solves the Hitchcock problem. We also drop the restrictions introduced in Section 4: assumptions 1 and 2 do not have to be satisfied any more, i.e. relations of the form (32) and (33) are allowed.

This Section is independent of the description of the mechanical model in Section 3, and also of the informal description of the numerical algorithm in Section 4. It will be sometimes convenient to use names which are reminiscent of the origin of the algorithm, such as "descent" or "force"; the reasoning, however, will be purely mathematical.

In what follows, unless otherwise specified, i is always understood to take all values from 1 to m and j to take all values from 1 to n.

We want to solve a Hitchcock problem defined by given m, n, a_i, b_j, c_{ij} (Section 2). The algorithm operates on the following collection of objects:

- A graph with $m + n + 2$ nodes labelled $A_1, \ldots, A_m, B_1, \ldots, B_n, P, Q$.

  This set of nodes remains invariant during the course of the computation. On the other hand, the set of edges varies.

- To each A_i node is associated a variable number α_i. To each B_j node is similarly associated a variable number β_j.

- To each edge is also associated a variable number, which will be called a *force*.

6.2 Properties

In the following Section, we will define the operation of the algorithm; simultaneously, we will prove that the following properties hold throughout the computation.

P1 Only the following kinds of edges are allowed: between an A_i and a B_j, between an A_i and Q, and between P and a B_j.

It follows that the graph is *bipartite*, the two subsets of nodes being $\{A_1, \ldots, A_m, P\}$ and $\{B_1, \ldots, B_n, Q\}$.

P2 The graph is a forest, consisting of one or two trees.

P3 When there are two trees, one of them contains P and at least one other node. The other tree contains Q and at least one other node. They will be called respectively T_f or *fixed tree* and T_m or *moving tree*.

P4 $\gamma_{ij} \geq 0$. (γ_{ij} is the *distance* defined by (35)).

P5 If there is an edge between A_i and B_j, then $\gamma_{ij} = 0$.

P6 Forces are positive or zero.

P7 The sum of the forces on the edges adjacent to node A_i equals a_i. The sum of the forces on the edges adjacent to node B_j equals b_j.

6.3 Algorithm

The algorithm consists in a succession of steps, described in the following Sections. Step 1 is executed only once. Then a main loop, made of steps 2 to 5, is executed a number of times; each execution will be called a *cycle*.

Step 1: Initialize

We set up the initial state of the graph and of the associated values as follows. An edge is established between Q and each of the A_i, with associated force a_i, and between P and each of the B_j, with associated force b_j (see Fig. 2). We set $\alpha_i = c_{\text{sup}}$, with c_{sup} satisfying (7), and $\beta_j = 0$. It is easily verified that properties **P1** to **P7** hold. There are two trees.

Step 2: Descent

Since there are two trees, property **P3** ensures that there is at least one node in the moving tree other than Q. Since Q can be connected only to A_i nodes, the moving tree includes at least one A_i node. Similarly, the fixed tree includes at least one B_j node. Therefore the following minimum exists and can be computed:

$$d = \min_{A_i \in T_m, B_j \in T_f} \gamma_{ij}. \tag{36}$$

From property **P4** we have: $d \geq 0$.

Next we effect the "descent of the moving tree":

$$\begin{aligned} \alpha_i &:= \alpha_i - d \qquad \text{for all } A_i \in T_m, \\ \beta_j &:= \beta_j - d \qquad \text{for all } B_j \in T_m. \end{aligned} \tag{37}$$

Note that d may be zero, in which case nothing changes.

We verify now that the properties still hold. Only the α_i and β_j have changed, therefore we have only to examine properties **P4** and **P5**. We consider first **P4**. If

α_i and β_j belong to the same tree, γ_{ij} does not change. If $A_i \in T_f$ and $B_j \in T_m$, γ_{ij} increases. Finally, if $A_i \in T_m$ and $B_j \in T_d$, γ_{ij} decreases by d, but remains positive or zero as a consequence of (36).

We verify also **P5**: if there is an edge between A_i and B_j, these nodes belong to the same tree, and therefore γ_{ij} does not change.

Step 3: Contact

We consider the pair of values $i = i_c$, $j - j_c$ which realized the minimum d in step 2, i.e. which were such that $A_{i_c} \in T_m$, $B_{j_c} \in T_f$, and $\gamma_{i_c j_c} = d$ before the descent. (If more than one pair (i, j) realized the minimum, we select one of them arbitrarily). From (37) we find that there is now, after the descent: $\gamma_{i_c j_c} = 0$.

We add one edge between nodes A_{i_c} and B_{j_c}, and we set the associated force equal to zero.

We consider the properties. **P1** is still satisfied since the new edge is between an A_i and a B_j node. Concerning **P2**, since we have linked one nodes of T_m with one node of T_f, the graph now consists of a single tree. **P3** does not apply any more. **P4** is not affected by a change in the graph. **P5** is satisfied for the new edge since $\gamma_{i_c j_c} = 0$. **P6** is satisfied since the new force is zero. Finally, **P7** still holds for nodes A_{i_c} andB_{j_c}, again because the new force is zero.

Step 4: Readjustment

We define the *main path* as the oriented path from Q to P. This path is unique since the graph consists of a single tree. The main path is made of three parts: (i) the path from Q to A_{i_c} in the previous moving tree; (ii) the newly created edge from A_{i_c} to B_{j_c}; (iii) the path from B_{j_c} to P in the previous fixed tree. From property **P1**, we deduce that the first part has an odd number of edges (the graph is bipartite, and Q and A_i belong to different subsets). Similarly, the last part has an odd number of edges. Thus, the main path as a whole has an odd number of edges, which is at least equal to 3. We number the edges along the main path, from Q to P, starting from 1. We note that the two end edges, adjacent to Q and P, are odd-numbered, and that the newly created edge is even-numbered.

We compute the minimum λ of the forces associated with the odd-numbered edges. There is $\lambda \geq 0$ by virtue of property **P6**.

We readjust the forces along the main path, by adding λ to the forces associated with even-numbered edges and subtracting λ from the forces associated with odd-numbered edges.

Only the forces have changed, therefore we have only to examine properties **P6** and **P7**. Property **P6** is obviously still true. For every node A_i or B_j along the main path, the changes of the forces associated with the two adjacent edges on the main path cancel each other, so that **P7** remains true.

Step 5: Breaking

We consider the odd-numbered edge of the main path which realized the minimum in the previous step. (If more than one edge realized the minimum, we select one of them arbitrarily). We will call it the *breaking edge*. The force associated with that edge is now zero.

We delete the breaking edge. This completes one cycle of the algorithm.

Properties **P1**, **P5**, **P6** are not affected since we have simply removed an edge. The graph consists now again of two trees: **P2** is satisfied. Property **P4** is not affected by a change in the graph. Property **P7** still holds for the two nodes adjacent to the deleted edge since the force associated with that edge was zero.

There remains to consider property **P3**. Since the deleted edge was on a path from Q to P, it is still true that one of the new trees contains Q and the other contains P. However it can happen that Q or P is now an isolated node. We distinguish two cases.

1. The moving tree contains nodes other than Q, and the fixed tree contains nodes other than P. Property **P3** is satisfied. We go back to step 2 for a new cycle.

2. The moving tree contains Q alone, or the fixed tree contains P alone. This signals the end of the computation.

It can be shown that case 2 necessarily happens after a finite number of steps: a definite upper bound on the number of cycles can be computed (see Appendix A). Thus the algorithm always terminates.

6.4 Solution

We show now that a solution of the Hitchcock problem has been obtained. This is similar to the proof given at the end of Section 3; the present derivation, however, makes no reference to the mechanical model.

We will describe only the case where the moving tree contains Q alone; the other case, where the fixed tree contains P alone, is treated in the same way, exchanging rows and columns. We consider the fixed tree. It contains all nodes A_i and B_j, in addition to P. We define f_{ij} as follows: if there is an edge between A_i and B_j, f_{ij} is equal to the associated force; otherwise $f_{ij} = 0$. We also define f_{*j} as follows: if there is an edge between P and B_j, f_{*j} is equal to the associated force; otherwise $f_{*j} = 0$.

By virtue of properties **P1** and **P7**, we have

$$\sum_j f_{ij} = a_i, \qquad (i = 1, \ldots, m), \tag{38}$$

$$\sum_i f_{ij} + f_{*j} = b_j, \qquad (j = 1, \ldots, n). \tag{39}$$

Summing these two equations over i and j respectively and combining with (1), we obtain

$$\sum_j f_{*j} = 0. \tag{40}$$

From property **P6** it follows that

$$f_{*j} = 0 \qquad (j = 1, \ldots, n). \tag{41}$$

The forces are equal to zero on all edges adjacent to P. Therefore f_{ij} satisfies the constraints (3) to (5): it is a feasible solution.

From property **P5**, we derive

$$f_{ij}\gamma_{ij} = 0. \tag{42}$$

Summing over i and j, we obtain

$$\sum_i a_i\alpha_i - \sum_j b_j\beta_j - \sum_i\sum_j f_{ij}c_{ij} = 0. \tag{43}$$

Consider another feasible solution f'_{ij}. From (5) and property **P4** we have

$$f'_{ij}\gamma_{ij} \geq 0 \tag{44}$$

and therefore, summing over i and j and using (3) and (4):

$$\sum_i a_i\alpha_i - \sum_j b_j\beta_j - \sum_i\sum_j f'_{ij}c_{ij} \geq 0. \tag{45}$$

Comparing with (43), we have

$$\sum_i\sum_j f'_{ij}c_{ij} \leq \sum_i\sum_j f_{ij}c_{ij} \tag{46}$$

which shows that f_{ij} is an optimal solution.

7 Notes on practical implementation

7.1 Dropping rows one by one

Experience showed that the following modification of the algorithm results in an important reduction in computing time (typically a factor 3 for $m = n = 100$). The stop Q is cut into m independent pieces $Q_1, \ldots, Q_m$, each supporting one row, and these stops are lowered one by one, each time waiting until an equilibrium has been reached before starting the next stop. This can be done by a simple modification of the algorithm described in Section 6.

7.2 General organization

Measurements show that most of the computing time is spent in the descent phase, and specifically in finding the minimum d in (36). On the other hand, most of the complexity of the program lies in updating the structure of the trees and the associated information. Therefore only the computation of d needs to be optimized for speed. In the remainder of the program, one can freely use the structures which allow the easiest, most natural, and most legible representation.

Experience shows that *triply linked trees* ([9, Section 2.3.3]) are a convenient structure. To each row are associated three pointers to its mother, its eldest daughter, and its next younger brother. Similarly, to each column are associated three pointers to its father, its eldest son, and its next younger sister. The first pointer is used to move upwards in the tree, for instance in order to determine the main path after contact has been made. The two other pointers are used to explore a branch, for instance in order to change its status from moving to fixed, or conversely, after a capture.

Each row has seven quantities associated with it: its weight a_i (which does not change during the computation); its height α_i; the three pointers; the force of contact with its mother; and a flag indicating whether the row belongs to the moving tree or to the fixed tree. Each column has seven similar associated quantities.

7.3 Methods for descent

Finding the minimum d defined by (36) would seem to be a trivial task, involving two loops over i and j and about 15 lines of code. This would require a computing time of order mn for each descent. In fact, there is room for considerable improvement over this simple scheme.

7.3.1 Version A

First we observe that our task is to find the minimum among quantities γ_{ij} which are positive or zero. Therefore if, during the examination of the γ_{ij}, we find one which is zero, then we know that we have found the minimum and we can end the search immediately. We will refer to the program incorporating this simple device as *Version A*.

This is especially effective when the c_{ij} take only a small number of integer values. We will refer to this as the *degenerate case* (see below, Section 8.2). The distances γ_{ij} then take themselves only a small number of distinct values. For large m and n, each of these values appears many times. In particular, as soon as the algorithm is in progress, the value $\gamma_{ij} = 0$ typically appears many times. Therefore the search can be discontinued at an early time and the computing time is much less than $O(mn)$.

When this method is used, experience shows that it is advisable to start the search at a variable point in the γ_{ij} matrix. If the search is always started from the beginning, the contact edge tends to be always selected from the same part of the matrix; an unbalanced situation develops, and computing time increases.

One method consists in choosing the starting point at random in the matrix. This has the disadvantage of requiring the use of a random number generator. A better solution (suggested by A. Noullez) is to compute the rank r' of each new starting point from the rank r of the previous one by a simple formula, such as

$$r' = r + [Kmn] \qquad (\text{mod } mn), \tag{47}$$

where $[\cdot]$ denotes the integer part.

The most uniform distribution of points is obtained by choosing $K = (\sqrt{5} - 1)/2$, the inverse of the golden ratio. This was found to give very good results.

7.3.2 Version B

Another line of attack is based on the realization that the structure of the system changes only partially from one cycle to the next. Therefore we can try to save and re-use information on the distances. (This approach was inspired by a study of the LSAP algorithm presented in [2, chap. 1] for the particular case of the assignment problem.) In particular we may try to take advantage of the tree structure which pervades the algorithm, noting that much of that structure is left intact in a capture episode. We will refer to the program developed along these lines as *Version B*.

Several methods were tried. We describe here the method which gave the best results , and which is incorporated in Version B. We define a *male branch* $\mathcal{A}_i$ as the branch (of the moving or fixed tree) whose head is row A_i. (Note that here is a one-to-one correspondence between rows and male branches). For every pair (i, j), we find the minimal distance δ_{ij} between the rows belonging to the male branch $\mathcal{A}_i$ and the column B_j, and we note for which row this minimal distance is realized. (Note that we consider here all male branches and all columns, irrespective of whether they are moving or fixed). When the structure of the trees changes, this information is updated. At the next descent, the updated information can then be used to compute quickly the minimum d: if the moving stop is Q_i, the whole moving tree consists of the male branch $\mathcal{A}_i$, and one has only to find the minimum among the stored distances δ_{ij} between that male branch and the fixed columns. This takes a time $O(n)$.

The updating of the distances δ_{ij} is somewhat complex. All male branches which have their head on the main path need to be reconsidered. The cases of capture by the fixed tree and by the moving tree require different treatments. Also the three pieces of the main path determined by the contact edge and the breaking edge have to be treated separately. Savings in computing time are achieved by looking for cases where it is not necessary to recompute the δ_{ij}.

Version B is more complex than Version A. It also requires about twice as much memory, since the δ_{ij} array must be saved in addition to the given c_{ij} array. However, it is definitely faster in the general, non-degenerate case (see below, Section 8.1).

8 Tests

Numerical tests were performed to verify the correctness of the algorithm and to measure its performance. The computations were made on a Hewlett-Packard Apollo

Series 700, Model 720 workstation.

Comparisons were made with the subroutine `H03ABF` of the NAG library [10]. In all computed cases (which number in the thousands) it was verified that exactly the same optimal cost (2) is found with the present algorithm and with the NAG algorithm.

The values a_i and b_j are taken as positive integers. They are first chosen at random in the intervals

$$1 \leq a_i \leq a_{\max}, \qquad 1 \leq b_j \leq b_{\max}, \tag{48}$$

where $a_{\max}$ and $b_{\max}$ are two constants satisfying $m a_{\max} = n b_{\max}$. Small adjustments are then made in order to satisfy the relation (1) exactly.

Tests show that the computing time is insensitive to the values of $a_{\max}$ and $b_{\max}$, provided that they are not too close to unity (A variation becomes detectable for values of 10 or less). In practice we take $a_{\max} = 160000/m$, $b_{\max} = 160000/n$.

The values c_{ij} are also taken as integers, randomly chosen in the interval

$$1 \leq c_{ij} \leq c_{\max}. \tag{49}$$

Again the computing time is found to be insensitive to the value of $c_{\max}$, provided that it is large enough. Tests show that the relevant quantity is the ratio

$$c_* = \frac{c_{\max}}{mn}. \tag{50}$$

Variations of the computing time begin to be noticeable when c_* is less than 1 (see below Section 8.2). This corresponds to the onset of degeneracy: for $c_* \ll 1$, each value in the allowed range (49) appears many times in the c_{ij} matrix. Thus, the value taken for $c_{\max}$ depends on whether the non-degenerate or the degenerate case is considered.

For simplicity only the square case $m = n$ was considered, with n ranging from 10 to 1000. Time measurements were generally averaged over a series of 100 computations, in order to obtain more accurate values.

8.1 Non-degenerate case

We take $c_* = 1$. Note that this corresponds to a case where each value in the allowed interval (49) is present once on average in the matrix.

Fig. 7 shows the computing time (divided by n^3 for better clarity) as a function of n, for three algorithms:

- Crosses correspond to the subroutine `H03ABF` of the NAG library [10]. The time appears to grow asymptotically as $n^{3.35}$. (A curious discontinuity is present: the computing time jumps up suddenly by a factor of about 2.2 between the values $n = 94$ and $n = 95$. This is probably due to peculiarities of the computer hardware.)

- Open circles represent Version A of the present algorithm (see Section 7.3). Computing time grows asymptotically as $n^{3.05}$.
- Filled circles represent Version B of the present algorithm (see Section 7.3). For low values of n, the computation is slower than with Version A because of the extra work involved in computing the distances δ_{ij}. Above $n = 100$, however, this extra work begins to pay off. Computing time grows asymptotically as $n^{2.5}$.

Version B is clearly the best method. For a 1000×1000 problem, the NAG subroutine takes about 7000 seconds, while Version B takes about 110 seconds. The ratio increases for larger values of n.

Figure 7: Hitchcock problem: computing time (divided by n^3) as a function of size n. Crosses: NAG subroutine. Open circles: present algorithm, Version A. Filled circles: present algorithm, Version B.

8.2 Degenerate case

A value $c_{\max} = 20$ was chosen as representative for a degenerate problem. In particular, this is a typical value for applications to lattice gas problems [5]. Thus, c_{ij} can take only integer values from 1 to 20.

Fig. 8 shows computing times as a function of n, for two algorithms: the NAG subroutine and Version A of the present algorithm. (Version B is inefficient in the degenerate case and is not shown).

- For the NAG subroutine (crosses), computing time is essentially the same as in the non-degenerate case (Fig. 7) up to about $n = 100$. For larger values, the effect of the degeneracy begins to be felt and the slope decreases. Asymptotically, the computing time appears to grow as about n^2.
- For the present algorithm (open circles), the decrease in computing time with respect to the non-degenerate case is more marked and starts earlier, at about $n = 20$. The time dependence is more complex. The final slope indicates a dependence in $n^{1.65}$.

For a 1000×1000 problem, the NAG subroutine takes 400 seconds, while Version A takes about 1.25 seconds. The ratio again increases for larger values of n.

We remark that an exponent of n less than 2 means that for large values of n, the time needed to solve the problem is small compared to the time needed to set it up, since simply copying the c_{ij} matrix into memory takes a time proportional to n^2 ! Note also that in this situation, most c_{ij} values will never be used.

Figure 8: Degenerate Hitchcock problem: computing time (divided by n^3) as a function of size n. Crosses: NAG subroutine. Open circles: present algorithm, Version A.

8.3 Assignment problem

Tests were also made for the particular case of the assignment problem, where $m = n$ and all rods have weights $a_i = 1$, $b_j = 1$. (It is then easily shown that an optimal solution obtained with the present algorithm automatically satisfies the constraint (10). In that case, comparisons can also be made with the subroutine `LSAP` of Burkard and Derigs [2] (noted BD here).

Fig. 9 compares computing times in the general (non-degenerate) case with $c_* = 1$. As before, the NAG subroutine is much slower. The BD algorithm is fastest: for a 1000×1000 problem, computing time is about 45 seconds for Version B and 22 seconds for the BD algorithm. The difference decreases when n increases, however: the asymptotic law is about $n^{2.4}$ for Version B, compared to $n^{2.7}$ for BD.

Fig. 10 compares NAG, Version A, and BD for the degenerate assignment problem, with $c_{\max} = 20$. Here Version A is fastest: for a 1000×1000 problem, computing time is 0.6 seconds for Version A, and 14 seconds for BD. The difference increases with n: the asymptotic behaviour is in $n^{1.55}$ for Version A, n^2 for BD and NAG.

9 Final comments

1. Many variations of the present algorithm could be imagined, and it is quite possible that some of them would increase its speed. For instance, sometimes an arbitrary choice can be made for the contact edge or the breaking edge (see steps 3 and 5 in Section 6); one might try to determine what is the best choice.

2. As mentioned in the Introduction, the present algorithm was conceived and developed independently, without recourse to the existing litterature. Looking back, however, it becomes clear that some relations exist. The heights of the rods, for instance, are nothing else than the classical *dual variables*; hence the choice of the customary notations α_i and β_j in the present paper.

In particular, after this work was completed, M. Hartmann called our attention to the references [13] and [1]. These papers describe algorithms for the minimum cost flow problem, which includes the Hitchcock problem as a special case. One starts from zero flow and continually increments it, maintaining at all times a minimum-cost solution, until the desired flow is attained. It appears that the algorithm of the present paper belongs essentially to the same family, known as *parametric algorithms*. The equivalent of the flow in the present model is the total force applied by the lines to the columns; as is easily shown, this total force starts from zero and increases with time, until it is equal to the total weight of the lines. More specifically, it increases by λ during each readjustment of the forces (Section 6.3, Step 4).

3. As mentioned in Section 7, most of the computing time in the present algorithm is spent on finding the minimum in a large set of numbers. Efficient algorithms have been developed for this operation on massively parallel computers [6]. A parallel implementation of the algorithm might therefore be of interest.

4. A comparison of figures 7 and 8, or 9 and 10, shows that the solution of the

Figure 9: Assignment problem: computing time (divided by n^3) as a function of size n. Crosses: NAG subroutine. Filled circles: present algorithm, Version B. Asterisks: Burkard and Derigs algorithm.

Figure 10: Degenerate assignment problem: computing time (divided by n^3) as a function of size n. Crosses: NAG subroutine. Open circles: present algorithm, Version A. Asterisks: Burkard and Derigs algorithm.

degenerate problem is much faster. This might be of interest in situations where a great accuracy is not required, or is not present in the data. In such cases, it will be very advantageous to round off the c_{ij} values so as to reduce them to a comparatively small set of values.

5. The idea of using mechanical analog computers for optimization problems is not new. For instance, reference [14] describes a mechanical device made of shafts and gears, which can in principle solve the general instance of the linear programming problem. However, this device is introduced in [14] only as a conceptual tool in a theoretical study of the complexity of analog computation; it is not intended as a model for a practical algorithm. We remark also that the mechanical model of the present paper is adapted to the special case of the Hitchcock problem, and is therefore much simpler (and presumably more efficient) in that special case. As a rough measure, the present model has $O(m + n)$ moving parts, while the model of [14] would have $O(mn)$ moving parts in a $m \times n$ Hitchcock problem.

In this connection, it is natural to ask whether the mechanical model used here to simulate the Hitchcock problem can be extended to the more general minimum cost flow problem, or to the even more general linear programming problem. We have not found any obvious way to do this.

A Bounds on the maximal number of cycles

We call $Z(m, n)$ the maximal number of cycles, for the $m \times n$ problem, using the algorithm described in Section 6. We derive here some rigorous bounds on this number.

A.1 Upper bound

We derive first a general upper bound for $Z(m, n)$.

We number with an index l the successive levels of the moving tree. The stop Q is at level $l = 0$, the sons of Q are at level $l = 1$, the grand-daughters of Q are at level $l = 2$, and so on. Note that odd levels correspond to rows and even levels to columns. We call g_l the number of nodes of the tree at level l. The sequence of numbers g_1, g_2, ..., will be called the *signature* of the moving tree.

We consider now all possible signatures for given m and n, assuming that the final state has not yet been reached, i.e. that the moving tree still contains at least one row and the fixed tree still contains at least one column. We order these signatures as follows. First we sort by decreasing g_1. Next we sort each subset by increasing g_2. Next we sort each subsubset (corresponding to given g_1 and g_2) by decreasing g_3; and so on, always using decreasing order for odd values of l and increasing order for even values. Finally, we number the sorted signatures with $K = 1, 2, \dots$.

It is then easy to show that K *always increases during a cycle*. There are two cases:

1. The moving tree captures a subtree from the fixed tree. The root of this subtree (after the capture) is a column. Therefore the first level l at which there is a

change in the signature is even, and g_l increases by one unit. From the above sorting method it follows that K increases.

2. The moving tree loses a subtree. The root of this subtree (before the capture) is a row. Therefore the first level l at which there is a change in the signature is odd, and g_l decreases by one unit. Again K increases.

Therefore we obtain an upper bound on the number of cycles simply by counting the signatures. We call $p = g_1 + g_3 + \dots$ the number of rows, and $q = g_2 + g_4 + \dots$ the number of columns in the moving tree. Since the moving tree is assumed to be non-empty, p can take values from 1 to m. Similarly, since the fixed tree is non-empty, q can take values from 0 to $n-1$. We evaluate first the number of signatures for given p and q. A signature can also be represented by a sequence of $p+q$ binary digits: we write g_1 digits 1, then g_2 digits 0, then g_3 digits 1, and so on. The first digit must be a 1. There are q digits 0, which can be placed anywhere in the remaining $p+q-1$ positions. Therefore the number of possible signatures is

$$\binom{p+q-1}{q}. \tag{51}$$

Summing over p and q, we obtain the following upper bound for Z:

$$Z(m,n) \leq Z_{\text{sup}}(m,n) = \binom{m+n}{m} - 1. \tag{52}$$

Exactly the same considerations can be applied to the fixed tree. We number with l the successive levels. We call g'_l the number of nodes at level l. The sequence of numbers g'_1, g'_2, …, will be called the signature of the fixed tree. We sort the signatures, using decreasing order for odd l (columns) and increasing order for even l (rows). We number the sorted signatures with $K' = 1, 2, \dots$. The number K' also always increases during a cycle. We call $q' = g'_1 + g'_3 + \dots$ the number of columns, and $p' = g'_2 + g'_4 + \dots$ the number of rows in the fixed tree. Counting the number of signatures as above, we find

$$\binom{m+n}{n} - 1 \tag{53}$$

i.e. exactly the same upper bound as in (52).

A.2 Better upper bound

A much better upper bound can be obtained by noting that only some combinations of K and K' are permitted. This is because we must have

$$p + p' = m, \qquad q + q' = n. \tag{54}$$

Thus, in a (K, K') plane, only a subset of points are allowed. Combining this with the fact that both K and K' must increase at each step, one can trace the possible paths in the plane and derive an upper limit on the number of steps, which we call $Z'_{\text{sup}}(m, n)$.

Unfortunately a general formula giving $Z'_{\text{sup}}(m, n)$ for arbitrary m and n has not been found. Results obtained by a computer program for values of m and n up to 10 are listed in Table 3.

Table 3: Upper limit $Z'_{\text{sup}}(m, n)$ on the number of cycles for $1 \leq m \leq 10$, $1 \leq n \leq 10$.

		n									
		1	2	3	4	5	6	7	8	9	10
m	1	1	2	3	4	5	6	7	8	9	10
	2	2	4	6	8	10	12	14	16	18	20
	3	3	6	10	14	19	24	30	36	43	50
	4	4	8	14	22	30	40	52	64	78	94
	5	5	10	19	30	46	62	83	108	138	170
	6	6	12	24	40	62	94	126	168	222	284
	7	7	14	30	52	83	126	190	254	339	448
	8	8	16	36	64	108	168	254	382	510	682
	9	9	18	43	78	138	222	339	510	766	1022
	10	10	20	50	94	170	284	448	682	1022	1534

For $m = n = 10$, for instance, we have $Z_{\text{sup}}(m, n) = 184755$, $Z'_{\text{sup}}(m, n) = 1534$.

A.3 Lower bound

In the square case $m = n$, the following lower bound can be proved:

$$Z(n, n) \geq Z_{\text{inf}}(n, n) = 3 \times 2^{n-1} - 2. \tag{55}$$

The proof is cumbersome and will not be given here. It consists in showing that the number of cycles equals Z_{inf} in the following case:

$$\begin{aligned}
a_1 &= 1, \qquad b_1 = 2, \\
a_i &= a_{i-1} + b_{i-1}, \qquad b_i = b_{i-1} + a_i, \qquad (i = 2, \ldots, n-1), \\
a_n &= a_{n-1} + b_{n-1}, \qquad b_n = b_{n-1}, \\
c_{ij} &= \begin{cases} (n+1-i)(n+1-j) - n^2(2^i + 2^j) & \text{if } i = j, \\ (n+1-i)(n+1-j) - n^2(2^j) & \text{if } i < j, \\ (n+1-i)(n+1-j) - n^2(2^i) & \text{if } i > j. \end{cases}
\end{aligned} \tag{56}$$

This was also verified by a direct application of the numerical algorithm for $n = 1$ to 17. Note that the sequence a_1, b_1, a_2, b_2, ... is the Fibonacci sequence, minus its first term.

Comparing with the diagonal of Table 3, we find that the upper bound given in that Table is identical to the lower bound given by (55). Therefore, for $n = 1$ to 10, we know the exact value of the maximal number of cycles, which is

$$Z(n, n) = 3 \times 2^{n-1} - 2, \tag{57}$$

and (56) is a worst case, achieving this maximal value. There is a strong suggestion that (57) holds for all values of n, but this has not been proved.

A.4 Comparison with observed values

We observe that (57) corresponds to a computing time which grows exponentially with n. Fortunately, numerical tests with randomly chosen examples show a much milder increase, which is approximately linear in n. Table 4 compares the values (57) with the average observed values of the number of cycles, for Version B of the algorithm, in the non-degenerate case. The r.m.s. dispersions are also given; they show that individual values do not deviate much from the average.

Table 4: Number of cycles in the "bad" case (56) (column 2), and observed number of cycles (column 3).

n	$Z_{\mathrm{inf}}(n, n)$	observed
5	46	10 ± 1
10	1534	23 ± 2
15	49150	37 ± 3
20	1572862	50 ± 4
25	50331646	64 ± 5
30	1610612734	80 ± 6
50		141 ± 10
100		304 ± 17

This considerable difference between the "bad case" (56)and the average case can be probably understood by noting that (56) is a rather extreme case: the coefficients a_i, b_j, c_{ij} form essentially geometrical progressions. For $m = 1000$, for instance, the ratio a_{1000}/a_1 is of the order of 10^{400}. This is not likely to be encountered in applications.

A.5 Acknowledgements

I thank P. Bernhard, U. Frisch, M. Hartmann, J. Morgenstern, A. Noullez, K. Steiglitz, and S. Stidham for discussions and comments.

References

[1] R. K. Ahuja, J. L. Batra, and S. K. Gupta, "A parametric algorithm for convex cost network flow and related problems," *European Journal of Operational Research* 16 (1984) 222–235.

[2] R. E. Burkard and U. Derigs, *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*, Lecture Notes in Economics and Mathematical Systems No. 184 (Springer, Berlin, 1980).

[3] L. R. Ford and D. R. Fulkerson, *Flows in Networks* (Princeton University Press, 1962).

[4] G. Hadley, *Linear Programming* (Addison-Wesley, Reading, Massachusetts, 1962).

[5] M. Hénon, "Optimization of collision rules in the FCHC lattice gas, and addition of rest particles," in: R. Monaco, ed., *Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics*, Proceedings of a Workshop held in Torino, September 20-24, 1988 (World Scientific, 1989) pp. 146–159.

[6] W. D. Hillis and G. L. Steele, "Data Parallel Algorithms," *Communications of the Association for Computing Machinery* 29 (1986) 1170–1183.

[7] F. L. Hitchcock, "The distribution of a product from several sources to numerous localities," *Journal of Mathematical Physics* 20 (1941) 224–230.

[8] D. d'Humières, P. Lallemand, and U. Frisch, "Lattice gas models for 3D hydrodynamics," *Europhysics Letters* 2 (1986) 291–297.

[9] D. Knuth, *The Art of Computer Programming*, Volume 1 (Addison-Wesley, Reading, Massachusetts, 1973).

[10] *NAG Fortran Library, Mark 15* (NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford, United Kingdom, 1991).

[11] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, New Jersey, 1982).

[12] J.-P. Rivet, M. Hénon, U. Frisch, and D. d'Humières, "Simulating fully three-dimensional external flow by lattice gas methods," *Europhysics Letters* 7 (1988) 231–236.

[13] S. R. Schmidt, P. A. Jensen, and J. W. Barnes, "An advanced dual incremental network algorithm," *Networks* 12 (1982) 475–492.

[14] A. Vergis, K. Steiglitz, and B. Dickinson, "The complexity of analog computation," *Mathematics and Computers in Simulation* 28 (1986) 91–113.