Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Català
Deutsch
Español
فارسی
Français
Հայերեն
Italiano
Magyar
Nederlands
Polski
Русский
ไทย
中文
Edit links

Hungarian algorithm

From Wikipedia, the free encyclopedia

The **Hungarian method** is a combinatorial optimization algorithm that solves the assignment problem in polynomial time and which anticipated later primal-dual methods. It was developed and published in 1955 by Harold Kuhn, who gave the name "Hungarian method" because the algorithm was largely based on the earlier works of two Hungarian mathematicians: Dénes Kőnig and Jenő Egerváry.[1][2]

James Munkres reviewed the algorithm in 1957 and observed that it is (strongly) polynomial.[3] Since then the algorithm has been known also as the **Kuhn–Munkres algorithm** or **Munkres assignment algorithm**. The time complexity of the original algorithm was $O(n^4)$, however Edmonds and Karp, and independently Tomizawa noticed that it can be modified to achieve an $O(n^3)$ running time. Ford and Fulkerson extended the method to general transportation problems. In 2006, it was discovered that Carl Gustav Jacobi had solved the assignment problem in the 19th century, and the solution had been published posthumously in 1890 in Latin.[4]

Contents [hide]

1 Layman's explanation of the assignment problem
2 Setting
3 The algorithm in terms of bipartite graphs
4 Matrix interpretation
5 Bibliography
6 References
7 External links
7.1 Implementations

Layman's explanation of the assignment problem [edit]

Say you have three workers: Jim, Steve, and Alan. You need to have one of them clean the bathroom, another sweep the floors, and the third wash the windows, but they each demand different pay for the various tasks. What's the lowest-cost way to assign the jobs? The problem can be represented in a matrix of the costs of the workers doing the jobs. For example:

	Clean bathroom	Sweep floors	Wash windows
Jim	$2	$3	$3
Steve	$3	$2	$3
Alan	$3	$3	$2

The Hungarian method, when applied to the above table, would give us the minimum cost: this is $6, achieved by having Jim clean the bathroom, Steve sweep the floors, and Alan wash the windows.

Setting [edit]

We are given a nonnegative *n*×*n* matrix, where the element in the *i*-th row and *j*-th column represents the cost of assigning the *j*-th job to the *i*-th worker. We have to find an assignment of the jobs to the workers that has minimum cost. If the goal is to find the assignment that yields the maximum cost, the problem can be altered to fit the setting by replacing each cost with the maximum cost subtracted by the cost.

The algorithm is easier to describe if we formulate the problem using a bipartite graph. We have a complete bipartite graph $G = (S, T; E)$ with n worker vertices (S) and n job vertices (T), and each edge has a nonnegative cost $c(i, j)$. We want to find a perfect matching with minimum cost.

Let us call a function $y : (S \cup T) \mapsto \mathbb{R}$ a **potential** if $y(i) + y(j) \leq c(i, j)$ for each $i \in S, j \in T$. The value of potential y is $\sum_{v \in S \cup T} y(v)$. It can be seen that the cost of each perfect matching is at least the value of each potential. The Hungarian method finds a perfect matching and a potential with equal cost/value which proves the optimality of both. In fact it finds a perfect matching of **tight edges**: an edge ij is called tight for a potential y if $y(i)+y(j)=c(i,j)$. Let us denote the subgraph of tight edges by G_y. The cost of a perfect matching in G_y (if there is one) equals the value of y.

The algorithm in terms of bipartite graphs [edit]

During the algorithm we maintain a potential *y* and an orientation of G_y (denoted by $\overrightarrow{G_y}$) which has the property that the edges oriented from *T* to *S* form a matching *M*. Initially, *y* is 0 everywhere, and all edges are oriented from *S* to *T* (so *M* is empty). In each step, either we modify *y* so that its value increases, or modify the orientation to obtain a matching with more edges. We maintain the invariant that all the edges of *M* are tight. We are done if *M* is a perfect matching.

In a general step, let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by *M* (so R_S consists of the vertices in *S* with no incoming edge and R_T consists of the vertices in *T* with no outgoing edge). Let Z be the set of vertices reachable in $\overrightarrow{G_y}$ from R_S by a directed path only following edges that are tight. This can be computed by breadth-first search.

If $R_T \cap Z$ is nonempty, then reverse the orientation of a directed path in $\overrightarrow{G_y}$ from R_S to R_T. Thus the size of the corresponding matching increases by 1.

If $R_T \cap Z$ is empty, then let $\Delta := \min\{c(i,j)-y(i)-y(j) : i \in Z \cap S, j \in T \setminus Z\}$. Δ is positive because there are no tight edges between $Z \cap S$ and $T \setminus Z$. Increase *y* by Δ on the vertices of $Z \cap S$ and decrease *y* by Δ on the vertices of $Z \cap T$. The resulting *y* is still a potential. The graph G_y changes, but it still contains *M*. We orient the new edges from *S* to *T*. By the definition of Δ the set *Z* of vertices reachable from R_S increases (note that the number of tight edges does not necessarily increase).

We repeat these steps until *M* is a perfect matching, in which case it gives a minimum cost assignment. The running time of this version of the method is $O(n^4)$: *M* is augmented *n* times, and in a phase where *M* is unchanged, there are at most *n* potential changes (since *Z* increases every time). The time needed for a potential change is $O(n^2)$.

Matrix interpretation [edit]

Given n workers and tasks, and an *n*×*n* matrix containing the cost of assigning each worker to a task, find the cost minimizing assignment.

First the problem is written in the form of a matrix as given below

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \\ d1 & d2 & d3 & d4 \end{bmatrix}$$

where a, b, c and d are the workers who have to perform tasks 1, 2, 3 and 4. a1, a2, a3, a4 denote the penalties incurred when worker "a" does task 1, 2, 3, 4 respectively. The same holds true for the other symbols as well. The matrix is square, so each worker can perform only one task.

Step 1

Then we perform row operations on the matrix. To do this, the lowest of all a_i (i belonging to 1-4) is taken and is subtracted from each element in that row. This will lead to at least one zero in that row (We get multiple zeros when there are two equal elements which also happen to be the lowest in that row). This procedure is repeated for all rows. We now have a matrix with at least one zero per row. Now we try to assign tasks to agents such that each agent is doing only one task and the penalty incurred in each case is zero. This is illustrated below.

0	a2'	a3'	a4'
b1'	b2'	b3'	0
c1'	0	c3'	c4'
d1'	d2'	0	d4'

The zeros that are indicated as 0' are the assigned tasks.

Step 2

Sometimes it may turn out that the matrix at this stage cannot be used for assigning, as is the case in for the matrix below.

0	a2'	a3'	a4'
b1'	b2'	b3'	0
0	c2'	c3'	c4'
d1'	0	d3'	d4'

In the above case, no assignment can be made. Note that task 1 is done efficiently by both agent a and c. Both can't be assigned the same task. Also note that no one does task 3 efficiently. To overcome this, we repeat the above procedure for all columns (i.e. the minimum element in each column is subtracted from all the elements in that column) and then check if an assignment is possible.

In most situations this will give the result, but if it is still not possible then we need to keep going.

Step 3

All zeros in the matrix must be covered by marking as few rows and/or columns as possible. The following procedure is one way to accomplish this:

First, assign as many tasks as possible.

- Row 1 has one zero, so it is assigned. The 0 in row 3 is crossed out because it is in the same column.
- Row 2 has one zero, so it is assigned.
- Row 3's only zero has been crossed out, so nothing is assigned.
- Row 4 has two uncrossed zeros. Either one can be assigned (both are optimum), and the other zero would be crossed out.

Alternatively, the 0 in row 3 may be assigned, causing the 0 in row 1 to be crossed instead.

0'	a2'	a3'	a4'
b1'	b2'	b3'	0'
0	c2'	c3'	c4'
d1'	0'	0	d4'

Now to the drawing part.

- Mark all rows having no assignments (row 3).
- Mark all (unmarked) columns having zeros in newly marked row(s) (column 1).
- Mark all rows having assignments in newly marked columns (row 1).
- Repeat for all non-assigned rows.

×				
0'	a2'	a3'	a4'	×
b1'	b2'	b3'	0'	
0	c2'	c3'	c4'	×
d1'	0'	0	d4'	

Now draw lines through all marked columns and **unmarked** rows.

×				
0'	a2'	a3'	a4'	×
b1'	b2'	b3'	0'	
0	c2'	c3'	c4'	×
d1'	0'	0	d4'	

The aforementioned detailed description is just one way to draw the minimum number of lines to cover all the 0s. Other methods work as well.

Step 4

Now remove the marked rows and columns. This will leave a matrix as follows:

$$\begin{bmatrix} a2 & a3 & a4 \\ c2 & c3 & c4 \end{bmatrix}$$

Return to step 1 and repeat the process until the matrix is empty.

Bibliography [edit]

- R.E. Burkard, M. Dell'Amico, S. Martello: *Assignment Problems* (Revised reprint). SIAM, Philadelphia (PA.) 2012. ISBN 978-1-61197-222-1
- M. Fischetti, "Lezioni di Ricerca Operativa", Edizioni Libreria Progetto Padova, Italia, 1995.
- R. Ahuja, T. Magnanti, J. Orlin, "Network Flows", Prentice Hall, 1993.
- S. Martello, "Jeno Egerváry: from the origins of the Hungarian algorithm to satellite communication". Central European Journal of Operations Research 18, 47–58, 2010

References [edit]

1. ^ Harold W. Kuhn, "The Hungarian Method for the assignment problem", *Naval Research Logistics Quarterly*, **2**: 83–97, 1955. Kuhn's original publication.
2. ^ Harold W. Kuhn, "Variants of the Hungarian method for assignment problems", *Naval Research Logistics Quarterly*, **3**: 253–258, 1956.
3. ^ J. Munkres, "Algorithms for the Assignment and Transportation Problems", *Journal of the Society for Industrial and Applied Mathematics*, **5**(1):32–38, 1957 March.
4. ^ http://www.lix.polytechnique.fr/~ollivier/JACOBI/jacobiEngl.htm

External links [edit]

- Bruff, Derek, "The Assignment Problem and the Hungarian Method", [1]
- Mordecai J. Golin, Bipartite Matching and the Hungarian Method, Course Notes, Hong Kong University of Science and Technology.
- R. A. Pilgrim, *Munkres' Assignment Algorithm. Modified for Rectangular Matrices*, Course notes, Murray State University.
- Mike Dawes, *The Optimal Assignment Problem*, Course notes, University of Western Ontario.
- On Kuhn's Hungarian Method – A tribute from Hungary, András Frank, Egervary Research Group, Pazmany P. setany 1/C, H1117, Budapest, Hungary.
- Lecture: Fundamentals of Operations Research - Assignment Problem - Hungarian Algorithm, Prof. G. Srinivasan, Department of Management Studies, IIT Madras.
- Extension: Assignment sensitivity analysis (with O(n^4) time complexity), Liu, Shell.
- Solve any Assignment Problem online, provides a step by step explanation of the Hungarian Algorithm.

Implementations [edit]

(Note that not all of these satisfy the $O(n^3)$ time constraint.)

- C implementation with $O(n^3)$ time complexity
- Java implementation of $O(n^3)$ time variant
- Python implementation (see also here)
- Ruby implementation with unit tests
- C# implementation
- D implementation with unit tests (port of the Java $O(n^3)$ version)
- Online interactive implementation Please note that this implements a variant of the algorithm as described above.
- Graphical implementation with options (Java applet)
- Serial and parallel implementations.
- Implementation in Matlab and C
- Perl implementation
- Lisp implementation
- C++ (STL) implementation (multi-functional bipartite graph version)
- C++ implementation
- C++ implementation of the $O(n^3)$ algorithm (BSD style open source licensed)
- Another C++ implementation with unit tests
- Another Java implementation with JUnit tests (Apache 2.0)
- MATLAB implementation
- C implementation
- Javascript implementation
- The clue R package proposes an implementation, solve_LSAP

Categories: Matching | Combinatorial optimization