# 匈牙利算法

维基百科，自由的百科全书

**匈牙利算法**是一种在多项式时间内求解任务分配问题的组合优化算法，并推动了后来的原始对偶方法。美国数学家哈罗德·库恩于1955年提出该算法。此算法之所以被称作匈牙利算法，是因为算法很大一部分是基于以前匈牙利数学家Dénes Kőnig和Jenő Egerváry的工作之上创建起来的。[1][2]

詹姆士·芒克勒斯在1957年回顾了该算法，并发现（强）多项式时间的。[3] 此后该算法被称为**Kuhn–Munkres算法**或**Munkres分配算法**。原始算法的时间复杂度为$O(n^4)$，但杰克·爱德蒙斯与卡普发现可以修改算法达到$O(n^3)$运行时间，富泽也独立发现了这一点。L·R·福特和D·R·福尔克森将该方法推广到了一般运输问题。2006年发现卡爾·雅可比在19世纪就解决了指派问题，该解法在他死后在1890年以拉丁文发表。[4]

## 目录



## Layman对指派问题的解释

你有三个工人：吉姆，史提夫和艾伦。 你需要其中一个清洁浴室，另一个打扫地板，第三个洗窗，但他们每个人对各项任务要求不同数目数量的钱。 以最低成本的分配工作的方式是什么？ 可以用工人做工的成本矩阵来表示该问题。例如：

| | 清洁浴室 | 打扫地板 | 洗窗 |
|---|---|---|---|
| 吉姆 | $2 | $3 | $3 |
| 史提夫 | $3 | $2 | $3 |
| 艾伦 | $3 | $3 | $2 |

当把匈牙利方法应用于上面的表格时，会给出最低成本：为6美元，让吉姆清洁浴室、史提夫打扫地板、艾伦清洗窗户就可以达到这一结果。

## 设定

给定一个 $n \times n$ 的非负矩阵，其中第 $i$ 行第 $j$ 列元素表示把第 $i$ 个工人派到第 $j$个工作的成本。我们必须找到成本最低的工人工作分配。如果目标是找到最高成本的分配，该问题可以将每个成本都换为最高一个成本减去该成本以适应题目。

如果用二分图来阐述该问题可以更容易描述这个算法。对于一个有 $n$个工人节点（$S$）与 $n$个工作节点（$T$）的完全二分图 $G=(S \cup T, E)$，每条边都有 $c(i,j)$ 的非负成本。我们要找到最低成本的完美匹配。

如果函数 $y:(S \cup T) \mapsto \mathbb{R}$ 满足对于每个 $i \in S, j \in T$ 都有 $y(i)+y(j) \leq c(i,j)$，则把该函数叫做**势**。势 $y$ 的值为 $\sum_{v \in S \cup T} y(v)$。可以看出，每个完美匹配的成本最低是每个势的值。匈牙利算法找出了完美匹配及与之成本/值相等的势，这证明了两者的最优性。实际上它找到了**紧边集**的完美匹配：紧边 $ij$ 是指对于势 $y$ 满足 $y(i)+y(j)=c(i,j)$。我们将紧边子图表示为 $G_y$。$G_y$ 中的完美匹配的成本（如果存在）就等于 $y$的值。

## 用二分图描述此算法

在算法中我们維持 $G_y$ 的势$y$和方向（表示为 $\overrightarrow{G_y}$ ），该方向有从 $T$到 $S$ 的边构成匹配 $M$ 的性质。初始时，$y$ 处处为 $0$，所有边都由 $S$ 指向 $T$（因此 $M$ 为空）。每一步中，我们或者改变 $y$ 使其值增加，或者改变方向以得到更多的边。我们保持 $M$ 的所有边都是紧边不发生改变。当 $M$ 为完美匹配时结束。

在一般的步骤中，令 $R_S \subseteq S$ 和 $R_T \subseteq T$ 为 $M$ 未覆盖的节点（则 $R_S$ 包含 $S$ 中入度为零的节点，而 $R_T$ 中包含 $T$ 中出度为零的节点）。令 $Z$ 为从 $R_S$ 只沿紧边的有向路径可到达 $\overrightarrow{G_y}$ 的节点。可由广度优先搜索求得。

若 $R_T \cap Z$ 非空，则将 $\overrightarrow{G_y}$ 中从 $R_S$ 到 $R_T$ 的有向路径反向。则相应匹配数增加$1$。

若 $R_T \cap Z$ 为空，则令 $\Delta := \min\{c(i,j)-y(i)-y(j) : i \in Z \cap S, j \in T \setminus Z\}$。$\Delta$ 为正，因为 $Z \cap S$ 与 $T \setminus Z$ 之间没有紧边。在 $Z \cap T$ 中的节点将$y$增加$\Delta$ 并在 $Z \cap S$ 中节点将 $y$减小 $\Delta$，得到的$y$仍然是势。图 $G_y$ 改变了，但它仍包含$M$。我们把新的边从$S$指向$T$。由 $\Delta$ 的定义，$R_S$ 可达的节点集 $Z$ 增大（注意到紧边的数量不一定增加）。

我们重复这些步骤直到$M$为完美匹配，该情形下给出的是最小成本（即时间消耗）的匹配。此版本的运行时间为 $O(n^4)$：$M$ 增广 $n$次，在 $M$ 不改变的一个阶段中，势最多改变 $n$ 次（因为 $Z$ 每次都增加）。改变势所需的时间在 $O(n^2)$。

### 证明：改变势函数$y$不改变$M$

证明改变$y$后$M$中每条边不发生改变，这等价于证明对于$M$中任意边，它的两个顶点要么都在$Z$中，要么都不在$Z$中。为此，定义$vu$为$M$中一条从$T$到$S$的边，则若$v \in Z$，那么有$u \in Z$。（反证）假设$u \in Z$但$v \notin Z$；由于$u$是一个匹配边的末端点，$u \notin R_S$，因此存在从$R_S$到$u$的有向路径。这条路

径必不能经过$v$（根据假设），因此这条路径上紧邻$u$的点是其他点$v' \in T$。$v'u$是一条从$T$到$S$的紧边因此也是$M$中的一个元素。但因此$M$包含两条有共点的边，与$M$是匹配的定义矛盾。因此$M$中每条边的两个顶点要么都在$Z$中，要么都不在$Z$中。

### 证明：$y$仍然是势函数

证明$y$在更改之后是势函数，等价于证明不存在总势超过成本的边。这已经在之前的论述中已经为$M$中的边建立这个概念，因此我们考虑任意从$S$到$T$的边$uv$。如果$y(u)$升高了$\Delta$，那么：(1) $v \in Z \cap T$，在这种情况下，$y(v)$减小了$\Delta$，使总体的势未发生改变；(2)$v \in T \setminus Z$，这种情况下$\Delta$的定义保证了$y(u) + y(v) + \Delta \leq c(u, v)$。因此$y$仍然是势函数。

## 矩阵解释

给定 $n$ 个工人和任务，以及一个包含分配给每个工人一个任务的成本的 $n \times n$矩阵，寻找成本最小化分配。

首先把问题写成下面的矩阵形式

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}$$

其中 $a, b, c, d$是执行任务 1, 2, 3, 4 的工人。$a_1, a_2, a_3, a_4$ 分别表示当工人 $a$ 做任务 1, 2, 3, 4 时的时间损失（成本）。对于其他符号也同样适用。该矩阵是方阵，所以每个工人只能执行一个任务。

**第 1 步**

接下来我们对矩阵的行进行操作。将所有 $a_i$ （$i$ 从 1 到 4）中最小的元素取走，并将该行每个元素都减去刚刚取走的元素。这会让该行至少出现一个零（当一行有两个相等的最小元素时会得到多个零）。对此过程为所有行重复。我们现在得到一个每行至少有一个零的矩阵。现在我们尝试给工人指派任务，以使每个工人只做一项任务，并且每个情况的耗散都为零。说明如下。

| | | | |
|---|---|---|---|
| 0 | $a'_2$ | $a'_3$ | $a'_4$ |
| $b'_1$ | $b'_2$ | $b'_3$ | 0 |
| $c'_1$ | 0 | $c'_3$ | $c'_4$ |
| $d'_1$ | $d'_2$ | 0 | $d'_4$ |

用 0' 表示的零为已指派的任务。

**第 2 步**

有时此阶段的该矩阵不能符合指派的要求，例如下面所示矩阵。

| | | | |
|---|---|---|---|
| $0$ | $a'_2$ | $a'_3$ | $a'_4$ |
| $b'_1$ | $b'_2$ | $b'_3$ | $0$ |
| $0$ | $c'_2$ | $c'_3$ | $c'_4$ |
| $d'_1$ | $0$ | $d'_3$ | $d'_4$ |

在上述情形下，不能做出指派。注意到任务 $1$ 由工人 $a$ 和 $c$ 做都很高效。只是不能把两个工人分配到同一个任务中去。还注意到，没有任何一个工人能有效地做任务 $3$。 为了克服这个问题，我们对所有列重复上述流程（即每一列所有元素都减去该列最小元素）并检查是否可以完成分配。

大多数情况下，这都会给出结果，但如果仍然是不可行，那么我们需要继续下去。

**第 3 步**

必须用尽可能少的列或行标记来覆盖矩阵中的所有零。下面的过程是完成这个要求的一种方法：

首先，尽可能多地分配任务。

- 第 1 行有一个零，所以分配了。第 3 列的 0 由于处于同一列而被划掉。
- 第 2 行有一个零，所以分配了。
- 第 3 行只有一个已经划掉的零，所以不能分配。
- 第 4 行有两个未划掉的零。可以分配任何一个（都是最优），并将另一个零划去。

或者，分配的是第 $3$ 行的 $0$，就会使第 $1$ 行的 $0$ 被划掉。

| | | | |
|---|---|---|---|
| $0'$ | $a'_2$ | $a'_3$ | $a'_4$ |
| $b'_1$ | $b'_2$ | $b'_3$ | $0'$ |
| $0$ | $c'_2$ | $c'_3$ | $c'_4$ |
| $d'_1$ | $0'$ | $0$ | $d'_4$ |

现在到了画图的部分。

- 标记所有未分配的行（第 3 行）。
- 标记所有新标记的行中 0所在（且未标记）的對應列（第 1 列）。
- 标记所有在新标记的列中有分配的行（第 1 行）。
- 对所有未分配的行重复上述过程。

| | | | | |
|---|---|---|---|---|
| × | | | | |
| $0'$ | $a'_2$ | $a'_3$ | $a'_4$ | × |
| $b'_1$ | $b'_2$ | $b'_3$ | $0'$ | |
| $0$ | $c'_2$ | $c'_3$ | $c'_4$ | × |
| $d'_1$ | $0'$ | $0$ | $d'_4$ | |

现在劃掉所有已标记的列和**未标记**的行（第 $1$ 列和第 $2, 4$ 行）。

| | | | | |
|---|---|---|---|---|
| × | | | | |
| $0'$ | $a'_2$ | $a'_3$ | $a'_4$ | × |
| $b'_1$ | $b'_2$ | $b'_3$ | $0'$ | |
| $0$ | $c'_2$ | $c'_3$ | $c'_4$ | × |
| $d'_1$ | $0'$ | $0$ | $d'_4$ | |

上述详细的描述只是画出覆盖所有 $0$ 的（直、行）线的一种方法。也可以使用其他方法。

**第 4 步**

现在删除已畫線的行和列。这将留下一个矩阵如下：

$$\begin{bmatrix} a_2 & a_3 & a_4 \\ c_2 & c_3 & c_4 \end{bmatrix}$$

返回到步骤 $1$，然后重复这个过程，直到矩阵是空的。

## 参考书目


- 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


## 参考文献


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. JACOBI'S BOUND (http://www.lix.polytechnique.fr/~ollivier/JACOBI/jacobiEngl.htm)


## 外部链接

- Bruff, Derek, "The Assignment Problem and the Hungarian Method", [1] (http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf) （页面存档备份 (https://web.archive.org/web/20120105112913/http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf)，存于互联网档案馆）
- Mordecai J. Golin, Bipartite Matching and the Hungarian Method (https://web.archive.org/web/20160303183021/http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf), Course Notes, Hong Kong University of Science and Technology.
- R. A. Pilgrim, *Munkres' Assignment Algorithm. Modified for Rectangular Matrices (http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html)* *（页面存档备份 (https://web.archive.org/web/20*

*070327101101/http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html)，存于互联网档案馆）*, Course notes, Murray State University.
- Mike Dawes, *The Optimal Assignment Problem (https://web.archive.org/web/20060812030313/http://www.math.uwo.ca/~mdawes/courses/344/kuhn-munkres.pdf)*, Course notes, University of Western Ontario.
- On Kuhn's Hungarian Method – A tribute from Hungary (http://www.cs.elte.hu/egres/tr/egres-04-14.pdf), András Frank, Egervary Research Group, Pazmany P. setany 1/C, H1117, Budapest, Hungary.
- Lecture: Fundamentals of Operations Research - Assignment Problem - Hungarian Algorithm (https://www.youtube.com/watch?v=BUGIhEecipE), Prof. G. Srinivasan, Department of Management Studies, IIT Madras.
- Extension: Assignment sensitivity analysis (with O(n^4) time complexity) (http://www.roboticsproceedings.org/rss06/p16.html), Liu, Shell.
- Solve any Assignment Problem online (http://www.hungarianalgorithm.com/solve.php), provides a step by step explanation of the Hungarian Algorithm.

## 实现

（请注意，并非所有这些都满足 $O(n^3)$ 时间约束。）

- C implementation with $O(n^3)$ time complexity (https://github.com/maandree/hungarian-algorithm-n3/blob/master/hungarian.c)
- Java implementation of $O(n^3)$ time variant (https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software_and_algorithms/stern_library/optimization/HungarianAlgorithm.java)
- Python implementation (http://software.clapper.org/munkres/) (see also here (https://github.com/xtof-durr/makeSimple/blob/master/Munkres/kuhnMunkres.py))
- Ruby implementation with unit tests (https://github.com/evansenter/gene/blob/f515fd73cb9d6a22b4d4b146d70b6c2ec6a5125b/objects/extensions/hungarian.rb)
- C# implementation (http://noldorin.com/blog/2009/09/hungarian-algorithm-in-csharp/)
- D implementation with unit tests (port of the Java $O(n^3)$ version) (http://www.fantascienza.net/leonardo/so/hungarian.d)
- Online interactive implementation (http://www.ifors.ms.unimelb.edu.au/tutorial/hungarian/welcome_frame.html) Please note that this implements a variant of the algorithm as described above.
- Graphical implementation with options (https://web.archive.org/web/20151228053040/http://web.axelero.hu/szilardandras/gaps.html) (Java applet)
- Serial and parallel implementations. (http://www.netlib.org/utk/lsi/pcwLSI/text/node220.html)
- Implementation in Matlab and C (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=6543)
- Perl implementation (https://metacpan.org/module/Algorithm::Munkres)
- Lisp implementation (https://web.archive.org/web/20070325192520/http://www.koders.com/lisp/fid7C3730AF4E356C65F93F20A6410814CBF5F40854.aspx?s=iso+3166)
- C++ (STL) implementation (multi-functional bipartite graph version) (http://students.cse.tamu.edu/lantao/codes/codes.php)
- C++ implementation (https://github.com/saebyn/munkres-cpp)
- C++ implementation of the $O(n^3)$ algorithm (http://dlib.net/optimization.html#max_cost_assignment) (BSD style open source licensed)

- Another C++ implementation with unit tests (http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=hungarianAlgorithm)
- Another Java implementation with JUnit tests (Apache 2.0) (http://timefinder.svn.sourceforge.net/viewvc/timefinder/trunk/timefinder-algo/src/main/java/de/timefinder/algo/roomassignment/)
- MATLAB implementation (https://web.archive.org/web/20120814231237/http://www.mathworks.com/matlabcentral/fileexchange/11609)
- C implementation (https://launchpad.net/lib-bipartite-match)
- Javascript implementation (http://twofourone.blogspot.com/2009/01/hungarian-algorithm-in-javascript.html)
- The clue R package proposes an implementation, solve_LSAP (http://cran.r-project.org/web/packages/clue/clue.pdf)

---