

Hopcroft–Karp algorithm

From Wikipedia, the free encyclopedia

In computer science, the **Hopcroft–Karp algorithm** is an algorithm that takes as input a bipartite graph and produces as output a maximum cardinality matching – a set of as many edges as possible with the property that no two edges share an endpoint. It runs in $O(|E|\sqrt{|V|})$ time in the worst case, where E is set of edges in the graph, and V is set of vertices of the graph. In the case of dense graphs the time bound becomes $O(|V|^{2.5})$, and for random graphs it runs in near-linear time.

The algorithm was found by John Hopcroft and Richard Karp (1973). As in previous methods for matching such as the Hungarian algorithm and the work of Edmonds (1965), the Hopcroft–Karp algorithm repeatedly increases the size of a partial matching by finding augmenting paths. However, instead of finding just a single augmenting path per iteration, the algorithm finds a maximal set of shortest augmenting paths. As a result only $O(\sqrt{n})$ iterations are needed. The same principle has also been used to develop more complicated algorithms for non-bipartite matching with the same asymptotic running time as the Hopcroft–Karp algorithm.

Contents

- 1 Augmenting paths
- 2 Algorithm
- 3 Analysis
- 4 Comparison with other bipartite matching algorithms
- 5 Non-bipartite graphs
- 6 Pseudocode
- 7 Notes
- 8 References

Augmenting paths

A vertex that is not the endpoint of an edge in some partial matching M is called a *free vertex*. The basic concept that the algorithm relies on is that of an *augmenting path*, a path that starts at a free vertex, ends at a free vertex, and alternates between unmatched and matched edges within the path. If M is a matching, and P is an augmenting path relative to M , then the symmetric difference of the two sets of edges, $M \oplus P$, would form a matching with size $|M| + 1$. Thus, by finding augmenting paths, an algorithm may increase the size of the matching.

Conversely, suppose that a matching M is not optimal, and let P be the symmetric difference $M \oplus M^*$ where M^* is an optimal matching. Then P must form a collection of disjoint augmenting paths and cycles or paths in which matched and unmatched edges are of equal number; the difference in size between M and M^* is the number of augmenting paths in P . Thus, if no augmenting path can be found, an algorithm may safely terminate, since in this case M must be optimal.

An augmenting path in a matching problem is closely related to the augmenting paths arising in maximum flow problems, paths along which one may increase the amount of flow between the terminals of the flow. It is possible to transform the bipartite matching problem into a maximum flow instance, such that the

alternating paths of the matching problem become augmenting paths of the flow problem.^[1] In fact, a generalization of the technique used in Hopcroft–Karp algorithm to arbitrary flow networks is known as Dinic's algorithm.

Input: Bipartite graph $G(U \cup V, E)$

Output: Matching $M \subseteq E$

$M \leftarrow \emptyset$

repeat

$\mathcal{P} \leftarrow \{P_1, P_2, \dots, P_k\}$ maximal set of vertex-disjoint shortest augmenting paths

$M \leftarrow M \oplus (P_1 \cup P_2 \cup \dots \cup P_k)$

until $\mathcal{P} = \emptyset$

Algorithm

Let U and V be the two sets in the bipartition of G , and let the matching from U to V at any time be represented as the set M .

The algorithm is run in phases. Each phase consists of the following steps.

- A breadth-first search partitions the vertices of the graph into layers. The free vertices in U are used as the starting vertices of this search, and form the first layer of the partition. At the first level of the search, only unmatched edges may be traversed (since the free vertices in U are by definition not adjacent to any matched edges); at subsequent levels of the search, the traversed edges are required to alternate between matched and unmatched. That is, when searching for successors from a vertex in U , only unmatched edges may be traversed, while from a vertex in V only matched edges may be traversed. The search terminates at the first layer k where one or more free vertices in V are reached.
- All free vertices in V at layer k are collected into a set F . That is, a vertex v is put into F if and only if it ends a shortest augmenting path.
- The algorithm finds a maximal set of *vertex disjoint* augmenting paths of length k . This set may be computed by depth first search from F to the free vertices in U , using the breadth first layering to guide the search: the depth first search is only allowed to follow edges that lead to an unused vertex in the previous layer, and paths in the depth first search tree must alternate between matched and unmatched edges. Once an augmenting path is found that involves one of the vertices in F , the depth first search is continued from the next starting vertex.
- Every one of the paths found in this way is used to enlarge M .

The algorithm terminates when no more augmenting paths are found in the breadth first search part of one of the phases.

Analysis

Each phase consists of a single breadth first search and a single depth first search. Thus, a single phase may be implemented in linear time. Therefore, the first $\sqrt{|V|}$ phases, in a graph with $|V|$ vertices and $|E|$ edges,

take time $O(|E|\sqrt{|V|})$.

It can be shown that each phase increases the length of the shortest augmenting path by at least one: the phase finds a maximal set of augmenting paths of the given length, so any remaining augmenting path must be longer. Therefore, once the initial $\sqrt{|V|}$ phases of the algorithm are complete, the shortest remaining augmenting path has at least $\sqrt{|V|}$ edges in it. However, the symmetric difference of the eventual optimal matching and of the partial matching M found by the initial phases forms a collection of vertex-disjoint augmenting paths and alternating cycles. If each of the paths in this collection has length at least $\sqrt{|V|}$, there can be at most $\sqrt{|V|}$ paths in the collection, and the size of the optimal matching can differ from the size of M by at most $\sqrt{|V|}$ edges. Since each phase of the algorithm increases the size of the matching by at least one, there can be at most $\sqrt{|V|}$ additional phases before the algorithm terminates.

Since the algorithm performs a total of at most $2\sqrt{|V|}$ phases, it takes a total time of $O(|E|\sqrt{|V|})$ in the worst case.

In many instances, however, the time taken by the algorithm may be even faster than this worst case analysis indicates. For instance, in the average case for sparse bipartite random graphs, Bast et al. (2006) (improving a previous result of Motwani 1994) showed that with high probability all non-optimal matchings have augmenting paths of logarithmic length. As a consequence, for these graphs, the Hopcroft–Karp algorithm takes $O(\log |V|)$ phases and $O(|E| \log |V|)$ total time.

Comparison with other bipartite matching algorithms

For sparse graphs, the Hopcroft–Karp algorithm continues to have the best known worst-case performance, but for dense graphs a more recent algorithm by Alt et al. (1991) achieves a slightly better time bound,

$O\left(n^{1.5} \sqrt{\frac{m}{\log n}}\right)$. Their algorithm is based on using a push-relabel maximum flow algorithm and then,

when the matching created by this algorithm becomes close to optimum, switching to the Hopcroft–Karp method.

Several authors have performed experimental comparisons of bipartite matching algorithms. Their results in general tend to show that the Hopcroft–Karp method is not as good in practice as it is in theory: it is outperformed both by simpler breadth-first and depth-first strategies for finding augmenting paths, and by push-relabel techniques.^[2]

Non-bipartite graphs

The same idea of finding a maximal set of shortest augmenting paths works also for finding maximum cardinality matchings in non-bipartite graphs, and for the same reasons the algorithms based on this idea take $O(\sqrt{|V|})$ phases. However, for non-bipartite graphs, the task of finding the augmenting paths within each phase is more difficult. Building on the work of several slower predecessors, Micali & Vazirani (1980) showed how to implement a phase in linear time, resulting in a non-bipartite matching algorithm with the same time bound as the Hopcroft–Karp algorithm for bipartite graphs. The Micali–Vazirani technique is complex, and its authors did not provide full proofs of their results; subsequently, a "clear exposition" was published by Peterson & Loui (1988) and alternative methods were described by other authors.^[3] In 2012, Vazirani offered a new simplified proof of the Micali–Vazirani algorithm.^[4]

Pseudocode

```
/*
G = G1 U G2 U {NIL}
where G1 and G2 are partition of graph and NIL is a special null vertex
*/

function BFS ()
  for v in G1
    if Pair_G1[v] == NIL
      Dist[v] = 0
      Enqueue(Q,v)
    else
      Dist[v] = ∞
  Dist[NIL] = ∞
  while Empty(Q) == false
    v = Dequeue(Q)
    if Dist[v] < Dist[NIL]
      for each u in Adj[v]
        if Dist[ Pair_G2[u] ] == ∞
          Dist[ Pair_G2[u] ] = Dist[v] + 1
          Enqueue(Q,Pair_G2[u])
  return Dist[NIL] != ∞

function DFS (v)
  if v != NIL
    for each u in Adj[v]
      if Dist[ Pair_G2[u] ] == Dist[v] + 1
        if DFS(Pair_G2[u]) == true
          Pair_G2[u] = v
          Pair_G1[v] = u
          return true
  Dist[v] = ∞
  return false
return true

function Hopcroft-Karp
  for each v in G
    Pair_G1[v] = NIL
    Pair_G2[v] = NIL
  matching = 0
  while BFS() == true
    for each v in G1
      if Pair_G1[v] == NIL
        if DFS(v) == true
          matching = matching + 1
  return matching
```

Notes

1. ^ Ahuja, Magnanti & Orlin (1993), section 12.3, bipartite cardinality matching problem, pp. 469–470.
2. ^ Chang & McCormick (1990); Darby-Dowman (1980); Setubal (1993); Setubal (1996).
3. ^ Gabow & Tarjan (1989) and Blum (2001).
4. ^ Vazirani (2012)

References

- Ahuja, Ravindra K.; Magnanti, Thomas L.; Orlin, James B. (1993), *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall.
- Alt, H.; Blum, N.; Mehlhorn, K.; Paul, M. (1991), "Computing a maximum cardinality matching in a bipartite graph in time $O\left(n^{1.5} \sqrt{\frac{m}{\log n}}\right)$ ", *Information Processing Letters* **37** (4): 237–240, doi:10.1016/0020-0190(91)90195-N (<https://dx.doi.org/10.1016%2F0020-0190%2891%2990195-N>).
- Bast, Holger; Mehlhorn, Kurt; Schafer, Guido; Tamaki, Hisao (2006), "Matching algorithms are fast in

sparse random graphs", *Theory of Computing Systems* **39** (1): 3–14, doi:10.1007/s00224-005-1254-y (<https://dx.doi.org/10.1007%2Fs00224-005-1254-y>).

- Blum, Norbert (2001), *A Simplified Realization of the Hopcroft-Karp Approach to Maximum Matching in General Graphs* (<http://theory.cs.uni-bonn.de/ftp/reports/cs-reports/2001/85232-CS.ps.gz>), Tech. Rep. 85232-CS, Computer Science Department, Univ. of Bonn.
- Chang, S. Frank; McCormick, S. Thomas (1990), *A faster implementation of a bipartite cardinality matching algorithm*, Tech. Rep. 90-MS-C-005, Faculty of Commerce and Business Administration, Univ. of British Columbia. As cited by Setubal (1996).
- Darby-Dowman, Kenneth (1980), *The exploitation of sparsity in large scale linear programming problems – Data structures and restructuring algorithms*, Ph.D. thesis, Brunel University. As cited by Setubal (1996).
- Edmonds, Jack (1965), "Paths, Trees and Flowers", *Canadian J. Math* **17**: 449–467, doi:10.4153/CJM-1965-045-4 (<https://dx.doi.org/10.4153%2FCJM-1965-045-4>), MR 0177907 (<https://www.ams.org/mathscinet-getitem?mr=0177907>).
- Gabow, Harold N.; Tarjan, Robert E. (1991), "Faster scaling algorithms for general graph matching problems", *Journal of the ACM* **38** (4): 815–853, doi:10.1145/115234.115366 (<https://dx.doi.org/10.1145%2F115234.115366>).
- Hopcroft, John E.; Karp, Richard M. (1973), "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM Journal on Computing* **2** (4): 225–231, doi:10.1137/0202019 (<https://dx.doi.org/10.1137%2F0202019>).
- Micali, S.; Vazirani, V. V. (1980), "An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs", *Proc. 21st IEEE Symp. Foundations of Computer Science*, pp. 17–27, doi:10.1109/SFCS.1980.12 (<https://dx.doi.org/10.1109%2FSFCS.1980.12>).
- Peterson, Paul A.; Loui, Michael C. (1988), "The general maximum matching algorithm of Micali and Vazirani", *Algorithmica* **3** (1-4): 511–533, doi:10.1007/BF01762129 (<https://dx.doi.org/10.1007%2FBF01762129>).
- Motwani, Rajeev (1994), "Average-case analysis of algorithms for matchings and related problems", *Journal of the ACM* **41** (6): 1329–1356, doi:10.1145/195613.195663 (<https://dx.doi.org/10.1145%2F195613.195663>).
- Setubal, João C. (1993), "New experimental results for bipartite matching", *Proc. Netflow93*, Dept. of Informatics, Univ. of Pisa, pp. 211–216. As cited by Setubal (1996).
- Setubal, João C. (1996), *Sequential and parallel experimental results with bipartite matching algorithms* (<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.3539>), Tech. Rep. IC-96-09, Inst. of Computing, Univ. of Campinas.
- Vazirani, Vijay (2012), *An Improved Definition of Blossoms and a Simpler Proof of the MV Matching Algorithm* (<http://arxiv.org/abs/1210.4594>), CoRR abs/1210.4594.

Retrieved from "http://en.wikipedia.org/w/index.php?title=Hopcroft–Karp_algorithm&oldid=593898016"

-
- This page was last modified on 4 February 2014, at 15:28.
 - Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.