Scientific Research Publishing

Journal of Applied Mathematics and Physics, 2018, 6, 1773-1782
http://www.scirp.org/journal/jamp
ISSN Online: 2327-4379
ISSN Print: 2327-4352

An $o(n^{2.5})$ Algorithm: For Maximum Matchings in General Graphs

Yingtai Xie

College of Information Science and Technology of Chengdu University, Chengdu, China
Email: Xyintai@189.cn

How to cite this paper: Xie, Y.T. (2018) An $o(n^{2.5})$ Algorithm: For Maximum Matchings in General Graphs. *Journal of Applied Mathematics and Physics*, **6**, 1773-1782.
https://doi.org/10.4236/jamp.2018.69152

Received: June 5, 2018
Accepted: August 31, 2018
Published: September 7, 2018

Abstract

This article extend the John E. Hopcroft and Richart M. Karp Algorithm (HK Algorithm) for maximum matchings in bipartite graphs to the non-bipartite case by providing a new approach to deal with the blossom in alternating paths in the process of searching for augmenting paths, which different from well-known "shrinking" way of Edmonds and makes the algorithm for maximum matchings in general graphs more simple.

Keywords

Matching, Augmenting Path, Blossom, Equivalent Digraph

1. Introduction

A matching in a graph is a set of edges, no two of which share a vertex. Given a graph, it is a well-known problem to find a matching of maximum carnality. All these algorithms for maximum matching, by berge's theorem in 1957 [1], are led to search for augmenting paths. In the case of bipartite graphs, it is easier to search for augmenting paths using of bread-first search. These algorithms require $o(mn)$ steps. But in the non-bipartite case, the problem becomes even more difficult so that until 1965 only exponential algorithm for finding a maximum matching in general graphs were known. The reason was that one did not know how to deal with odd cycles (Blossoms) in alternating paths in the process of searching for augmenting paths. Edmonds [4] defined the key notion of blossoms and finessed this difficulty in non-bipartite graphs by "shrinking" blossom. The straightforward implementation of his approach led to an $o(n^4)$ algorithm for maximum matching in general graphs. After further improvement there are $o(n^3)$ or $o(nm)$ algorithms [2] [3] for it.

In 1973 Hoperoft and Karp [5] proved the following fact. If one computes in

one phase a maximal set of shortest augmenting paths, then $o(\sqrt{n})$ such phases would be sufficient. For the bipartite case they showed that a phase can be implemented by a breath-first search followed by a depth-first search. This led to an $o(n+m)$ implementation of one phase and hence to an $o(\sqrt{nm})$ algorithm for maximum matching in bipartite graphs. The HK Algorithm for maximum matching in bipartite graphs [1] is still the most efficient known algorithm for the problem.

The success of Hopcroft and Karp algorithm in bipartite graphs has made many people attempt to extend it to non-bipartite case. In fact ,in their paper almost all the results are derived for general graphs. The specialization to the bipartite case occurs in computing in one phase a maximal set of shortest augmenting paths only.

In 1980 Mieali and Vijay Vazirani [6] stated a matching algorithm in pseudocode, claimed to have an $o(m)$ implementation of a phase in general graphs. Although their result led to a most efficient general graph maximum matching algorithm running time of $o(\sqrt{nm})$ and is cited in many papers and also in some textbooks but no proof of correctness was available. Also the paper of Peterson and Loui [7] does not clarify the situation.

In this article a new approach to deal with the blossom is proved, which makes a phase can be implemented by a breath-first search followed by a depth-first search.

After summarizing basic theory of matched problem and introducing the HK algorithm in bipartite graph in Section 1, we discuss the obstacle to extend the HK algorithm to the non-bipartite case, that is blossom, in Section 2. The theoretical basis of the new approach is established in Section 3 and then we construct the new algorithm, which extend HK algorithm in bipartite graph to non-bipartite case in Section 4.

2. The Basic Theory and Contribution of Hopcroft and Karp

Let $G=\{V,E\}$ be a finite undirected graph (without loops or multiple edges) having the vertex set $V(G)$ and the edge set $E(G)$. An edge $e\in E(G)$ incident with vertices u and v is written $e=uv$. A path in G is a sequence of vertices and edges (without repeated vertices)

$$P=(u_0,u_1,u_2,\ldots,u_r,u_{r+1})$$

In which $e_i=u_iu_{i+1}\in E(G)$ (i = 0, 1, ···, r). Two vertices u_0 and u_{r+1} are called end-poinds of the path. Especially, when $u_0=u_{r+1}$, *i.e.* two end-poinds overlap then

$$c=(u_0,u_1,\ldots,u_r,u_0)$$

Is a circuit.

Let $u\in V(G)$

$$N(u)=\{v:uv\in E(G)\}$$

is called ***neighborhood*** of u.

A set $M \subseteq E$ is a matching if no vertex is incident with more than one edge in *M*. A matching of maximum cardinality is called a maximum matching. We say the maximum matching is ***complete***, or ***perfect*** when $|v| = 2m$ and $|M| = m$. We make the following definitions relative to a matching *M*. Edges in *M* are called ***matched*** edges; the others are ***free***. A vertex *v* is ***free*** if it is incident with no edge in *M* otherwise it is ***matched***.

A path

$$P = (v_0, v_1, v_2, ..., v_{2r}, v_{2r+1})$$

is called ***M-alternating*** path if its edges are alternately in *E-M* and in *M*; *i.e.*

$$P \cap M = \{v_1v_2, v_3v_4, v_5v_6, ..., v_{2r-1}v_{2r}\} \quad (1.1)$$

$$P - (P \cap M) = \{v_0v_1, v_2v_3, ..., v_{2r}v_{2r+1}\} \quad (1.2)$$

A path with odd length $2r + 1$ (without repeated vertices) is called an M-*augmenting* path if its end vertices v_0 and v_{2r+1} are both free.

It is easy to result in the lemma by Equation (1.1) and Equation (1.2):

Lemma 1.1. $M' = P \oplus M = (P \cup M) - (P \cap M)$ is a new matching and $|M'| = |M| + 1$.

And the following theorem is also well known.

Theorem 1.1. (Berge) *M* is not a maximum matching if and only if there is an augmenting path relative to *M*.

According to above theorems, the algorithm for finding the maximum matching in a graph boils down repeatedly to search for a augmenting path *P* relative to current matching *M* and augment current matching to $M \oplus P$.

The contribution of Hopcroft and Karp is to compute in one phase a maximal set of shortest vertex-disjoint augmenting paths with a same length $P_1, P_2, ..., P_k$ and augment current matching to $M \oplus P_1 \oplus P_2 \oplus ... \oplus P_k$ and proved that the number of phases at most $2[\sqrt{n}] + 2$.

Algorithm A: Maximum Matching Algorithm (Hopcroft and Karp).

1) $M \leftarrow \varphi$

2) Let $\ell(M)$ be the length of a shortest *M*-augmenting path. Find a maximum set of paths $\{P_1, P_2, ..., P_t\}$ with the properties that:

a) For each *i*, P_i is an *M*-augmenting path and $|P_i| = \ell(M)$;

b) The $P_i (i = 1, 2, ..., t)$ are vertex-disjoint.

3) $M \leftarrow M \oplus P_1 \oplus P_2 \oplus ... \oplus P_t$; go to step 1.

Theorem 1.2. (Hopcroft and Karp). [5] If the cardinality of a maximum matching is *S* then Algorithm A constructs a maximum matching within $2[\sqrt{s}] + 2$ executions of step 1.

This way of describing the construction of a maximum matching suggests that we should concentrate on the efficient implementation of an entire phase (*i.e.*, the execution of Step 1 in Algorithm A).

For the bipartite case，Hopcroft and Karp showed that a phase can be implemented by a breath-first search followed by a depth-first search，which has time complexity $o(m)$. Hence, by theorem 1.2, the algorithm A has time complexity $o(\sqrt{n}m)$.

3. Nonbipartite Matching: Blossoms

Unfortunately, in non-bipartite graphs, the lack of bipartite structure makes one might be fail of searching augmenting path by breath-first search in the execution of Step 1 in Algorithm A and this situation makes the task of implementing a phase far more difficult.

The problem remains how to deal with blossoms in computing in one phase a maximal set of shortest augmenting paths in non-bipartite case.

Definition 2.1. A odd circuit with $2k+1$ vertices having k matched edges is called a blossom.

In **Figure 1** (the matched edge are drawn with heavy line, the free vretices are drawn with small cycle). The odd circuits,

$$c_1 = (v_4, v_5, v_6, v_4)$$

$$c_2 = (v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_2)$$

$$c_3 = (v_{11}, v_1, v_2, v_8, v_7, v_6, v_5, v_4, v_3, v_9, v_{10}, v_{11})$$

are blossoms. The only vertex incident with no matched edges in the blossom is called basis of the blossom, as v_4, v_2 and v_{11}.

The Hopkroft and Karp's Algorithm [5] of implementation of step 1 of Algorithm A might fail when using it in the non-bipartite graph if which contains a blossom, because one alternating path at the basis of a blossom traverses this blossom and starts "folding" on itself, with a disastrous consequence .

Edmond's method of dealing with blossoms are also not adequate for the task of finding shortest augmenting paths since by "shrinking" them, length information is completely lost.

We attempt to find a way to through the barrier of blossom keeping length information in finding augmenting paths and then extend the John E. Hopcroft and Richart M. Karp Algorithm (HK Algorithm) for maximum matchings in bipartite graphs to general graphs.

4. Equivalent Digraph

A digraph Γ is defined by a set $V(\Gamma)$ of elements called vertices, a set $W(\Gamma)$ of elements called directed edges or ***darts***, and two relations of incidence. These associate each dart d, the one with a vertex $h(d)$ called its head and the other with a vertex $t(d)$ called its tail. A dart $d \in W(\Gamma)$ incident with $h(d) = v$, $t(d) = u$ is written $d = \overrightarrow{uv}$ and $d^- = \overrightarrow{vu}$ is called inverse of d.

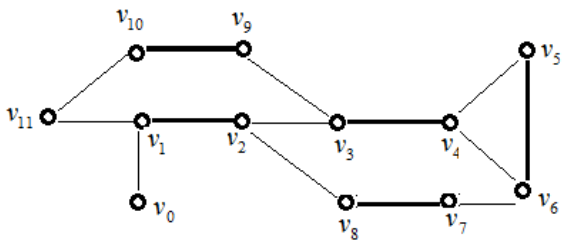

Figure 1. M-matched graph.

For a vertex, the number of darts sharing the vertex as head is called the ***in-degree*** of the vertex and the number of darts sharing it as tail is called its ***out-degree***.

The in degree of v is denoted by $\deg^-(v)$ and its out degree is denoted by $\deg^+(v)$.

A path in digraph Γ is a sequence of $r \geq 1$ darts,

$$P = (d_0, d_1, ..., d_{r-1}) \tag{3.1}$$

Or

$$P = \overrightarrow{v_0 v_1 ... v_r} \tag{3.2}$$

where $d_j = \overrightarrow{v_j v_{j+1}}$ of Γ, not necessarily all distinct. It requires that the $t(d_{k+1}) = h(d_k)$, whenever $0 \leq k \leq r-1$. The first and last vertices of path P, that is $t(d_0)$ and $h(d_{r-1})$, are called the tail $t(P)$ and head $h(P)$ of path P, respectively. The number r is called the length $|P|$ of path P.

$$p^- = \overrightarrow{v_r v_{r-1} v_{r-2} ... v_2 v_1 v_0} = (d_{r-1}^-, d_{r-2}^-, ..., d_2^-, d_1^-, d_0^-)$$

Is called ***inverse*** of path P, where $d_j^- = \overrightarrow{v_{j+1} v_j}$.

Consider two paths P and Q in Γ, if $h(P) = t(Q)$, there is a path PQ, called the product of P and Q, formed by writing down the terms of P, in their order in P, and continuing with the terms of Q, in their order in Q. It is easy to verify that $t(PQ) = t(P), h(PQ) = h(Q)$ and $PQR = (PQ)R = P(RQ)$, $(PQR)^- = R^- Q^- P^-$, $|PQ| = |P| + |Q|$.

One dart is a path with length 1. One path $P = (d_0, d_1, ..., d_{r-1})$ is the product of r darts $d_0, d_1, ..., d_{r-1}$, *i.e.* $P = d_0 d_1 ... d_{r-1}$.

We say that P is ***dart-simple*** if no dart repeated in it.

For a given graph G, we associate with each edge $e = uv$ of G two distinct elements $d = \vec{uv}$ and $d^- = \vec{vu}$ called the opposite darts on e. We form an ***equivalent digraph*** $\vec{G}$ of G from the given graph G.

$V(\vec{G}) = V(G)$ and $W(\vec{G}) = \{d, d^- : e \in E(G),\ d, d^-$ are opposits darts on $e\}$

A path in $\vec{G}$ is ***edge-simple*** if no two of its terms are darts on the same edge of G. Thus an edge-simple is necessarily dart-simple.

Let M be a matching in G. Edges in M and the darts on them are called matched. The other edges and darts on them are free. A path in $\vec{G}$ is M-***alternating*** if its darts are alternately free and matched and the first dart is necessarily free.

Let $P = d_0 d_1 ... d_i ... d_r$ be M-alternating then d_i is free when i is even and d_i is matched when i is odd.

Lemma 3.1. Let

$$P = d_0 d_1 ... d_r (d_i = \overrightarrow{v_i v_{i+1}}; 0 \leq i \leq r)$$

be a M-alternating path in equivalent digraph $\vec{G}$ of G from a free vertex $v_0 = t(d_0)$ to another free vertex $v_{r+1} = h(D_r)$. If P is edge-simple in $\vec{G}$ then the alternating path $P' = (v_0, v_1, ..., v_{r+1})$ is a M-augmenting path in G.

Proof. Each vertex in P is the head or tail of a matched dart, except v_0, v_{r+1}, but $v_0 \neq v_{r+1}$, so if $0 < i < t < r+1; v_i = v_t$ then the matched darts, which incident with v_i, v_t, are on the same edge of G, because no two of matched edges share an vertex. This would conflict with that P is edge-simple. Then the vertices $v_0, v_1, ..., v_r, v_{r+1}$ in P are all distinct. P' is a M-augmenting in G. We call a edge-simple path P a M-augmenting path, for simple.

Definition 3.1. Assume i is odd, t. is even and $i < t$, the M-alternating path $P = d_0 d_1 ... d_{i-1} d_i ... d_{t-1} d_t ... d_r$ meet the condition that $d_t = d_{i-1}^-$. We call $P_{b(i,t)} = d_i d_{i+1} ... d_{t-1}$ a ***blossom*** and d_{i-1} ***in-dart***, d_t ***out-dart*** of the blossom.

It is easy to see:

1) d_{i-1}, d_t are matched therefore d_i, d_{t-1} are free, because $i-1, t$ are even.

2) $P_{b^-(i,t)}^- = d_{t-1}^- d_{t-2}^- ... d_i^-$ is also a blossom in $P^- = d_r^- d_{r-1}^- ... d_t^- P_{b^-(t,i)}^- d_i^- d_{i-1}^- ... d_0^-$ and d_t^-, d_{i-1}^- are in-dart and out-dart of it respectively.

The M-alternating path P from a free vertex v to a free vertex u is called ***shortest*** if P is of least carnality among these M-alternating paths.

Lemma 3.2. Let $P = d_0 d_1 ... d_r (d_i = \overrightarrow{v_i v_{i+1}}, i = 0, 1, ..., r)$ be a shortest M-alternating path in $\vec{G}$ from a free vertex $v_0 = t(d_0)$ to an another free vertex $v_{r+1} = h(d_r)$ ($v_{r+1} \neq v_0$), then either P is a M-augmenting path or P pass though a blossom.

Proof. If P be not M-augmenting then there is $i < t$, $v_i = v_t$. Let t be as small as possible, so that $v_{i-1} \neq v_{t-1}$ then $d_{i-1} \neq d_{t-1}$, but d_{i-1}, d_{t-1} share vertex $v_i = v_t$ as their head, so d_{t-1} is necessarily free, because $d_{i-1} d_i$ is alternating. If $d_i \in M$ then $d_t = d_i$. We can obtain a shorter alternating path from v_0 to v_{r+1} by deleting $d_i, d_{i+1}, ..., d_{t-1}$ from P. This is contradict with that P is shortest.

Can only be the case that $d_t = d_{i-1}^-$, because they share v_t as their tail. The $d_i d_{i+1} ... d_{t-1}$ is a blossom and d_{i-1} is ***in-dart***, d_t is ***out-dart*** of it. □

5. The Algorithm to Implement Step 1 of Algorithm A in General Graph

We construct a directed graph

$$\hat{G} = (\hat{V}, \hat{D}) \tag{4.1}$$

from the equivalent digraph $\vec{G}$ of G, which exhibits conveniently all shortest M-augmenting paths in $\vec{G}$.

Let $\vec{G}$ be the equivalent digraph of G. D is a set of darts. Define:

$$h(D) = \{h(d) : d \in D\}$$

$$t(D) = \{t(d) : d \in D\}$$

$$t(D_0) = \{v : v \in V(G) \,\&\, v \text{ is } M - free\}$$

For $i = 0, 1, 2, ...$

$$D_{2i} = \{\overrightarrow{uv} : (u \in t(D_0)(i=0)\ ;\ u \in h(D_{2i-1})(i>0)) \,\&\, uv \in E(G) - M\}$$

$$D_{2i+1} = \{\overrightarrow{uv} : u \in h(D_{2i}), uv \in M\}$$

The construction continues until for the first time a set D_{2I^*} is constructed so that there is a $d \in D_{2I^*}$ and $h(d)$ is M-free. Then

$$\widehat{V} = t(D_0) \cup \bigcup_{j=0}^{2I^*} h(D_j) \qquad \widehat{D} = \bigcup_{j=0}^{2I^*} D_j$$

For example, the $\widehat{G}$ shown by **Figure 2** (the direction of all darts as shown by the big arrow) is constructed from the equivalent digraph $\vec{G}$ of the graph G shown by **Figure 1**. **Figure 2** exhibit all of the shortest M-alternating paths from free vertex to free vertex in **Figure 1**.

The following properties of $\widehat{G}$ is easy to see:

1) Each path from a free vertex $u \in D_0$ to a free vertex $v \in D_{2I^*}$ in $\widehat{G}$ is a shortest M-alternating path with length $2I^*+1$.We call it ***regular M-alternating path*** in $\widehat{G}$

2) If a M-alternating path $P \in \widehat{G}$, $u = t(P) \in D_0$ and $v = h(P) \in D_{2I^*}$ then $P^- \in \widehat{G}$ and $t(P^-) = h(P) = v, h(P^-) = t(P) = u$. P^- is also a regular M-alternating path in $\widehat{G}$.

3) Let $\mathbb{R}$ be the set all regular M-alternating paths in $\widehat{G}$ then $t(\mathbb{R}) \subseteq t(D_0)$, $h(\mathbb{R}) \subseteq h(D_{2I^*})$ and $t(\mathbb{R}) = h(\mathbb{R})$, by (b).

4) If $|h(\mathbb{R})| = 1$ then there is no augmenting path in $\widehat{G}$ because there is only one M-alternating path P in $\widehat{G}$ and $t(P) = h(P)$. In this case we remove the vertex $h(P)$ from $h(D_{2I^*})$, and continue to construct.

Let $|h(\mathbb{R})| > 1$ Then it's a matter of finding out the shortest M-augmenting paths in $\widehat{G}$, *i.e.* to implement the step 1 in Algorithm A.

Let $d \in D_{2I^*}$ and $\mathbf{v} = h(d) \in h(D_{2I^*})$ is M-free. We construct a directed graph

$$J(L_0, \mathbf{v}) \tag{4.2}$$

which is set of all shortest M-alternating paths with length $2I^* + 1$ from a set $L_0 \subseteq t(D_0)$ of M-free vertices to the free vertex $\mathbf{v} \in h(D_{2I^*})$

$$D_{2I^*}^f = \{d : d \in D_{2I^*} \ \& \ h(d) = \mathbf{v}\}$$

$$D_j^f = \{d \in D_j \ \& \ h(d) \in t(D_{j+1}^f)\} (j = 2I^* - 1, \ldots, 2, 1, 0)$$

$$J(L_0, v) = (\vec{V}, \vec{D})$$

where

$$L_0 = D_0^f - \{\mathbf{v}\}, \ \vec{V} = L_0 \cup \bigcup_{j=0}^{2I^*} D_j^f, \ \vec{D} = \bigcup_{j=0}^{2I^*} D_j^f$$

Case 1: $L_0 = \varphi$. In this case there is no augmenting path with **V** as a end-vertex.

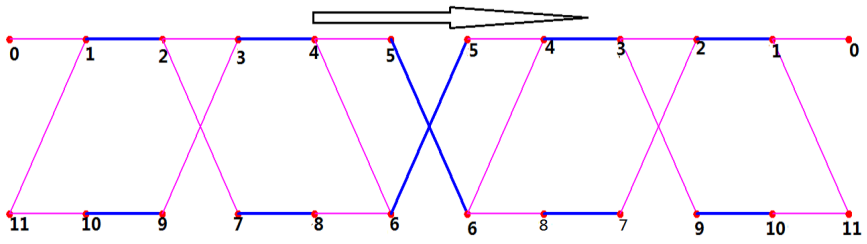

Figure 2. Directed graph including shortest M-alternating paths.

Case 2: $L_0 \neq \varphi$. In this case there is at least one M-alternating path from a free vertex u (other than **V**) to **V** in $J(L_0, \mathrm{v})$.

For example, **Figure 3** is $J(11,0)$ constructed from **Figure 3**, in which,. $L_0 = \{11\}$, D_5^f include two darts $\overrightarrow{(5,6)}$ and $\overrightarrow{(6,5)}$. D_9^f include only one dart $\overrightarrow{(2,1)}$ called ***Cut-dart***. General, if $d_i \in D_i^f$ is a only one of D_i^f we call d_i a ***cut-dart*** in $J(L_0, \mathrm{v})$. If d_i is a cut-dart then all of the paths in $J(L_0, \mathrm{v})$ pass d_i.

Claim 4.1. Each path $P \in J(L_0, \mathrm{v})$, either P is a M-augmenting path or P pass through a blossom, by lemma 3.2.

Claim 4.2. If d_i is a cut-dart and there is a M-augmenting path in $J(L_0, \mathrm{v})$ then d_i^- is not a cut-dart.

Claim 4.3 Each path $P \in J(L_0, \mathrm{v})$ is a M-augmenting path if there is no blossom in $J(L_0, \mathrm{v})$.

Claim 4.4. Let P be a M-alternating path in $J(L_0, \mathrm{v})$ and $\mathrm{u} = t(P) \in t(D_0)$, $\mathrm{v} = h(P) \in (D_{2I^*})$ then P^- is in $\widehat{G}$ and $\mathrm{v} = t(P^-) \in t(D_0)$;
$\mathrm{u} = h(P^-) \in (D_{2I^*})$ therefore $P^- \in J(L_0^{-\mathrm{u}}, \mathrm{u})$, where $\mathrm{u} \notin L_0^{-\mathrm{u}} \subset t(D_0)$.

Claim 4.5. If $P \in J(L_0, \mathrm{v})$ pass through a blossom and dart d is the in-dart (out-dart) of the blossom then $P^- \in J(L_0^{-\mathrm{u}}, \mathrm{u})$ is also pass through a blossom and d^- is the in-dart (out-dart) of it.

A blossom will be broken if to remove the in-dart or out-dart from a path $P \in J(L_0, \mathrm{v})$, which traverse the blossom .

Following theorem is the basis of the Algorithm to implement step1 of algorithm A.

Theorem 4.0 Let P be a augmenting path in $J(L_0, \mathrm{v})$ and

$$\mathrm{u} = t(P) \in t(D_0), \quad \mathrm{v} = h(P) \in (D_{2I^*})$$

d is a in-dart(out-dart) of a blossom in $J(L_0, \mathrm{v})$ and d isn't a cut-dart then $P \in J(L_0, \mathrm{v})$ or $P^- \in J(L_0^{-\mathrm{u}}, \mathrm{u})$ can't be broken after removing d.

Proof: If d is not on P then it can't be broken after removing d. If pass through d is on P then d^- is on $P^- \in J(L_0^{-\mathrm{u}}, \mathrm{u})$ and d is not on P^-. Therefore d isn't a cut-dart and it is a in-dart(out-dart) of a blossom in $P^- \in J(L_0^{-\mathrm{u}}, \mathrm{u})$.

Algorithm A1. The algorithm to implement step 1 of Algorithm A

1) to construct $\widehat{G}$ by (4.1). Let $D_{2I^*}^o = \{d : d \in D_{2I^*} \ \& \ h(d) \text{ is M-free}\}$

2) If $\left|h(D_{2I^*}^o)\right| = 0$ then end

3) For a $\mathrm{v} \in h(D_{2I^*}^o)$ to construct $J(L_0, \mathrm{v})$ by (4.2)

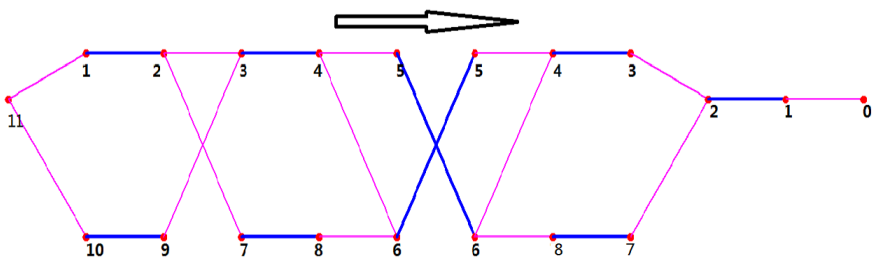

Figure 3. In-dart (1,2) being not cut-dart.

4) If $J(L_0, \mathrm{v}) = \varphi$ then remove v from $h(D^o_{2I^*})$ go to 1.

5) Get a M-alternating path P in $J(L_0, \mathrm{v})$.

6) If P is edge-simple then get a M-augmenting path and remove P and P^- from $\widehat{G}$, go to 1.

7) P Traverse a blossom and d is in-dart of it: if d isn't cut-dart then remove d else remove d^- from $J(L_0, \mathrm{v})$, go to 3.

To remove such vertices or darts (as in algorithm A1) may render certain other vertices useless, in the sense that they cannot be in any M-alternating paths. These vertices and darts must also be deleted; for, these dead-ends will make searching for an augmenting path more complicate. Hence we need an algorithm which deletes these dead-ends.

This is achieved as follows:

B: Deletion algorithm

For $i = 2I^*$ to 0 step −1

For each $v \in h(D_i)$ if $\deg^-(v) = 0$ then remove v from D_i

For $j = 0$ to $2I^*$

For each $v \in t(D_j)$ if $\deg^+(v) = 0$ then remove v from D_j in $J(11,0)$ (**Figure 3**).

For example, to get a M-alternating path $P = \overrightarrow{(11,1,2,3,4,5,6,4,3,2,1,0)}$ in $J(11,0)$ (**Figure 3**). Dart $\overrightarrow{(1,2)}$ on P is a in-dart and isn't a cut-dart To remove $\overrightarrow{(1,2)}$ from $J(11,0)$

The digraph after removing $\overrightarrow{(1,2)}$ from $J(11,0)$ is shown by **Figure 4**, to get a M-alternating path $P = \overrightarrow{(11,10,9,3,4,5,6,4,3,2,1,0)}$ in it, $d = \overrightarrow{(3,4)}$ is a in-dart and is a cut-dart. We remove out-dart $d^{-1} = \overrightarrow{(4,3)}$ and get a M-augmenting path $P = (11,10,9,3,4,5,6,8,7,2,1,0)$ in the result digraph.

The implementation of a phase consists of the construction of a digraph $\widehat{G}$, followed by an iteration which alternately finds an augmenting path and executes the deletion algorithm. The iteration stops when $D^o_{2I^*} = \phi$. It is easy to see that each dart of $\widehat{G}$ is examined once when $\widehat{G}$ is first constructed, and once more if the $h(d)$ or $t(d)$ of that dart d is deleted.

The execution time of a phase is $o(2m+n)$, where m is the number of edges in G, and n is the number of vertices. Hence the execution time of the entire algorithm is $o((2m+n)s)$, where s is the carnality of a maximum matching.

If G has n vertices then $m \le \frac{n^2}{4}$ and $s \le \sqrt{n}$, so that the execution time is bounded by $o(n^{5/2})$.

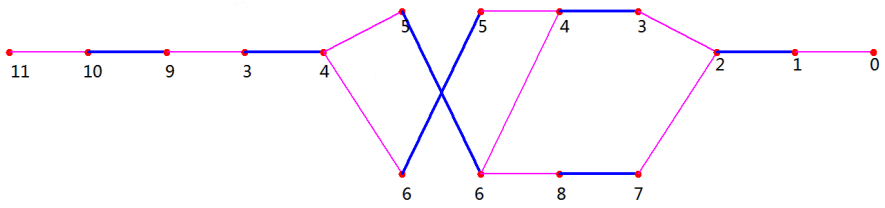

Figure 4. Out-dart being not cut-dart.

6. Conclusion

In this paper an algorithm running time of $o(m)$ to implement step 1 of Algorithm A in general graph has been proved. Different from the way of "shrinking" blossoms, which need to store the shrinked blossoms, the removed darts don't need to be stored when run this algorithm, because they can never be on any augmenting paths. The space complexity of this algorithm is also $o(m)$. The proved algorithm is very easy to program. The Mieali and Vijay Vazirani [6] algorithm, by contrast, is quite elaborate and its pseudo code is extensive.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Berge, C. (1957) Two Theorems in Graph Theory. *Proceedings of the National Academy of Sciences of the USA*, **43**, 449-844. https://doi.org/10.1073/pnas.43.9.842

[2] Balinski, M.L. (1969) Labelling to Obtain a Maximum Matching, in Combinatorial Mathematics and Its Applications. In: Bose, R.C. and Dowling, T.A., Eds., University of North Carolina Press, Chapel Hill, 585-602.

[3] Witzgall, C. and Zahn Jr., C.T. (1965) Modification of Edmond's Maximum Matching Algorithm. *Journal of Research of the National Bureau of Standards*, **69B**, 91-98.

[4] Edmonds, J. (1965) Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, **17**, 449-467. https://doi.org/10.4153/CJM-1965-045-4

[5] Hopcroft, J. and Karp, R.M. (1973) An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM Journal on Computing*, 225-231. https://doi.org/10.1137/0202019

[6] Micali, S. and Vazirani, V.V. (1980) An $O\left(\sqrt{|V|}.|E|\right)$ Algorithm for Finding Maximum Matching in General Graphs. *IEEE Annual Symposium on Foundations of Computer Science*.

[7] Peterson, P.A. and Lout, M.C. (1988) The General Matching Algorithm of Micali and Vazirani. *Algorithmica*, 511-533. https://doi.org/10.1007/BF01762129