A DFS ALGORITHM FOR MAXIMUM MATCHINGS IN GENERAL GRAPHS

Tony T. Lee[1], Bojun Lu[2] and Hanli Chu[1]

[1]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen
[2]School of Data Science, The Chinese University of Hong Kong, Shenzhen

ABSTRACT

In this paper, we propose a depth-first search (DFS) algorithm for finding maximum matching in general graphs. Unlike blossom shrinking algorithms, which store all possible alternative alternating-paths in the super-vertices shrinking from blossoms, the newly proposed algorithm does not involve blossom shrinking. The basic idea is to deflect the alternating path when facing blossoms. The algorithm maintains detour information in an auxiliary stack to minimize the redundant data structures. A benefit of our technique is to avoid spending the time on shrinking and expanding blossoms. This DFS algorithm can determine a maximum matching of a general graph with m edges and n vertices in $O(mn)$ time with space complexity $O(n)$.

KEYWORDS

Maximum Matching, Augmenting Path, Blossom, Trunk, Sprout

1. INTRODUCTION

The maximum matching in an undirected graph is a set of disjoint edges that has the maximum cardinality. Finding a maximum matching is a fundamental problem in combinatorial optimization [1, 2]. It has wide applications in the development of graph theory and computer science [3]. In a bipartite graph with n vertices and m edges, finding a maximum matching problem is solved by the Hopcroft-Karp algorithm in $O((m+n)\sqrt{n})$time [4]. For constructing a maximum matching in a general graph, the blossom shrinking algorithm proposed by Edmonds in [5] is the first polynomial algorithm that runs in time $O(n^4)$. The complexity of this algorithm has been improved from $O(n^4)$ to $O(n^3)$by Gabow [6], and further reduced to $O(mn)$ by Gabow and Tarjan [7] for a graph with nvertices and m edges. The best-known algorithm is given by Micali and Vazirani [8] and [9] that runs in $O\left(m\sqrt{n}\right)$, but it is rather difficult to understand and too complex for efficient implementation. Almost all these algorithms follow Edmonds' idea of shrinking blossoms [10], which requires data structures to represent blossoms, and the time spent on shrinking and expanding blossoms.

The main contribution of this paper is to present a depth-first search (DFS) maximum matching algorithm that does not involve blossom shrinking. The basic idea is to deflect the alternating path when a blossom, or odd cycle, is formed. This deflection algorithm adopts two stacks, one is a directional alternating path, and the other one is an ordered list of edges to maintain detour information. The two stacks interact with each other to grow or to prune in the exploring process, until an augmenting path is identified or it confirms that no augmenting paths exist. Unlike the Edmonds' algorithm, which is a breadth-first search (BFS) algorithm that stores all possible

David C. Wyld et al. (Eds): NIAI, MoWiN, AIAP, SIGML, CNSA, ICCIoT, CCSEIT - 2023
pp. 115-129, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130310

alternative alternating paths in the super-vertices shrunk from blossoms. The deflection algorithm maintains such detour information in the sprout stack to minimize the redundant data structures. This newly proposed maximum matching algorithm can achieve a complexity of $O(mn)$, because it avoids spending the time on shrinking and expanding blossoms.

The organization of this paper is as follows. In Sect. 2, we present the definitions of terminologies used in this paper. In Sect. 3, we describe the issue of parity conflicts arising from blossoms and illustrate our deflection method with some examples. This section is mainly expository in nature, and it compares the method of deflection versus shirking when blossoms occur. In Sect. 4, we present a DFS algorithm to enumerate augmenting paths, and discuss the performance of this algorithm. Finally, we conclude this paper in Sect. 5.

2. PRELIMINARIES

Definition. An undirected graph $G(V, E)$ consists of a **vertex set** V and an **edge set** E. An edge is an unordered pair of vertices $\{v, u\}$and written as $e = \langle v, u\rangle$. The number of vertices$n = |V(G)|$ is the **order** of G, and the number of edges $m = |E(G)|$ is the **size** of G.

Without loss of generality, we assume that the general graph G under consideration is a simple graph without loops, multiple edges, or isolated vertices.

Definition. A set $M \subseteq E$ is a **matching** if no two edges in M have a vertex in common, or no vertex $v \in V$ is incident with more than one edge in M. A matching of maximum cardinality is called a **maximum matching**. A **perfect matching** of a graph$G$ is a matching which covers all vertices of V. Relative to a matching M in G, a vertex v is called a **matched vertex**, or **covered vertex**, if it is incident to an edge in M. Otherwise, the vertex v is called a **free vertex**. The set of M-matched vertices is denoted by $\partial(M)$, and the set of M-free vertices $\bar{\partial}(M)$. Similarly, edges in M are **matched edges**, while edges not in $M$are **free edges**. Every matched vertex v has a **mate**, the other endpoint of the matched edge.

Definition. A**path** $P = \{\langle v_1, v_2\rangle, \langle v_2, v_3\rangle, \cdots, \langle v_{l-1}, v_l\rangle\}$ is a sequence of edges, which alternately join a sequence of distinct vertices. The path P can also sometimes be written as $P = v_1, v_2, \cdots, v_{l-1}, v_l$. A **cycle** is a path with an edge joining the first and last vertices. Relative to a matching M in G, an ***M*-alternating path***P*is a path in which edges alternate between those in M and those not in$M$, and it is called an ***M*-augmenting path** if its endpoints v_1 and v_l are both free, in which case l must be even.

The following result shows that an M-augmenting path P can enlarge the size of M by one.

Lemma 1. IfP is an M-augmenting path relative to a matching M, then the symmetric difference defined by

$$M \oplus P = (M - P) \cup (P - M) = (M \cup P) - (M \cap P)$$

is also a matching, and $|M \oplus P| = |M| + 1$.

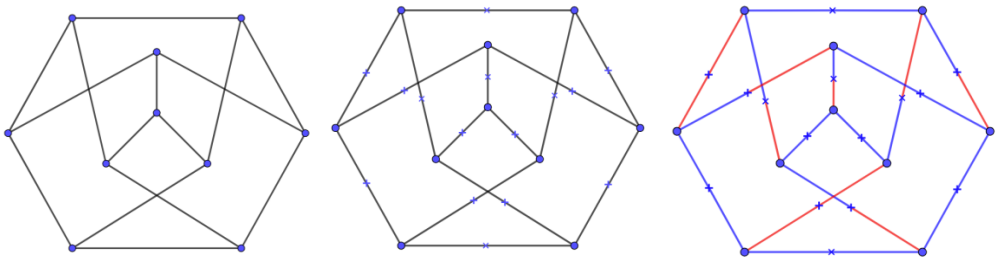
An immediate consequence of this result is the following theorem due to Berge [11] that characterizes maximum matchings.

Theorem 1. (Augmenting Path Theorem) A matching$M$ is maximum if and only if there is no M-augmenting path.

Theorem 1 implies that if a matching M in a graph G is not maximum, then there exists an M-augmenting path. This is the basis of almost all algorithms for determining maximum matchings in general graphs. The basic idea is to enlarge an existing matching M by any M-augmenting path. Repeat the searching process until no augmenting paths exist anymore.

Suppose that M is a matching in a graph $G(V, E)$, if we assign the red color to the edges in M and the blue color to those edges not in M, then there is a one-to-one correspondence between the complex-colored graph and the matching M. Follow almost all algorithms for finding maximum matching starting with some existing matching, we will adopt the complex coloring method proposed in [12] and [13] to initialize our maximum matching algorithm.

The complex coloring is a variable elimination method. In a graph $G(V, E)$, suppose that a fictitious vertex is inserted in the middle of an edge $\langle v_i, v_j \rangle$ to divide the edge into two links. These two links connect the fictitious vertex to the two endpoints v_i orv_j, respectively. As an example, the graph displayed in Figure 1(a) with inserted fictitious vertices is shown in Figure 1(b). Instead of coloring the edges, the complex coloring is assigning colors to links.



1(a). Petersen Graph. 1(b). Petersen Graph with fictitious vertices. 1(c). A color configuration of Petersen Graph.

Figure 1. Complex coloring of Petersen Graph.

Definition. Assigning the two colors $\{r := red, b := blue\}$ to the two links of each edge, the coloring is **consistent** if one and only one of the links incident to each vertex is assigned red color r, and all other links are colored in blue color b. If the two links of an edge are colored with different colors r and b, then the edge is called a (r, b)**variable**, otherwise it is a **constant**, as Figure 1 shows. The notation $\langle v_i, v_j \rangle \rightarrow (color_1, color_2)$is used to indicate the assigning of color pair $(color_1, color_2)$to edge $\langle v_i, v_j \rangle$. The **color configuration** of a complex-colored graph G is represented by the two-tuple $C(M) = \{M, \bar{\partial}(M)\}$, where M is the set of edges that are fully colored in red color r, and $\bar{\partial}(M)$ is the set of vertices that are not covered by M and they are incident to the red link of a (r, b) variable. Vertices in $\bar{\partial}(M)$ are also called ***M*-exposed**.

Since only one red link is incident to each vertex, there is a natural one-to-one correspondence between a matching M and the color configuration $C(M) = \{M, \bar{\partial}(M)\}$, in which the set of red edges M corresponds to a matching and $\bar{\partial}(M)$ is the set of free vertices relative to M. For example, the graph shown in Figure 1(c) is consistently colored by the set of colors $\{r, b\}$. The initial color assignment is random; the consistency requirement can be easily satisfied if we only assign the red color r to one of the links incident to each vertex.

Definition. The binary **color-exchange** operation "$\otimes$"that operates on two adjacent colored edges $(color_1, color_2)$,$(color_3, color_4)$ is defined by,

$$(color_1, color_2) \otimes (color_3, color_4) = (color_1, color_3) \odot (color_2, color_4)$$

where $\odot$ indicates the adjacency of two colored edges. A color exchange is **effective** if the operation does not increase the number of variables.

An example of the binary color-exchange operation " $\otimes$ " performed on two adjacent variables $\langle u, v\rangle \to (b, r)$ and $\langle v, w\rangle \to (b, r)$ is illustrated in Figure 2. The two variables were eliminated as the result of this color-exchange operation $(b, r) \otimes (b, r) = (b, b) \odot (r, r)$.

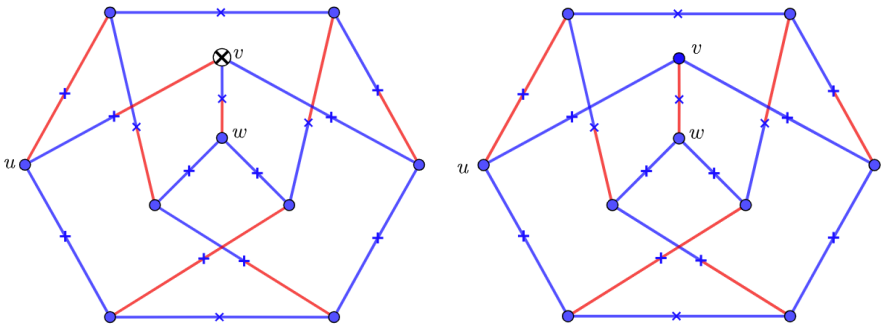


Figure 2. Variable elimination via color exchange operations.

It is important to note that the above color-exchange operation preserves the consistency of a color configuration. Since we only allow effective color exchange operation $(color_1, color_2) \otimes (color_3, color_4) = (color_1, color_3) \odot (color_2, color_4)$, and the effectiveness is assured if either $(color_1, color_2)$ or $(color_3, color_4)$, or both are variables. Thus, the color exchange operation may either eliminate adjacent variables, or move a variable to an adjacent edge. Non-adjacent variables in a consistently colored graph must move next to each other before they can be eliminated.

Since a variable (r, b) is always moving within an alternating path, the symmetric difference operation $M \oplus P$ defined in Lemma 1 is equivalent to the elimination of two variables at the two ends of an augmenting path P. As Figure 3 shows, a (r, b)variable $\langle v_0, v_1\rangle$walks on a complex-colored augmenting path $\langle v_0, v_1\rangle, \langle v_1, v_2\rangle, \langle v_2, v_3\rangle$ by a sequence of color exchanges to cancel another (r, b)variable $\langle v_3, v_4\rangle$. Note that the two end vertices v_0 and v_3 are both free vertices. Thus, according to Theorem 1, finding a maximum matching in a complex-colored graph G is equivalent to repeatedly eliminating variables until remaining variables are all irreducible.

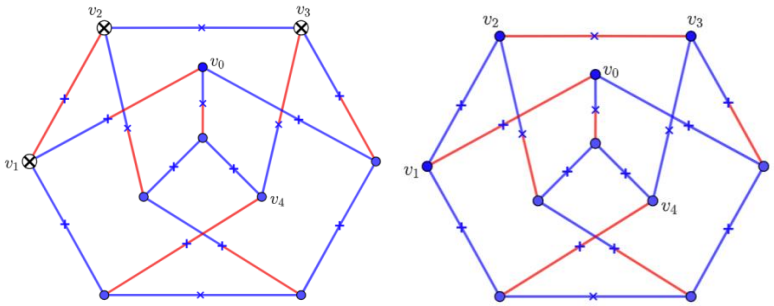


Figure 3. Cancellation of variables by walking on a complex-colored augmenting path.

3. BLOSSOMS: SHRINKING VERSUS DEFLECTION

The algorithm for maximum matching is a process of searching for successive M-augmenting paths starting from an initial matching M. In the exploration process of an M-augmenting path, the M-alternating path is a directional path, which starts from a free vertex and continuously grows in one direction. In the matching M depicted in Figure 4(a), there are two M-alternating paths starting from the free vertex v_0 that can reach v_d, namely

$$P_1 = v_0, v_1, v_a, v_e, v_d, v_x,$$

and

$$P_2 = v_0, v_1, v_a, v_b, v_c, v_d, v_e.$$

If we take the M-alternating path P_1, then we can reach the other free vertex v_x and obtain an M-augmenting path, in which the two end variables $\langle v_0, v_1\rangle \to (r, b)$ and $\langle v_d, v_x\rangle \to (b, r)$ can be eliminated by a sequence of color exchanges. However, if we take the M-alternating path P_2, then we miss this M-augmenting path. This divergent path problem arises when the vertex v_d belongs to an odd cycle, called **blossom** by Edmonds.

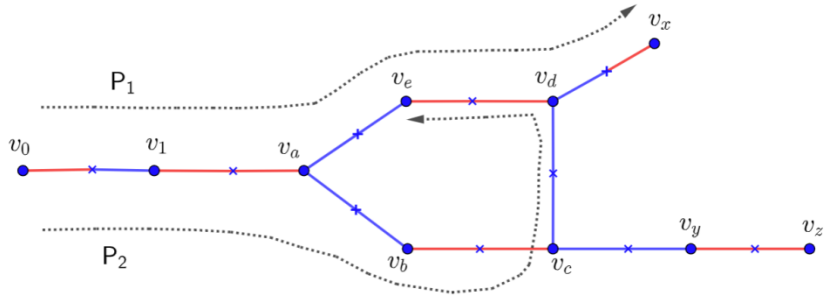


Figure 4(a). The original graph G.

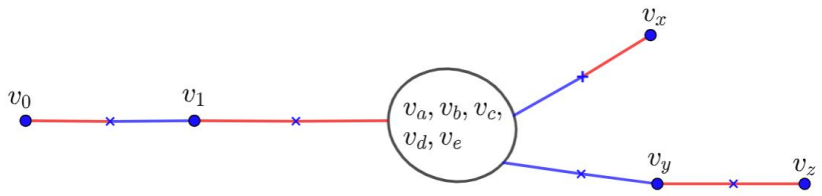


Figure 4(b). The contracted graph G'.
Figure 4. Illustration of Blossom Shrinking.

Edmonds' algorithm solves this difficult problem by shrinking blossoms, or odd cycles, down to single super-vertices, thus to reveal augmenting paths, as Figure 4(b) shows. When we find an augmenting path from a free vertex $v_0$to another free vertex v_x in the contracted graph G', then we immediately obtain an augment path in the original graph G by unshrinking the super-vertices.

Definition. Given a matching $M$and an M-alternating path P starting from a free vertex v_0, the **parity bit** of a vertex v on P, denoted by $\pi(v)$,is determined by the distance (number of edges) between this vertex v and the initial free vertex $v_0$along P. If the distance is even then $\pi(v) = 0$; otherwise, $\pi(v) = 1$.

We adopt the convention that the link incident to the initial free vertex v_0 in an M-alternating path $P$is always colored red, if not we can always change it to red by color-exchanging with its neighbouring red link. With this convention, the parity bit of each vertex on the path Pcan be defined by the following equivalent localized definition:

$$\pi(v) = \begin{cases} 0, & if input link to v is red and output is blue (v is in even state), \\ 1, & if input link to v is blue and output is red (v is in odd state). \end{cases}$$

Consider the two M-alternating paths,

$$P_1 = v_0, v_1, v_a, v_e, v_d, v_c, v_b \text{ and } P_2 = v_0, v_1, v_a, v_b, v_c, v_d, v_e$$

in the graph G shown in Figure 4(a). We write the two paths P_1 and P_2 with their sequences of parities as follows:

$$P_1: v_0\pi(v_0)v_1\pi(v_1)v_a\pi(v_a)v_e\pi(v_e)v_d\pi(v_d)v_c\pi(v_c)v_b\pi(v_b) = v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_c 1 v_b 0,$$
$$P_2: v_0\pi(v_0)v_1\pi(v_1)v_a\pi(v_a)v_b\pi(v_b)v_c\pi(v_c)v_d\pi(v_d)v_e\pi(v_e) = v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_d 1 v_e 0.$$

Comparing the above two sequences, we can sum up the following properties of blossoms.

Property 1. The M-alternating path P always enters the blossom at a vertex $v_a$with even parity bit $\pi(v_a) = 0$,called **base**, because path divergence occurs only when the input to v_a is red and multiple outputs are blue. On the other hand, if the input is blue, then there is only one red output, which is the case of entering the base of an even cycle.

Property 2. The parity bit $\pi(v)$of a vertex$v$in the blossom, other than the base, can be either 0 (even) or 1 (odd), depending on the direction of the path$P$.

Property 3. If the M-alternating path P return to the base v_a and form a blossom, then the last vertex v always possesses an even parity $\pi(v) = 0$, which conflicts with the parity $\pi(v_a) = 0$ of the base vertex v_a.For example, the last vertex v_b in P_1, andv_e in P_2.

In the exploration of M-alternating paths, the difficulty arising from blossoms is mainly due the parity conflicts characterized in Property 2 and 3. The aim of shrinking the blossom to a single super-vertex is two-fold: eliminating the parity conflicts, and reserving all M-alternating paths passing through the blossom.

In contrast to shrinking, the algorithm proposed in this paper deflects the M-alternating path and makes a detour around blossoms. This dynamic exploration mechanism is a two tuple$T = \{P, S\}$, called**trunk**, which consists of an M-alternating path$P$starting from a free vertex, and a stack of **sprout**S that maintains all possible detours of$P$. The M-alternating path $P$is a stack of ordered sequence of vertices, and the sprout S is a stack of ordered sequence of edges, in which each edge is a sprout that represents the starting point of a reserved detour for the alternating path P.

Definition. A vertex v in the M-alternating path P with **even parity** $\pi(v) = 0$ is called a **sprout root** and is abbreviated as ***s*-root**. The set of **free edges** incident with an ***s*-root**v is defined as

$$Sprout(v) = \{\langle v, u\rangle \mid v \in P, \pi(v) = 0, \langle v, u\rangle \in \bar{M}\},$$

and the set of vertices mated with an s-root v by free edges is defined as

$$Mate(v) = \{u \mid v \in P,\ \pi(v) = 0,\ \langle v, u\rangle \in \bar{M}\}.$$

The M-alternating path P is directional; at an odd parity matched vertex v with$\pi(v) = 1$, there is a unique path to continue P from a blue input link to the only red output link. However, at an even matched parity vertex v with$\pi(v) = 0$, the vertex$v$ is an s-root, and the path$P$ can be continued from a red input link to any one of the multiple blue output links. In our DFS algorithm, the path P will arbitrarily select one of the edges in $Sprout(v)$, and keep the others in reserve in the sprout stack S, in case that the path P needs detours in the future.

The searching process of this dynamic trunk$T = \{P, S\}$starts from an initial free vertex v_0 and one of its mates$u \in Mate(v_0)$, meaning that initially we have $P = \{v_0, u\}$with sprout set$S = Sprout(v_0)\backslash\{\langle v_0, u\rangle\}$. As the path P extends, the process keeps adding pairs of vertices to the alternating path$P$, and appending sprouts to the stack S along the extension of path P. If the path P hits another free vertex, then an M-augmenting path is identified and the searching process stops. Otherwise, the exploration process will continue until the path hits a **dead end** or an **active vertex** in P. The latter case indicates that the path P forms a cycle. In either case, the path P will make a detour. The algorithm concedes defeat if the stack of sprout S is empty, otherwise it will retrieve the last sprout in S, namely an edge $e = \langle v_s, v_t\rangle$, and replace the entire sub-path in P starting fromv_s with the sequence v_s, v_t. The algorithm continues the searching process after making the detour. Table 1 lists each step of the searching process starting from the free vertex v_0 in the graph G shown in Figure 4(a).

Table 1.The process of searching for an augmenting path through an odd cycle

Steps	Alternating Path P	Sprout Stack S	Remarks
1 (initialization)	$v_0 0 v_1 1$	$\emptyset$	v_0 is the initial free vertex, add v_0, v_1 to P.
2	$v_0 0 v_1 1 v_a 0 v_b 1$	$\langle v_a, v_e \rangle$	Add v_a, v_b to P and $\langle v_a, v_e \rangle$ to S.
3	$v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_d 1$	$\langle v_a, v_e \rangle, \langle v_c, v_y \rangle$	Add v_c, v_d to P and $\langle v_c, v_y \rangle$to S.
4	$v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_d 1 v_e$	$\langle v_a, v_e \rangle, \langle v_c, v_y \rangle$	Add v_e, v_a to P, the vertex v_a appeared twice in $P$with conflict parity, detect an odd cycle.
5 (detour)	$v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_y 1$	$\langle v_a, v_e \rangle$	Make a detour around cycle. Retrieve sprout $\langle v_c, v_y \rangle$ from S, and replace the sequence v_c, v_d, v_e, v_ain P with v_c, v_y.
6 (dead end)	$v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_y 1$	$\langle v_a, v_e \rangle$	The path hits a dead end at v_z.
7 (detour)	$v_0 0 v_1 1 v_a 0 v_e 1$	$\emptyset$	Make a detour around the dead end v_z. Retrieve sprout $\langle v_a, v_e \rangle$ from$S$, and replace the sequence v_a, v_b, v_c, v_y in P with v_a, v_e. Starting from here, the path is in the clockwise direction of the odd cycle v_a, v_b, v_c, v_d, v_e.
8 (termination)	$v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_x 1$	$\langle v_d, v_c \rangle$	If $\langle v_d, v_x \rangle$ is selected, add v_d, v_x to P, the process may move to v_c or to v_x, in the latter case, the augmenting path $v_0, v_1, v_a, v_e, v_d, v_x$ is identified and the process is stopped. If $\langle v_d, v_c \rangle$ is selected then the next step is 8A.
8A	$v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_c 1$	$\langle v_d, v_x \rangle$	If the process selects $\langle v_d, v_c \rangle$ instead of $\langle v_d, v_x \rangle$ in step 8, then add$v_d, v_c$to P, and $\langle v_d, v_x \rangle$ to S.
9A	$v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_c 1$ $v_b 0 v_a 1$	$\langle v_d, v_x \rangle$	Add v_b, v_a to P, the vertex v_a appeared twice in $P$with conflict parity, detect an odd cycle.
10A (detour and termination)	$v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_x 1$	$\emptyset$	Make a detour around cycle. Retrieve sprout $\langle v_d, v_x \rangle$ from S, and replace the sequence v_d, v_c, v_b, v_ain P with v_d, v_x. The augmenting path $v_0, v_1, v_a, v_e, v_d, v_x$ is identified and the process is stopped.

The parity conflicts will not occur when the alternating path forms an even cycle. As Figure 5 shows, there is only one M-alternating path transits the even cycle because the parity of the **base** vertex v_a of the even cycle is odd with $\pi(v_a) = 1$. Unlike odd cycles, an even cycle is a legitimate two-colored M-alternating cycle, which is naturally compatible with the M-alternating path P. The odd cycle and even cycle displayed in Figure 4 and Figure 5, respectively, clearly demonstrate this key point. Table 2 provides the searching process starting from the free vertexv_0 in the graph shown in Figure 5.

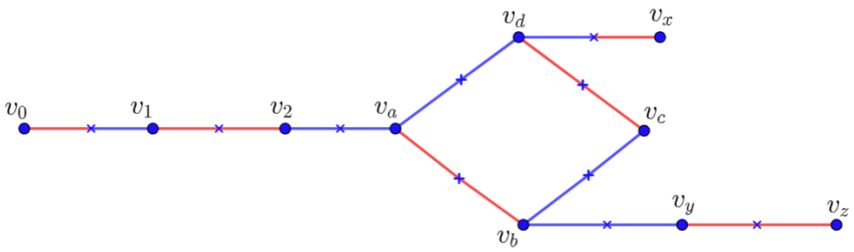


Figure 5. The graph $G$with an even cycle.

Table 2.The process of searching for an augmenting path through an even cycle

Steps	Alternating Path P	Sprout Stack S	Remarks
1 (initialization)	$v_0 0 v_1 1$	$\emptyset$	$v_0$is the initial free vertex, add v_0, v_1 to P.
2	$v_0 0 v_1 1 v_2 0 v_a 1$	$\emptyset$	Add v_2, v_a to P.
3	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_c 1$	$\langle v_b, v_y \rangle$	If the edge $\langle v_b, v_c \rangle$ is selected, add v_b, v_c to P and $\langle v_b, v_y \rangle$ to S. Otherwise, the next step is 3A.
4	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_c 1 v_d 0$	$\langle v_b, v_y \rangle, \langle v_d, v$	Add v_d, v_a to P, the vertex v_a appeared twice in P with same parity, detect an even cycle.
5 (detour and termination)	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_c 1 v_d 0$	$\langle v_b, v_y \rangle$	Make a detour around the cycle. Retrieve sprout$\langle v_d, v_x \rangle$ from$S$, and replace the sequence v_d, v_ain P with v_d, v_x. Theaugmentingpath $v_0, v_1, v_2, v_a, v_b, v_c, v_d, v_x$ is identified and the process is stopped.
3A (dead end)	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_y 1$	$\langle v_b, v_c \rangle$	If the edge $\langle v_b, v_y \rangle$ is selected, add v_b, v_y to P and $\langle v_b, v_c \rangle$to S. Then the path P hits a dead end at v_z.
4A (detour)	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_c 1$	$\emptyset$	Make a detour around the dead end v_z. Retrieve sprout$\langle v_b, v_c \rangle$ from$S$, and replace the sequence v_b, v_y in P with v_b, v_c.
5A (termination)	$v_0 0 v_1 1 v_2 0 v_a 1 v_b 0 v_c 1 v_d 0$	$\langle v_d, v_a \rangle$	Add v_d, v_x to $P$and $\langle v_d, v_a \rangle$to S. The augmenting path $v_0, v_1, v_2, v_a, v_b, v_c, v_d, v_x$ is identified and the process is stopped.

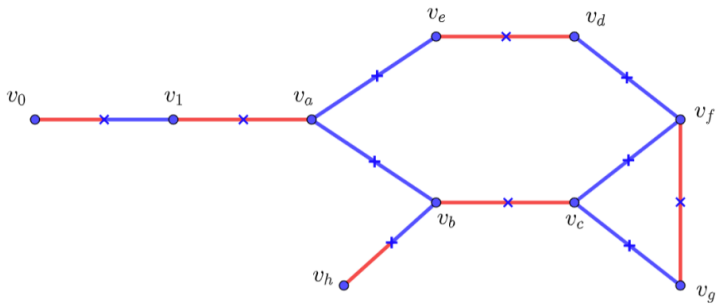


Figure 6. The graph G with two nested odd-cycles.

The searching process adaptively changes the directional alternating path according to the topology of the graph. The graph G displayed in Figure 6 has two nested odd cycles. Starting at free vertex v_0, the alternating path P encountered odd cyclesfour times before it finds another free vertex v_h. The following sequence of the searching process reveals the resilience of the dynamic trunk$T = \{P, S\}$.

1. $P = v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_f 1 v_g 0 v_c 1$

   $S = \{\langle v_a, v_e\rangle, \langle v_c, v_g\rangle\}$
2. $P = v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_g 1$

   $S = \{\langle v_a, v_e\rangle\}$
3. $P = v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_g 1 v_f 0 v_d 1 v_e 0 v_a 1$

   $S = \{\langle v_a, v_e\rangle, \langle v_f, v_c\rangle\}$
4. $P = v_0 0 v_1 1 v_a 0 v_b 1 v_c 0 v_g 1 v_f 0 v_c 1$

   $S = \{\langle v_a, v_e\rangle\}$
5. $P = v_0 0 v_1 1 v_a 0 v_e 1$

   $S = \emptyset$
6. $P = v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_f 1 v_g 0 v_c 1 v_b 0 v_a 1$

   $S = \{\langle v_b, v_h\rangle\}$
7. $P = v_0 0 v_1 1 v_a 0 v_e 1 v_d 0 v_f 1 v_g 0 v_c 1 v_b 0 v_h 1$

   $S = \emptyset$

/* augmenting path identified successfully. */

4. THE DFS ALGORITHM

In this Section, we summarize the DFS algorithm and describe the details of the searching process. The algorithm consists of two phases: a *growing phase* and a *pruning phase*. The alternating path P and the sprout stack S will be updated in both phases.

Input: A general graph $G(V, E)$, a color configuration $C(M) = \{M, \bar{\partial}(M)\}$of graph $G(V, E)$ with a current matching M, and a free vertex $v_0 \in \bar{\partial}(M)$.

Idea: Explore a trunk $T = \{P, S\}$ from the$M$-exposed vertex v_0, stretching the alternating path P and the sprout stack S as far as possible. In each step, an ordered pair of vertices (v_a, v_b)will be

added to the alternating path P; the first vertex v_a is an s-root with even parity in P. The edge $e = \langle v_a, v_b \rangle$ is a sprout selected from the set $Sprout(v_a)$; the rest edges in $Sprout(v_a) \backslash \{\langle v_a, v_b \rangle\}$ will then be added to S. If a dead end was detected or a cycle was formed, the trunk T will make a detour according to the last sprout in stack S. Declare a failure if S is empty, otherwise continue the process to reach another free vertex that yields an augmentation.

Initialization: $P = \{v_0, v_1\}, S = Sprout(v_0) \backslash \{\langle v_0, v_1 \rangle\}$.

Iteration: (**Growing Phase**) If the next pair of vertices v_a, v_b extended from the current alternating path P are not in P, then perform the following updating operation:

$$P = P \cup \{v_a, v_b\},$$
$$S = S \cup Sprout(v_a) \backslash \{\langle v_a, v_b \rangle\}.$$

(Pruning Phase) When a dead end or a cycle was found, stop if $S = \emptyset$ and there is no M-augmenting path from v_0, otherwise, retrieve the last sprout $\langle v_a, v_b \rangle$ from S, eliminate all vertices after v_a in the path $P = v_0, \cdots, v_a, \cdots, v_x$and replace them with v_a, v_b. Update trunk T as follows and continue the searching process:

$$P = v_0, \cdots, v_a, v_b,$$
$$S = S \backslash \{\langle v_a, v_b \rangle\}.$$

In the pruning phase, we implicitly claim that if the edge $\langle v_a, v_b \rangle \in S$then the vertex $v_a \in P$. This claim is always valid because the vertex $v_a$and the edge $\langle v_a, v_b \rangle \in Sprout(v_a)$ were added to P and S, respectively and simultaneously, in the growing phase.

The algorithm terminates when no augmenting paths exist. The searching process halts when either an augmenting path was identified, or every alternating path starting from a free vertex was inspected and returned with an empty sprout stack. We show in the following lemma that all possible alternating paths starting from a free vertex v_0 will be visited if the exploration process ends with an empty sprout stack.

Lemma 2. If an alternating path P starting from a free vertex v_0 ends the searching process with an empty sprout stack $S = \emptyset$, then Phas visited every alternating path starting from v_0.
Proof. Suppose $Q = v_0, \cdots, v_x, v_a, v_b$ is the shortest alternating path that P has never visited, where $\pi(v_a) = 0$ and $\pi(v_b) = 1$. Then neither Phas visited the alternating path $Q' = v_0, \cdots, v_x$, because $\langle v_x, v_a \rangle \in Sprout(v_x)$but it was eventually disappeared in the final sprout set $S = \emptyset$.That is, if P has visited$Q' = v_0, \cdots, v_x$then it certainly has visited $Q = v_0, \cdots, v_x, v_a, v_b$through the sprout $\langle v_x, v_a \rangle$, which is impossible according to our assumption. On the other hand, if P has never visited$Q' = v_0, \cdots, v_x$, then this contradicts our assumption that $Q = v_0, \cdots, v_x, v_a, v_b$is the shortest alternating path that P has never visited.■

In the DFS algorithm, we assume that if a free vertex $v_0$failed to find another free vertex through an alternating path, then $v_0$will never access any other free vertices, even if other augmenting paths modified the graph configuration. By definition, any alternating path in a maximum matching M should contain at most one M-exposed vertex. This point can be further elaborated by the Gallai-Edmonds decomposition of a graph G, in which every M-exposed vertex v_0 of a maximum matching$M$is locked up in an odd component of G. This isolation property ensures that repeating an exploring process starting from the same free vertex $v_0$is not necessary.

The Sylvester's graph is a good example to illustrate the isolation property of free vertices in a maximum matching M. As Figure 7 shows, the three odd components G_1,G_2,G_3 of graph $G$are connected by a vertex v_a, we observe the following properties:

1. Deleting v_a,M covers all but one vertex of each odd component $G_i, i = 1,2,3$.
2. Mcovers the vertex v_a.
3. If M matches one of the free vertices in $G_i, i = 1,2,3$, with v_a, then the other two free vertices in G_j , $j \neq i$, will be isolated, and they cannot be connected by an alternating path.

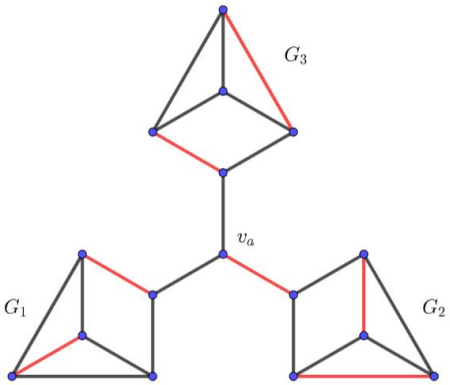


Figure 7. A Maximum Matching of The Sylvester's Graph.

Definition. In a graph $G(V,E)$, for $S \subseteq V(G)$, let $N_G(S)$ denote the set of vertices in $G - S$which have at least one neighbour in S, and let $G[S]$ denote the subgraph of G induced by S. The graph G is **factor-critical** if $G - v$ has a perfect matching for every vertex $v \in V(G)$. A matching in G is **near-perfect** if it matches all but one vertex of G.

A factor-critical graph is connected, and has an odd number of vertices. Simple examples include odd-length cycle C_n and the complete graph K_n of odd order n.

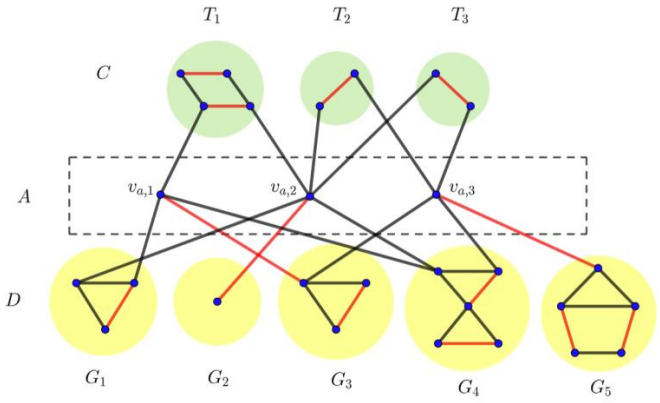
Definition. In a graph $G(V,E)$, let B be the set of vertices covered by every maximum matching in G, and let $D = V(G) - B$. The set $B$is further partitioned into $B = A \cup C$, where $A$is the set of vertices that are adjacent to at least one vertex in D, and $C = B - A$. The Gallai-Edmonds decomposition of G is the partition of $V(G)$ into three mutually disjoint subsets $V = A \cup C \cup D$.

Theorem 2 (Gallai-Edmonds Structure Theorem[14]) Let A, C, D be the sets in the Gallai-Edmonds Decomposition of a graph $G(V,E)$. Let $T_1, \cdots, T_l$ be the components of $G[C]$, and G_1, $\cdots, G_k$ be the components of $G[D]$. If M is a maximum matching in G, then the following properties hold:
1. Each $T_i, i = 1,2,\ldots,l$, is an even component, and$M$ restricts to a perfect matching on T_i.
2. Each $G_i, i = 1,2,\ldots,k$, is an odd component, which is factor-critical, and$M$ restricts to a near-perfect matching on G_i.
3. M completely matches A into distinct components $G_1, \cdots, G_k$ of $G[D]$. ■
A detailed proof of this theorem is given by Lovász and Plummer in[1, 2], and a short proof is provided by West in [15]. The property 3 in the above theorem can be explained by Hall's Theorem. Contracting each component $G_i$of$G[D]$ to a single vertex$v_{g,i}$, we define an auxiliary bipartite graph $H(A \cup Y, E_H)$ as follows:

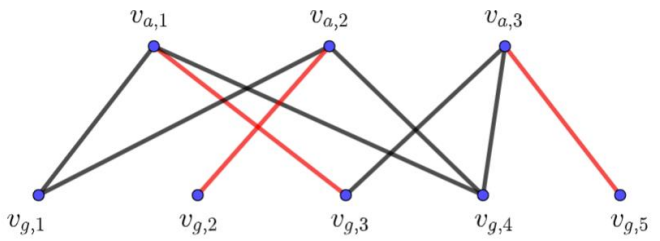
$$Y = \{v_{g,1}, v_{g,2} \cdots, v_{g,k}\}, \text{ and } A = \{v_{a,1}, v_{a,2} \ \cdots, v_{a,h}\},$$

$$E_H = \{(v_{a,j}, v_{g,i}) | \, v_{a,j} \in A having an neighbour in G_i\}.$$

In the Gallai-Edmonds decomposition and a maximum matching M of a general graph G, as Figure 8 illustrates, the isolated M-exposed vertex in G_1 cannot access to that in G_4 by any alternating paths. It can be shown that Hall's condition $|S| \leq |N_H(S)|$ holds for any $S \subseteq A$ [16], thus the M restricts to a matching on bipartite graph H that covers A.



(a) The decomposition $V = A \cup C \cup D$.



(b) Auxiliary bipartite graph H.

Figure 8. The Gallai-Edmonds decomposition of a general graph G.

Since each odd component G_i is factor-critical, and any vertex of G_i can be the one unmatched by a maximum matching M. Thus, the only unmatched vertex in each odd component G_i can either be matched with a vertex $v_a \in A$, or be isolated in the odd component G_i. Therefore, any free vertex can only be the source of an augmenting path at most once in the DFS algorithm. Since the initial number of free vertices is upper bounded by the order of $O(n)$, and the length of each alternating path P is proportional to the number of edges $m = |E|$, the complexity of the DFS algorithm is given in Theorem 3.

Theorem 3. The DFS algorithm can determine a maximum matching of a general graph in $O(mn)$ time with space complexity $O(n)$.

Experiments were conducted to verify the performance of our maximum matching algorithm. A set of Δ-regular graphs with n vertices and $m = \frac{\Delta n}{2}$ edges was randomly generated. Figure 9 shows the experimental results of average running time, in which 25 graphs were randomly generated for every pair of (Δ, n), $n = 100, 200, \ldots, 2500$ and $\Delta = 3, 4, 5$. These experimental results confirm the performance of our maximum matching algorithm given in Theorem 3. As

shown in Figure 9, for each set of graphs under consideration, the running time of our algorithm is on the order of $O(n^2)$for a given degree Δ.

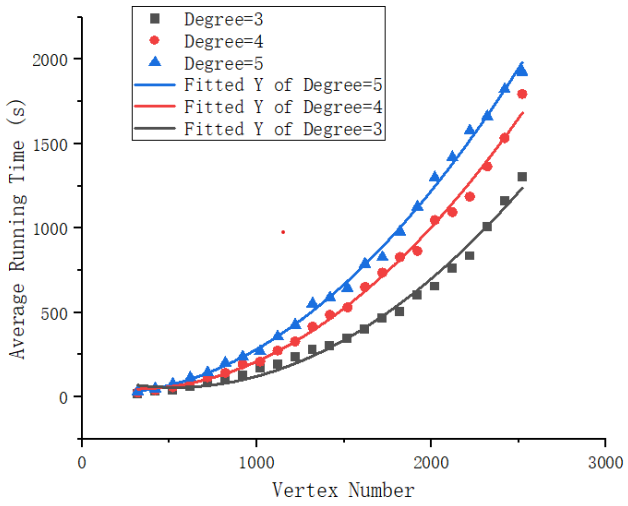


Figure 9. The running time of the maximum matching algorithm.

5. CONCLUSION

The fundamental problem of finding maximum matching in general graphs is the existence of odd cycles, or blossoms. Instead of shrinking blossoms, this paper proposed a deflection algorithm to cope with the parity conflicts caused by odd cycles. This new algorithm achieves $O(mn)$ time complexity with $O(n)$ data structure. This newly proposed algorithm is complementary to Edmonds' blossom algorithm in two important aspects: depth-first search (DFS) versus breadth-first search (BFS), and deflection from blossoms versus shrinking of blossoms. In the future, we will explore the application of this method to maximum matching of weighted graphs.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Shahbaz Khan of Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee, India, for many useful criticism and suggestions.

REFERENCES

[1] Lovász, László& Plummer, Michael D. (1986)*Matching Theory*, Vol. 29, Annals of Discrete Mathematics,North-Holland, Amsterdam.

[2] Lovász, László& Plummer, Michael D.(2009) *Matching Theory*, Vol. 367, American Mathematical Society.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009)*Introduction to Algorithms*, MITpress.

[4] Hopcroft, John E. & Karp, Richard M., (1973) "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs",SIAM *Journal on Computing*, Vol. 2, No. 4, pp. 225–231.

[5] Edmonds, Jack, (1965) "Paths, trees, and flowers",*Canadian Journal of Mathematics*, Vol. 17, pp. 449–467.

[6] Gabow, Harold N., (1976) "An efficient implementation of Edmonds' algorithm for maximum matching ongraphs",*Journal of the ACM (JACM)*, Vol. 23, No. 2, pp. 221–234.
[7] Gabow, Harold N. &Tarjan, Robert E., (1985)"A linear-time algorithm for a special case of disjoint set union",*Journal of Computer and System Sciences*, Vol. 30, No. 2, pp. 209–221.
[8] Micali,Silvio&Vazirani, Vijay V., (1980) "An $O(\sqrt{|V||E|})$ algorithm for finding maximum matching ingeneral graphs", in the 21st *Annual Symposium on Foundations of Computer Science* (1980), pp. 17–27, IEEE.
[9] Vazirani, Vijay V., (1994) "A theory of alternating paths and blossoms for proving correctness of the$O(\sqrt{|V||E|})$general graph maximum matching algorithm",*Combinatorica*, Vol. 14, No. 1, pp. 71–109.
[10] Tarjan, Robert E. (1983)*Data Structures and Network Algorithms*,Society for industrial and Applied Mathematics.
[11] Berge, Claude, (1957) "Two theorems in graph theory",*Proceedings of the National Academy of Sciences ofthe United States of America*, Vol. 43, No. 9, pp. 842–844.
[12] Lee, Tony T., Wan, Y., & Guan, H., (2013) "Randomized Δ-edge colouring via exchanges of complexcolours",*International Journal of Computer Mathematics*, Vol. 90, No. 2, pp. 228–245.
[13] Wang, L., Ye, T., Lee, Tony T., & Hu, W., (2018) "A parallel complex coloring algorithm for scheduling ofinput-queued switches", IEEE *Transactions on Parallel and Distributed Systems*, Vol. 29, No. 7,pp. 1456-1468.
[14] Gallai, Tibor, (1963) "Kritische graphen ii",*Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, Vol. 8, pp. 373–395.
[15] West, Douglas B.,(2011) "A short proof of the Berge–Tutte formula and the Gallai–Edmonds structuretheorem",*European Journal of Combinatorics*, Vol. 32, No. 5, pp. 674–676.
[16] West, Douglas B. (2001)*Introduction to Graph Theory*, Prentice-Hall, Inc., Upper Saddle River.

AUTHORS

TONY T. LEE received the BSEE degree from National Cheng Kung University, Taiwan, and the M.S. and Ph.D. degrees in electrical engineering from the Polytechnic Institute of New York University (now Tandon School of Engineering, New York University), Brooklyn, NY, USA. From 2013 to 2017, he was a Zhiyuan Chair Professor with the Electronics Engineering Department, Shanghai Jiao Tong University. From 1993 to 2013, he was a Chair Professor with the Information Engineering Department, The Chinese University of Hong Kong. From 1991 to 1993, he was a Professor of electrical engineering with the Polytechnic Institute of New York University. From 1989 to 1991, he was an adjunct Associate Professor with the Department of Electrical Engineering of Columbia University, New York. He was with AT&T Bell Laboratories, Holmdel, NJ, USA, from 1977 to 1983, and with Bellcore (now Telcordia Technologies), Morristown, NJ, USA, from 1983 to 1993. He is currently a Professor with the School of Science and Engineering, The Chinese University of Hong Kong (Shenzhen). He is a Fellow of IEEE and HKIE. He has received many awards, including the 1989 Leonard G. Abraham Prize Paper Award from the IEEE Communication Society, and the 1999 National Natural Science Award from China. He has served as an Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS, and an Area Editor of the Journal of Communication Network.

Dr. Bojun Lu received her Bachelor's degree in Mathematics and Applied Mathematics from the University of Science and Technology of China (USTC), China in 2008, and her Ph.D. degree at the Department of Systems Engineering and Engineering Management from The Chinese University of Hong Kong (CUHK), Hong Kong in July 2014. After graduation, Bojun enriched her industrial experiences in quantitative finance industry with positions of quantitative researcher for around three years. She joined the CUHK-Shenzhen with a Lecturer position in January 2018, and is promoted as Assistant Professor (Teaching) in 2022. Her teaching area includes applied mathematics, mathematical statistics, and combinatorial mathematics. Her research interests include algorithms and theory in matching problems with possible real-world applications, methodologies in multivariate data analysis, quantitative finance, lattice theory and possible applications in cryptography encoding and decoding.

Hanli Chu received his Bachelor's degree in Electronic Information Engineering from the Chinese University of Hong kong, Shenzhen (CUHKSZ).In September 2021, he started his master programme study in Computer Information Engineering at CUHKSZ. His research interests include image classification, semantic segmentation, etc.