A matching algorithm based on the depth first search for the general graph

Chengcheng Yu
Department of Network Engineering,
Hainan College of Software Technology,
Qionghai, 571400, Hainan, China
1670836831@qq.com

Sheng Zhonge
College of Information Science and Technology,
Hainan University,
Haikou, 570228, Hainan, China
shzhong@hainu.edu.cn

***Abstract*—In this paper, a matching algorithm of general graph based on depth-first traversal is proposed. The algorithm does not need to shrink and expand treatment when a flower is searched. This algorithm's time complexity of search an augmenting path is equal to corresponding graph's depth-first traversal algorithm's time complexity, it is one of the most efficient algorithm. Experiments show that this algorithm can correctly handle the associated practical problems, and have the correct conclusion.**

Keywords Graph matching; Matching algorithm; Priority traversal; Flowering algorithm;

I. INTRODUCTION

The research on the matching theories and the matching algorithms are one of the core content in the area of graph theory and application research. It has a strong application background. The research results are widely used in process arrangement, personnel assignment, information transfer, and transportation problem etc.

The algorithm about matching problem is proposed first by Kuhn [1] and Hall [2] for search the perfect matching in a bipartite graph, it is a linear programming algorithm. In 1965, the flower algorithm is proposed by Edmonds to look for the perfect matching in a non-bipartite graph, it is a effective algorithm [3, 4]. The better implementation algorithms [5, 6], the effective implementation algorithm of Edmonds method is proposed by Gabow [7]. Some other effective label method similar to Gabow algorithm [8, 9, 10], the algorithm's time complexity to $O(n^{5/2})$.

In this paper, a matching algorithm of general graph based on depth-first traversal is proposed. The algorithm does not need to shrink and expand treatment when a flower is searched.

II. RELATED CONCEPTS ABOUT THE EDGE MATCHING OF THE GENERAL GRAPH

Definition 1 $G = (V, E)$, G is a graph where V is a vertex set and E is a edge of the graph G. $M \subseteq E$ is called a match of the graph G if satisfy that any two edge in M have not the same end point.

The matching problem is to look for maximum matching M (the maximum number of edges). $|M| = \lfloor |V|/2 \rfloor$, M is called a perfect matching. Obviously, the perfect matching must be a maximum matching.

Definition 2 Suppose M is a matching of the graph $G = (V, E)$, $e \in E$, $u \in V$, $v \in V$, the edge $e \in M$ is called a matching edge. The edge $e \notin M$ is called a free edge. The u and v each other is called a consort if $e_{uv} \in M$. The u is called a covering point if that u is the end point of the edge in M. The u is called an uncovering point if that u is not the end point of the edge in M. $p = (u_1, u_2, \cdots, u_k)$ is a path in the G, the p is called an alternating path if all adjacent edge $e_{u_{i-1}u_i}, e_{u_i u_{i+1}}, (1 < i < k)$ are not free edges or matching edges at the same time. The alternating path is called an augmenting path if u_1, u_k are an uncovering point. The vertex u_{i-1}, u_{i+1} are called the exterior point and the u_i is called the interior point if the $e_{u_{i-1}u_i}$ is a free edge and the $e_{u_i u_{i+1}}$ is a matching edge.

Lemma 1 p=$(u_1, u_2, \cdots, u_k)$, p is an augmenting path about matching M of the graph G, P is a set of all edge of the p, then the $M' = P - M$ is a matching that it contains $|M| + 1$ edges.

Theorem 1 The matching M of the graph G is a maximum matching if and only if there is not exist an augmenting path about matching M in the G.

Definition 3 The circle is called odd circle if the number of edges is odd in the circle. The odd circle is called a flower if just one consort of vertexes in the circle is not in the circle. The vertex in the flower is called a flower stalks if the consort of the vertex is not in the circle. Two vertex in the flower are called petal if they adjoin to flower stalks.

The flower is found possible when the matching algorithm to search augmenting path. Then, here are some theorem can help us to deal the flowers

Theorem 2 The flower b is found when the matching algorithm to search augmenting path about the matching M from the uncovering point u start in the graph G. Then any point v in the flower b exist an alternating path from u to v in the graph G, and last one edge must is matching edge in the alternating path

Definition 4} Suppose b is a flower about matching M in the graph G. A new graph G/b is called the flower b contraction graph. Where $G/b = (V/b, E/b)$, $V/b = V - V_b + \{v_b\}$, the V_b is a set of vertexes in the flower b, the v_b is a new vertex, the E/b is a set of edges, $E_{V-\{b\}}$ is a set of the edges which do away with all vertexes in the b, the E_{v_b} is a set of the edges, $E_{v_b} = \{e_{iv_b} | e_{ij} \in E, i \notin V_b$ and $j \in V_b$, let $j = v_b\}$, $E/b = E_{V-\{b\}} + E_{v_b}$.

The v_b is an exterior point, augmenting path searching can from this point to continue.

Theorem 3 The flower b is found when the matching algorithm to search augmenting path about the matching M from the uncovering point u start in the graph G. Then there is an augmenting path about the matching M from the covering point u start if only if exist an augmenting path about the matching M in the graph G/b.

This theorem tell us that it will not affect the existence of the augmented path about M in a graph G after the flower shrinkage,

Theorem 4 If there is exist an augmented path about the matching M from the uncovering point u start in the graph G, let p is an augmented path, then there is exist an augmented path about the matching P - M from the uncovering point u start in the graph G, the P is a set of edges in the p.

III. THE ALGORITHM IDEA

Let $M=\phi$ is a matching. $S_u = \phi$ is an uncovering point set. $F_p = \phi$ is a flag set of the vertex p. A_p is a set of all adjacency point of the vertex p, ($p=v_1, v_2, \cdots, v_n$), n is the number of vertices of the graph G. $f_c(p)$ is a function return the consort of the vertex p. *p.pointer* is a pointer of the vertex p, it points to the pioneer in the searching path. *Stack* is a stack, initialized to an empty, *stack.push*(x) push x into the stack, *stack.pop*() pop an element from the stack, *stack.read*() read the stack top element. The "*COVER*" is the covering point flag. The "*UNCOVER*" is the uncovering point flag. The "*INTERIOR*" is the interior point flag. The "*EXTERIOR*" is the exterior point flag.

Using the algorithm of the depth-first traversal to traverse the graph G. In the process of traversal, suppose p is the current search vertex. $F_p = F_p \cup \{UNCOVER\}$, the vertex p marked as the uncovering point, and $S_u = S_u \cup \{p\}$, the p merge into S_u, and *stack.pop*() if $F_p = \phi$ and all points in A_p are visited, else, let $q \in A_p$ is going to visit next time, $F_p = F_p \cup \{COVER\}$, the vertex p marked as the covering point, $F_q = F_q \cup \{COVER\}$, the vertex q marked as the covering point, $M = M \cup \{e_{pq}\}$, the edge e_{pq} append into M, $p = q$, the vertex q to act as the current searching point, and *stack.push*(p) if $F_p = \phi$. After the completion of the traversal, get the initial matching M. The M is maximum matching if $S_u = \phi$, then the algorithm end.

While $S_u \neq \phi$ to do searching an augmenting path as follows procedure:

1. $\forall p(F_p = F_p$-{*INTERIOR*, *EXTERIOR*}, $p \in V$, Clear all interior point flag and exterior point flag.
2. Choose an uncovering point $p \in S_u$ as the starting point of the augmented path, and $F_p = F_p \cup \{EXTERIOR\}$, *stack.push*($p$), and $S_u = S_u$ - $\{p\}$. *EXTERIOR* flag append to the flag set of the vertex p, p push into the stack, delete p from the uncovering point set.
3. Search the $q \in A_p$ that satisfy $q \in S_u$ if any exist, at this time, the algorithm found an augmenting path. $F_q = F_q \cup \{COVER\}$-{*UNCOVER*} and to do $M = M \cup \{e_{pq}\}$, $q = f_c(p)$, $M = M \cup \{e_{pq}\}$, p = *p.pointer*, until *UNCOVER* $\in F_p$, $F_p = F_p \cup \{COVER\}$-{*UNCOVER*}. Go to repeat start to searching next augmenting path.
4. Search the $q \in A_p$ that satisfy *INTERIOR* $\notin F_q$ and *EXTERIOR* $\notin F_q$ if any exist, at this time, to do *stack.push*(q), $F_q = F_q \cup \{INTERIOR\}$, $q = f_c(q)$, *stack.push*(q), $F_q = F_q \cup \{EXTERIOR\}$, *q.pointer* = p, p=q. Go to 3. A covering point q, that it is not visited, is found, The q and its consort push into the stack, *INTERIOR* flag append to the flag set of the vertex q, *EXTERIOR* flag append to the flag set of the consort of the vertex q, the pointer of the consort of the vertex q point to the vertex p, the consort of the vertex q regard as the current searching point.
5. Search the $q \in A_p$ that satisfy *EXTERIOR* $\in F_p$, *INTERIOR* $\in F_r$where the $r = f_c(p)$ is the consort of the p, *EXTERIOR* $\in F_q$ and *EXTERIOR* $\in F_t$ where the $t = f_c(q)$ is the consort of the q if any exist, at this time, an alternating path is found, to do *r.pointer* = q, $F_r = F_r \cup \{COVER\}$-{*UNCOVER*}, $q = r$, p = *p.pointer*, $r = f_c(p)$, until *EXTERIOR* $\in F_r$, go to 7.
6. Search the $q \in A_p$ that satisfy *EXTERIOR* $\in F_p$ and *INTERIOR* $\in F_r$ where the $r = f_c(p)$ is the consort of the p, *EXTERIOR* $\in F_q$, and $f_c(q)$ is not exist or *INTERIOR* where the $t = f_c(q)$ is the consort of the q, if any exist, at this time, a flower is found, let $s = q$, to do *r.pointer* = q, $F_r = F_r \cup \{COVER\}$-{*UNCOVER*}, $q = r$, p = *p.pointer*, $r = f_c(p)$, until p = s.
7. *stack.pop*(), go to repeat start to searching next augmenting path if the stack is empty. P = *stack.read*(), *stack.pop*() if *INTERIOR* $\in F_p$, else, go to 3.

The algorithm description as follows:

```
Matching DFSGM(G)
M = ϕ; S_u = ϕ; ∀(p ∈ V)F_p = ϕ; Set the stack is empty;
start by finding any p ∈ V; stack.push(p);
while the stack is not empty do
    p = stack.read();
    if ∃(q ∈ A_p)F_q = ϕ then
        stack.push(q);
        if F_p = ϕ then
            F_p = F_p ∪ {COVER}; F_q = F_q ∪ {COVER};
            M = M ∪ {e_pq};
        end if
        p = q;
    else
        if F_p = ϕ then
            F_p = F_p ∪ {UNCOVER}; S_u = S_u ∪ {p};
        end if
        stack.pop();
    end if
end while
if S_u = ϕ then
    return M;
end if
while S_u ≠ ϕ do
    ∀(p ∈ V) F_p = F_p − {INTERIOR, EXTERIOR};
    start by finding any p ∈ S_u;
    F_p = F_p ∪ {EXTERIOR}; stack.push(p); S_u = S_u − {p};
```

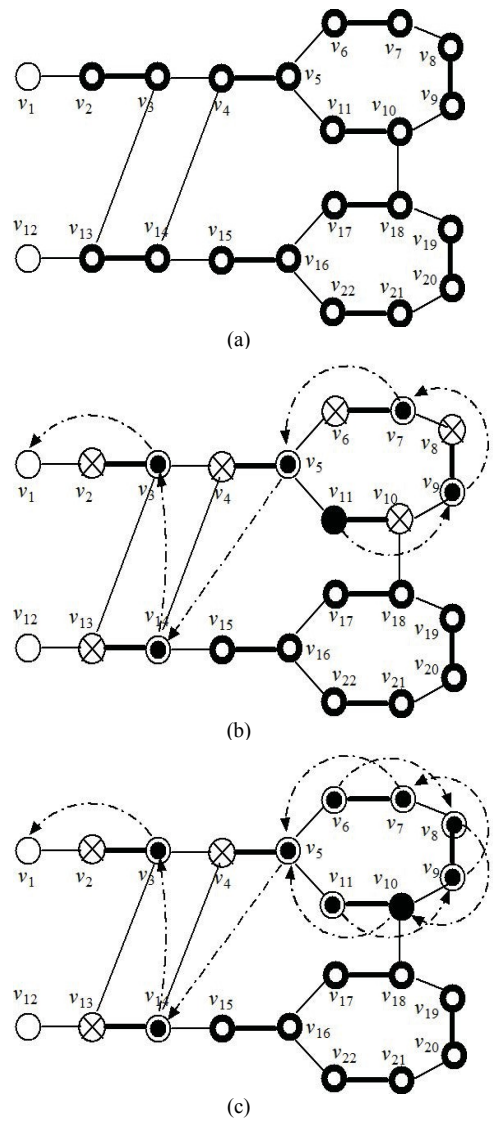
```
27:     if ∃(q ∈ A_p) q ∈ S_u then
28:        F_q = F_q ∪ {COVER} − {UNCOVER};
29:        while UNCOVER ∉ F_p do
30:           M = M ∪ {e_pq}; q = f_c(p);
31:           M = M − {e_pq}; p = p.pointer;
32:        end while
33:        M = M − {e_pq};
34:        F_p = F_p ∪ {COVER} − {UNCOVER}; go to 23;
35:     end if
36:     if ∃(q ∈ A_p) (INTERIOR ∉ F_q and EXTERIOR ∉ F_q) then
37:        stack.push(q); F_q = F_q ∪ {INTERIOR}; q = f_c(q);
38:        stack.push(q); F_q = F_q ∪ {EXTERIOR};
39:        q.pointer = p; p = q; go to 27;
40:     end if
41:     if ∃(q ∈ A_p) EXTERIOR ∈ F_q then
42:        r = f_c(p); t = f_c(q);
43:        if INTERIOR ∈ F_r and EXTERIOR ∈ F_t then
44:           while EXTERIOR ∉ F_r do
45:              r.pointer = q;
46:              F_r = F_r ∪ {COVER} − {UNCOVER};
47:              q = r; p = p.pointer; r = f_c(p);
48:           end while
49:           go to 60;
50:        end if
51:        if INTERIOR∈F_r and (t is not exist or INTERIOR∈F_t) then
52:           s = q;
53:           repeat
54:              r.pointer = q;
55:              F_r = F_r ∪ {COVER} − {UNCOVER};
56:              q = r; p = p.pointer; r = f_c(p);
57:           until p = s
58:        end if
59:     end if
60:     stack.pop();
61:     if the stack is empty then
62:        go to 23;
63:     end if
64:     p = stack.read();
65:     while INTERIOR ∈ F_p and the stack is not empty do
66:        stack.pop(); p = stack.read();
67:     end while
68:     if the stack is not empty then
69:        go to 27;
70:     end if
71: end while
72: return M ;
```

IV. THE EXAMPLE OF THE ALGORITHM

Using the algorithm of the depth-first traversal to traverse the graph *G*, to get the initial matching *M*, as shown in the Fig. 1. (a). From the uncovering point v_1 start to search an augmenting path. When searching to the vertex v_{11}, the flower $v_5v_6v_7v_8v_9v_{10}v_{11}$ is found, the v_5 is flower stalks of the flower, as shown in the Fig. 1. (b). For all interior point vertex in the flower to add a pointer, at this time, all vertex in the flower can arrive go by the alternating path $v_5v_7v_9v_{11}$ or $v_5v_{10}v_8v_6$, therefore, change all interior point to exterior point. After processing the flowers, to pop a point from the stack, start by the stack top element to continue to search an augmenting path, as shown in the Fig. 1. (c). When searching to the vertex v_{15}, the flower $v_{14}v_{15}v_{16}v_{17}v_{18}v_{10}v_{11}v_5v_4$ is found, the v_{14} is flower stalks of the flower, as shown in the Fig. 1. (d). Processing mode ditto, you can see the result in Fig. 1. (e).

When the current searching point is the vertex v_{19}, find the vertex v_{18} and its consort v_{17} is all exterior point, as shown in the Fig. 1. (f), at this time, the alternating path of that arrive to the v_{19} come over from a vertex of the same flower with vertex v_{18}, else, from a point in the flower that have been search start, can search to the v_{19}. In such cases, merely modifying the interior point in the path $v_{19}v_{20}v_{21}v_{22}$ that is not in the flower, as shown in the Fig. 1. (g).

When the stack popping back to v_4, the flower $v_3v_{13}v_{14}v_{15}v_{16}v_{17}v_{18}v_{10}v_{11}v_5v_4$ is found, the v_3 is flower stalks of the flower, as shown in the Fig. 1. (g). You can see the processing result in Fig. 1. (h). When the stack popping back to v_{13}, the uncovering point is found, Finally, the augmenting path $v_{12}v_{13}v_{14}v_{15}v_{16}v_{17}v_{18}v_{10}v_{11}v_5v_4v_3v_2v_1$ is found, as shown in the Fig. 1. (i).



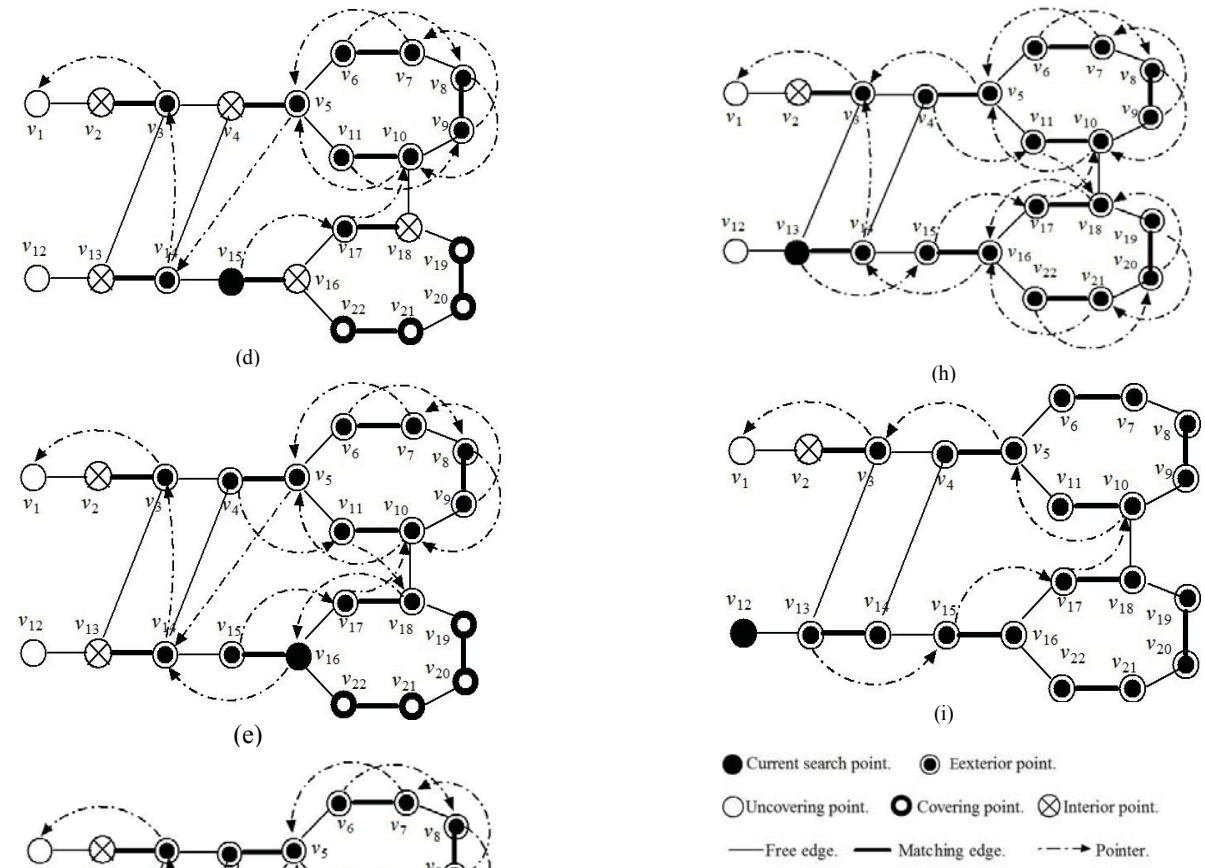


Fig. 1. The example of the algorithm

V. ALGORITHM ANALYSIS

Theorem 5 The DFSGM algorithm is correct.

Prove In algorithm DFSGM searching process, only in the dispose of flower is different with the bipartite graph search. The algorithm DFSGM let all vertex of the flower will become the exterior point, such, in the backtracking courses of the stack out, each vertex in the flower can as start point to search, it is equivalence with the flower contract to a vertex. The algorithm deals the flower is equivalence with Edmonds algorithm. Prove algorithm is correct.

The DFSGM algorithm to avoid the shrinkage, expansion of flower. After finding maximum matching, when the vertex or edge of the graph to add or delete operation, using DFSGM algorithm can quickly calculate the new maximum matching.

- Append a vertex. The appended vertex as start point to searching an augmenting path.
- Delete a vertex. Change the consort point of the deleted vertex to uncovering point, and as start point to searching an augmenting path.
- Append a edge. Append the edge into matching M if two end point of the edge are uncovering point, or the uncovering point as start point to searching an augmenting path if an end point of the edge is covering point and the other one of the edge is uncovering point, or the tow covering point as start point to searching two

alternating path of non-intersect and uncovering point as end points, if any exist, an augmenting path is found.

- Delete a edge. Make no difference if the deleted edge is freely edge, and the two point of the deleted edge as start point to searching an augmenting path.

VI. CONCLUSION

A matching algorithm base on DFS is the simpler. The time efficiency of the algorithm is the same order of magnitude with the graph traverse. The worst is $O(|V||E|)$. When the point or the edge of the graph dynamic increase or decrease, the algorithm can quickly adjust the maximum matching.

ACKNOWLEDGMENT

Projects: Hainan university research start fund, No.Kyqd1229. Hainan province natural science foundation, No. 61353.

Corresponding author: Sheng Zhong

REFERENCES

[1] H. W. Kuhn, "The Hungarian method foe the assignment problem", Naval Res. Logist. Quart. 2(1955), pp.83-97.

[2] M. Hall, "An algorithm for distinct representatives", Amer. Math. Monthly, 1956, pp. 716-717.

[3] J.Edmonds, "Maximum matching and a polyhedron with(0,1) vertices", J.Res. Nat. Bur. Standards Seet. B 69B(1965), pp.125-130.

[4] J. Edmonds, Paths, "Trees and Flowers", Canadian J. Math., 17(1965), pp.449-467.

[5] S. Micali, V.Vazirani, "An O(m) algorithm for finding maximum matchings in general graphs", In Proe. 21st. Symp. Foundations of Computing, 1980, pp. 17-27.

[6] P. Vaidya, "Geometry helps in matching", In Proe. 20th ACM Symp. Theory of Computing, (1988), pp. 422-425.

[7] H. N. Gabow, "An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs", J.ACM, 1976,V0l.23, pp. 22l-234.

[8] M. L. Bajinski, "Labelling to Obtain a Maximum Matching in Combina- torial Mathematics and Its Applications", (R. C. Bose and T. A. Dowling, Eds.), Univ. North Carolina Press, Chappel Hill, N. C., 1967, pp. 585-602.

[9] D. Witzgall and C. T. Zahn, Jr., "Modification of Edmonds' Algorithm for Maximum Matching of Graphs" J. Res. Nat. Bur. std., 1965,Vol. 69B, pp.91-98.

[10] T. Kameda and I. munro, "A O(|V| × |E|) Algorithm for MaximumMatching of Graph Computing", 1974,V0l. 12, pp. 91-98