Breadth-First Search of Graphs

Analysis of Algorithms

Prepared by
John Reif, Ph.D.
Distinguished Professor of Computer Science
Duke University
Applications of Breadth-First Search of Graphs

a) Single Source Shortest Path

b) Graph Matching
Reading on Breadth-First Search of Graphs

- Reading Selection:
  - CLR, Chapter 24
Breadth-First Search Algorithm Input

\textit{input} : undirected graph } G = (V,E) \text{ with root } r \in V
Breadth-First Search Algorithm

initialize: \( L \leftarrow 0 \)

for each \( v \in V \) do visit(v) \( \leftarrow \) false

LEVEL(0) \( \leftarrow \{r\}; \) visit (r) \( \leftarrow \) true

while LEVEL(L) \( \neq \{\} \) do

begin

LEVEL(L+1) \( \leftarrow \{\} \)

for each \( v \in \text{LEVEL}(L) \) do

begin

for each \( \{v,u\} \in E \) s.t. not visit(u)

\( \text{do} \)

add u to LEVEL(L+1)

visit (u) \( \leftarrow \) true

\text{od}

end

L \( \leftarrow L + 1 \)

end

Breadth-First Search Algorithm Output

output : LEVEL(0), LEVEL(1), ..., LEVEL(L-1)
        O(|V|+|E|) time cost
Edges in Breadth-First Search

- All edges \( \{u,v\} \in E \) have level distance \( \leq 1 \)

Example

```
root r
```

```
LEVEL(0)
LEVEL(1)
LEVEL(2)
```
Breadth-First Search Tree

- Breadth First Search Tree \( T \)

Root \( r \)

LEVEL(0) = \{1\}
LEVEL(1) = \{2, 3, 4, 5\}
LEVEL(2) = \{6, 7, 8\}
Single Source Shortest Paths Problem

Input: digraph $G = (V,E)$ with root $r \in V$
weighting $d: E \rightarrow$ positive reals

Problem: For each vertex $v$, determine $D(v) = \text{min length path from root } r \text{ to } v$
Dijkstra’s Algorithm for Single Source Shortest Paths

initialize:

\[ Q \leftarrow \{\} \]

\[ \text{for each } v \in V-\{r\} \text{ do } D(v) \leftarrow \infty \]

\[ D(r) \leftarrow 0 \]

until no change do

choose a vertex \( u \in V-Q \)

with minimum \( D(u) \)

add \( u \) to \( Q \)

\[ \text{for each } (u,v) \in E \text{ s.t. } v \in V-Q \text{ do} \]

\[ D(v) \leftarrow \min (D(v), D(u) + d(u,v)) \]

output:

\[ \forall v \in V \]

\[ D(v) = \text{weight of min. path from } r \text{ to } v \]
Example Single Source Shortest Paths Problem

- example

![Diagram of a graph with labeled edges and a root node labeled r.](image-url)
# Example Execution of Dijkstra’s Algorithm

<table>
<thead>
<tr>
<th>Q</th>
<th>u</th>
<th>D(1)</th>
<th>D(2)</th>
<th>D(3)</th>
<th>D(4)</th>
<th>D(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>{1}</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>∞</td>
<td>100</td>
</tr>
<tr>
<td>{1,2}</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>{1,2,3}</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>{1,2,3,4}</td>
<td>5</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>90</td>
</tr>
</tbody>
</table>
Proof of Dijkstra’s Algorithm

• Use induction hypothesis:

\[ \begin{align*}
\text{(1)} & \quad \forall v \in V, \\
& \quad D(v) \text{ is weight of the minimum cost of path } p \text{ from } r \text{ to } v, \text{ where } p \text{ visits only vertices of } Q \cup \{v\} \\
\text{(2)} & \quad \forall v \in Q, \\
& \quad D(v) = \text{minimum cost path from } r \text{ to } v \\
& \quad \text{basis } D(r) = 0 \text{ for } Q = \{r\}\end{align*} \]
induction step

if $D(u)$ is minimum for all $u \in V-Q$

then claim:

(1) $D(u)$ is minimum cost of path from $r$ to $u$ in $G$
suppose not: then path $p$ with
weight $< D(u)$ and such that $p$ visits
a vertex $w \in V-(Q \cup \{u\})$. Then
$D(w) < D(u)$, contradiction.

(2) is satisfied by

$$D(v) = \min_{(u,v) \in E} (D(v), D(u) + d(u,v))$$

for all $v \in Q \cup \{u\}$
Time Cost of Dijkstra’s Algorithm on a RAM Model

- Time cost: per iteration

\[ \begin{align*}
&- O(\log|V|) \text{ to find } u \in V-Q \\
&\quad \text{with min } D(u) \\
&- O(\text{degree}(u)) \text{ to update weights}
\end{align*} \]

- Since there are $|V|$ iterations,

Total Time $O\left( |V| \left( \log |V| \right) + |E| \right)$
Graph Matching

- Graph $G = (V,E)$
- Graph Matching $M$ is a subset of $E$
  - if $e_1, e_2$ distinct edges in $M$
  - Then they have no vertex in common

Vertex $v$ is matched if $v$ is in an edge of $M$
Graph Matching Problem

Graph Matching Problem:
    Find a maximum size matching

- Suppose:
  - $G = (V,E)$ has matching $M$

Goal:
- find a larger matching
Augmenting Path in $G$ given Graph Matching $M$

- An augmenting path $p = (e_1, e_2, ..., e_k)$

\[
\text{require} \left\{ \begin{array}{l}
\text{begins and ends at unmatched vertices} \\
e_1, e_3, e_5, ..., e_k \in E-M \\
e_2, e_4, ..., e_{k-1} \in M
\end{array} \right. 
\]
Graph Matching (cont’d)

• Initial matching $M$ in $G$

```
    1 -- 2 -- 3 -- 4
    |     |     |     |
    5 -- 6 -- 7 -- 8
```

- $|M| = 2$

• Augmenting path
  $p = ((5,2), (2,6), (6,4), (4,7), (7,3))$
Graph Matching (cont’d)

- New matching $M' = P \oplus M = (P \cup M) - (P \cap M)$

$$|P \oplus M| = 3$$
Characterization of a Maximum Graph Matching via Lack of Augmented Path

- **Theorem**
  
  M is *maximum* matching  
  iff there is *no* augmenting path  
  relative to M
Proof of Characterization of Maximum Graph Matching (cont’d)

- Proof
  
  (1) If $M$ is smaller matching and $p$ is an augmenting path for $M$,
  
  then $M \oplus P$ is a matching size $> |M|$

  (2) If $M$, $M'$ are matchings with
  
  $|M| < |M'|$
Claim: \( M \oplus M' \) contains an augmenting path for \( M \).

Proof

- The graph \( G' = (V, M \oplus M') \) has only paths with edges alternating between \( M \) and \( M' \).
- Repeatedly delete a cycle in \( G' \) (with equal number of edges in \( M, M' \))
- Since \( |M| < |M'| \) must eventually get augmenting path remaining for \( M \).
Maximum Matching Algorithm

- **Algorithm**

\[ \text{input} \quad \text{graph } G = (V,E) \]

[1] \( M \leftarrow \{ \} \)

[2] \text{while} \quad \text{there exists an augmenting path } p \text{ relative to } M

\[ \text{do} \quad M \leftarrow M \oplus P \]

[3] \text{output} \quad \text{maximum matching } M
Maximum Matching (cont’d)

• Remaining problem:
  Find augmenting path

• Assume
  \textit{weighting} \ d: E \to \mathbb{R}^+ = \text{pos. reals}
Maximum Weighted Matching Algorithm

- Assume
  - \textit{weighting} \( d : E \rightarrow \mathbb{R}^+ \) = positive reals
- Theorem
  - Let \( M \) be maximum weight among matchings of size \(|M|\).
  - Let \( P \) be an augmenting path for \( M \) of maximum weight.
  - Then matching \( M \oplus P \) is of maximum weight among matchings of size \(|M|+1\).
Proof of Maximum Weighted Matching Algorithm

- **Proof**
  - Let $M'$ be any maximum weight matching of size $|M| + 1$.
  - Consider the graph $G' = (V, M \oplus M')$.
  - Then the maximum weight augmenting path $p$ in $G'$ gives a matching $M \oplus P$ of the same weight as $M'$. 
Bipartite Graph

- Bipartite Graph \( G = (V, E) \)

\[
V = V_1 \cup V_2, \quad V_1 \cap V_2 = \emptyset
\]

\( E \) is a subset of \( \{ \{u,v\} \mid u \in V_1, v \in V_2 \} \)

Graph representation:
- \( V_1 \)
- \( V_2 \)
- Edges connecting \( V_1 \) and \( V_2 \)
Breadth-First Search Algorithm for Augmented Path

- Assume $G$ is a bipartite graph with matching $M$.
- Use Breadth-First Search:

  $\text{LEVEL}(0) =$ some unmatched vertex $r$

  For odd $L > 0$,
  $$\text{LEVEL}(L) = \{u \mid \{v,u\} \in E - M \text{ when } v \in \text{LEVEL}(L - 1) \text{ and when } u \text{ in no lower level}\}$$

  For even $L > 0$,
  $$\text{LEVEL}(L) = \{u \mid \{v,u\} \in M \text{ when } v \in \text{LEVEL}(L - 1) \text{ and } u \text{ in no lower level}\}$$
Proof of Breadth-First Search Algorithm for Augmented Path

• Cases
  (1) If for some odd $L > 0$, $\text{LEVEL}(L)$ contains an unmatched vertex $u$ then the Breadth First Search tree $T$ has an augmenting path from $r$ to $u$
  (2) Otherwise no augmenting path exists, so $M$ is maximal.
Finding a Maximal Matching in a Bipartite Graph

• **Theorem**
  If $G = (V,E)$ is a bipartite graph, then the maximum matching can be constructed in $O(|V|(|V| + |E|))$ time.

• **Proof**
  Each stage requires $O(|V| + |E|)$ time for Breadth First Search construction of augmenting path.
Generalizations of Matching Algorithm

- **Generalizations:**

  1. Compute Edge Weighted Maximum Matching
  2. Edmonds gives a polynomial time algorithm for maximum matching of any graph
Computing Augmented Paths in General Graphs

- Let $M$ be matching in general graph $G$
- Fix starting vertex $r$ to be an unmatched vertex

Let vertex $v \in V$ be *even* if

$\exists$ even length augmenting path from $r$ to $v$

and *odd* if

$\exists$ odd length augmenting path from $r$ to $v$. 
Why Algorithm for Augmented Paths in Bipartite Graphs does not work for General Graphs

Case
G is bipartite
⇒ no vertex is both even and odd

Case
G is not bipartite
⇒ some vertices may be both even and odd!
Edmond’s Algorithm for Augmented Paths in General Graphs

P is *augmenting path* from r to v

STEM is subpath of p from r to v

BLOSSOM is subpath of p from v to w plus edge \{w,v\}

Base w even vertex

Shrink Blossom
Blossom Shrinking Maintains the Existence of Augmented Paths

• *Theorem*
  
  If $G'$ is formed from $G$ by shrinking of blossom $B$, then
  
  $G$ contains an augmenting path iff $G'$ does.
Proof of Blossom Shrinking

- **Proof**
  1. If $G'$ contains an augmenting path $p$, then if $p$ visits blossom $B$ we can insert an augmenting subpath $p'$ within blossom into $p$ to get a new **augmenting path** for $G$.
  2. If $G$ contains an augmenting path, then apply Edmond’s blossom shrinking algorithm to find an **augmenting path in $G'$**.
Edmond’s Blossom Shrinking Algorithm

*input* Graph $G = (V,E)$ with matching $M$

*initialization* $\overrightarrow{E} = \{(v,w), (w,v) | \{v,w\} \in E\}$

**Main Ideas of Edmond’s algorithm:**

- The algorithm incrementally constructs a forest of trees whose paths are partial augmenting paths.
- If a cycle is formed, contract it to a vertex.
- Try to link two partial augmenting paths of distinct trees to form a full augmenting path.
Edmond’s Blossom Shrinking Algorithm (cont’d)

- Note: We will let \( P(v) = \) parent of vertex \( v \)

\[
\begin{align*}
[0] & \text{ for each unmatched vertex } v \in V \\
& \quad \text{do label } v \text{ as "even"} \\
[1] & \text{ for each matched } v \in V \\
& \quad \text{do label } v \text{ "unreached" set } p(v) = \text{null} \\
& \quad \text{if } v \text{ is matched edge } \{v,w\} \\
& \quad \quad \text{then } \text{mate } (v) \leftarrow w \\
& \quad \text{end if} \\
& \quad \text{od}
\end{align*}
\]
Main Loop

- Edmond’s Main Loop:

Choose an unexplored edge \((v,w) \in \vec{E}\)

where \(v\) is "even"

(if none exists, then terminate and output current matching \(M\), since there is no augmenting path)
Main Loop (cont’d)

- **Case 1**  
  if w is “odd” then do nothing.

- **Case 2**  
  if w is “unreached” and matched then set w “odd” and set mate (w) “even”

Set $P(w) \leftarrow v$, $P(\text{mate}(w)) \leftarrow w$
Main Loop (cont’d)

- Case 3 \( \text{if w is “even” and v, w are in distinct trees } T, T' \text{ then output augmenting path p from root of } T \text{ to } v, \text{ through } \{v, w\}, \text{ in } T' \text{ to root.} \)

![Diagram showing two trees T and T', with unmatched root, v even, and w even. There is an unmatched edge between T and T'.]
Main Loop (cont’d)

• Case 4  \( w \) is “even” and \( v, w \) in same tree \( T \) then \( \{v, w\} \) forms a blossom \( B \) containing all vertices which are both (i) a descendant of \( \text{LCA}(v, w) \) and (ii) an ancestor of \( v \) or \( w \) where \( \text{LCA}(v, w) = \) least common ancestor of \( v, w \) in \( T \)
Main Loop (cont’d)

- Shrink all vertices of B to a single vertex b. Define \( p(b) = p(LCA(v,w)) \) and \( p(x) = b \) for all \( x \in B \)
Proof Edmond’s blossom-shrinking algorithm succeeds

• **Lemma**

  Edmond’s blossom-shrinking algorithm succeeds iff

  \[ \exists \text{ an augmenting path in } G \]

• **Proof**

  Uses an induction on blossom shrinking stages
Time Bounds for Matching in General Graphs

- Edmond’s blossom-shrinking algorithm costs time $O(n^4)$

- [Gabow and Tarjan] implement in time $O(nm)$ all $O(n)$ stages of matching algorithm taking $O(m)$ time per stage for blossom shrinking

- [Micali and Vazirani] using network flow to find augmented paths and reduce time to $O(n^{1/2}m)$ for unweighted matching in general graphs