Dynamic Matching Algorithms in Practice

presented by Martin Grösbacher

University of Salzburg
Department of Computer Science

paper by Monika Henzinger, Shahbaz Khan, Richard Paul, Christian Schulz

Motivation

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point
- Applications sometimes require matchings with certain properties (maximal, maximum, maximal weight, etc.)

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point
- Applications sometimes require matchings with certain properties (maximal, maximum, maximal weight, etc.)
- Matchings can be computed in polynomial time

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point
- Applications sometimes require matchings with certain properties (maximal, maximum, maximal weight, etc.)
- Matchings can be computed in polynomial time
- If underlying graph often changes (i.e. dynamicity) computing new matches from scratch every time can still be an expensive task!

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point
- Applications sometimes require matchings with certain properties (maximal, maximum, maximal weight, etc.)
- Matchings can be computed in polynomial time
- If underlying graph often changes (i.e. dynamicity) computing new matches from scratch every time can still be an expensive task!
- New fully dynamic matching algorithms have been developed recently

Motivation

- A matching $\mathcal{M}$ is a subset of edges in a graph, such that no two elements of $\mathcal{M}$ share a common end point
- Applications sometimes require matchings with certain properties (maximal, maximum, maximal weight, etc.)
- Matchings can be computed in polynomial time
- If underlying graph often changes (i.e. dynamicity) computing new matches from scratch every time can still be an expensive task!
- New fully dynamic matching algorithms have been developed recently
- Bridge the gap between theory and practice by testing out and comparing these algorithms

Preliminaries

Preliminaries

- Let $G = (V, E)$ be an undirected graph without parallel edges or self-loops

Preliminaries

- Let $G = (V, E)$ be an undirected graph without parallel edges or self-loops
- $|V| = n, |E| = m$

Preliminaries

- Let $G = (V, E)$ be an undirected graph without parallel edges or self-loops
- $|V| = n, |E| = m$
- $N(v) := \{u : \{v, u\} \in E\}$ denotes the set of neighbours of v

Preliminaries

- Let $G = (V, E)$ be an undirected graph without parallel edges or self-loops
- $|V| = n, |E| = m$
- $N(v) := \{u : \{v, u\} \in E\}$ denotes the set of neighbours of v
- $deg(v) := |N(v)|$

Preliminaries

- Let $G = (V, E)$ be an undirected graph without parallel edges or self-loops
- $|V| = n, |E| = m$
- $N(v) := \{u : \{v, u\} \in E\}$ denotes the set of neighbours of v
- $deg(v) := |N(v)|$

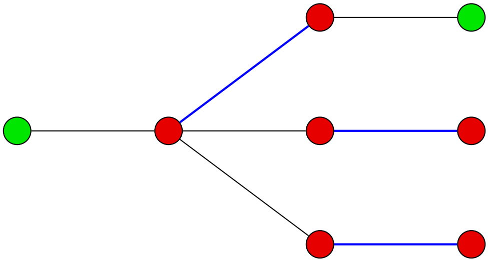
Definition (Matching)

A set of edges $\mathcal{M} \subseteq E$ such that for all pairs of edges $((u, v), (r, s)) \in \mathcal{M} : r, s, u, v$ are distinct.

Preliminaries

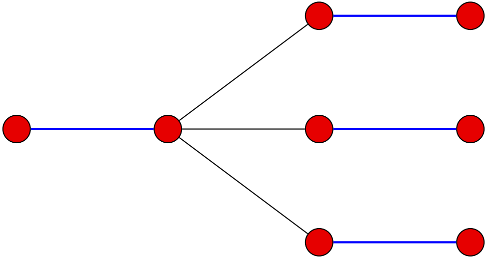
Preliminaries

- A matching is called *maximal*, if there is no edge in E that can be added to $\mathcal{M}$



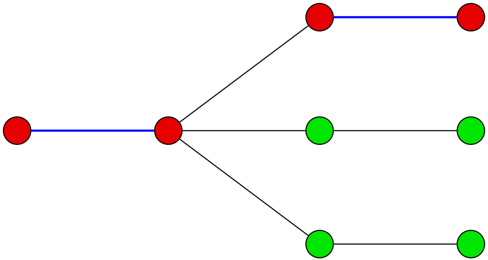
Preliminaries

- A *maximum matching* $\mathcal{M}_{\text{opt}}$ is a maximal matching that contains the largest number of possible edges



Preliminaries

- An α-approximate maximum matching is a matching that contains at least $\frac{|\mathcal{M}_{\text{opt}}|}{\alpha}$ edges



Preliminaries

Preliminaries

- A vertex is called *free*, if it is not incident to an edge $(u, v) \in \mathcal{M}$ and *matched* otherwise

Preliminaries

- A vertex is called *free*, if it is not incident to an edge $(u, v) \in \mathcal{M}$ and *matched* otherwise
- For a matched vertex u with $(u, v) \in \mathcal{M}$ we call v the *mate* of u

Preliminaries

- A vertex is called *free*, if it is not incident to an edge $(u, v) \in \mathcal{M}$ and *matched* otherwise
- For a matched vertex u with $(u, v) \in \mathcal{M}$ we call v the *mate* of u

Definition (Augmenting Path)

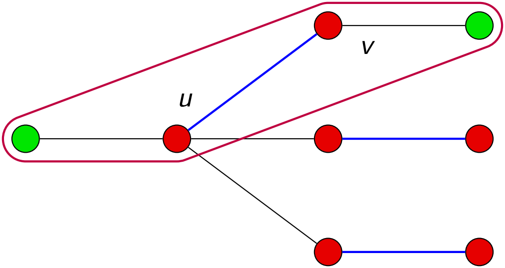
An augmenting path is a cycle-free path in G that starts and ends on a free vertex and where edges alternate from $\mathcal{M}$ with edges from $E \setminus \mathcal{M}$

Preliminaries

- A vertex is called *free*, if it is not incident to an edge $(u, v) \in \mathcal{M}$ and *matched* otherwise
- For a matched vertex u with $(u, v) \in \mathcal{M}$ we call v the *mate* of u

Definition (Augmenting Path)

An augmenting path is a cycle-free path in G that starts and ends on a free vertex and where edges alternate from $\mathcal{M}$ with edges from $E \setminus \mathcal{M}$

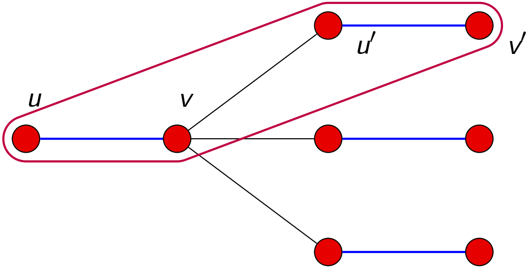


Preliminaries

- A vertex is called *free*, if it is not incident to an edge $(u, v) \in \mathcal{M}$ and *matched* otherwise
- For a matched vertex u with $(u, v) \in \mathcal{M}$ we call v the *mate* of u

Definition (Augmenting Path)

An augmenting path is a cycle-free path in G that starts and ends on a free vertex and where edges alternate from $\mathcal{M}$ with edges from $E \setminus \mathcal{M}$



Preliminaries

Preliminaries

- Any matching without augmenting paths is a maximum matching (Theorem of Berge)

Preliminaries

- Any matching without augmenting paths is a maximum matching (Theorem of Berge)
- Any matching without augmenting paths of length at most $2k-3$ is a $\frac{k}{k-1}$-approximate maximum matching (Hopcroft and Karp)

Preliminaries

- Any matching without augmenting paths is a maximum matching (Theorem of Berge)
- Any matching without augmenting paths of length at most $2k-3$ is a $\frac{k}{k-1}$-approximate maximum matching (Hopcroft and Karp)
- Hence, a maximal matching without augmenting paths of length one is a 2-approximate maximum matching

Preliminaries

- Any matching without augmenting paths is a maximum matching (Theorem of Berge)
- Any matching without augmenting paths of length at most $2k-3$ is a $\frac{k}{k-1}$-approximate maximum matching (Hopcroft and Karp)
- Hence, a maximal matching without augmenting paths of length one is a 2-approximate maximum matching
- In the following, Δ denotes the largest degree that can be found in any state of the dynamic graph

Algorithms

Algorithms

1. Random Walk-based algorithm:
   - Maintains $(1+\epsilon)$-approximate maximum matching w.h.p.
   - Performs random walks trying to find augmenting paths
   - Update time: $O(\frac{\Delta^{\frac{2}{\epsilon}-1}\log(n)}{\epsilon})$

Algorithms

1. Random Walk-based algorithm:
   - Maintains $(1+\epsilon)$-approximate maximum matching w.h.p.
   - Performs random walks trying to find augmenting paths
   - Update time: $O(\frac{\Delta^{\frac{2}{\epsilon}-1}\log(n)}{\epsilon})$
2. Blossom-based algorithm (deterministic):
   - Maintains $(1+\epsilon)$-approximate maximum matching
   - Performs depth bounded augmenting path search
   - Update time: $O(\Delta^{\frac{2}{\epsilon}-1})$

Algorithms

1. Random Walk-based algorithm:
   - Maintains $(1+\epsilon)$-approximate maximum matching w.h.p.
   - Performs random walks trying to find augmenting paths
   - Update time: $O(\frac{\Delta^{\frac{2}{\epsilon}-1}\log(n)}{\epsilon})$
2. Blossom-based algorithm (deterministic):
   - Maintains $(1+\epsilon)$-approximate maximum matching
   - Performs depth bounded augmenting path search
   - Update time: $O(\Delta^{\frac{2}{\epsilon}-1})$
3. Baswana, Gupta & Sen (randomized):
   - Maintains 2-approximate maximum matching w.h.p.
   - Vertices on multiple levels, edges are *owned* by vertices
   - Update time: $O(\log(n)^k)$ (amortized)

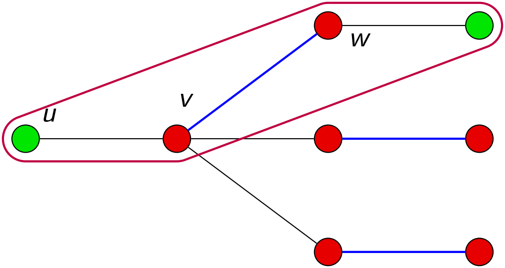
Algorithms

1. Random Walk-based algorithm:
   - Maintains $(1+\epsilon)$-approximate maximum matching w.h.p.
   - Performs random walks trying to find augmenting paths
   - Update time: $O(\frac{\Delta^{\frac{2}{\epsilon}-1}\log(n)}{\epsilon})$
2. Blossom-based algorithm (deterministic):
   - Maintains $(1+\epsilon)$-approximate maximum matching
   - Performs depth bounded augmenting path search
   - Update time: $O(\Delta^{\frac{2}{\epsilon}-1})$
3. Baswana, Gupta & Sen (randomized):
   - Maintains 2-approximate maximum matching w.h.p.
   - Vertices on multiple levels, edges are *owned* by vertices
   - Update time: $O(\log(n)^k)$ (amortized)
4. Neiman & Solomon (deterministic):
   - Maintains $(\frac{3}{2})$-approximate maximum matching
   - Uses concept of high degree/low degree vertices
   - Update time: $O(\sqrt(m))$ (worst case)

Random Walk-based Algorithm

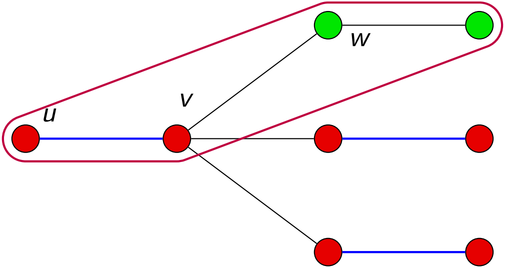
Random Walk-based Algorithm

1. Pick a free vertex u
2. Randomly choose neighbour v of u
3. If v is free: match (u, v) and stop walk
4. Else: unmatch $(v, mate(v))$ and match (u, v)



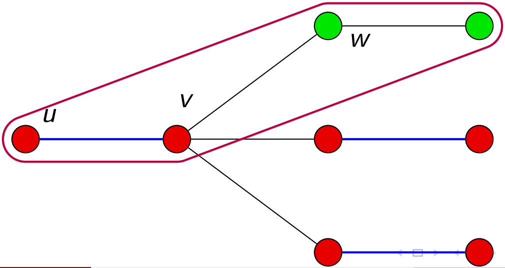
Random Walk-based Algorithm

- Now *mate*(v) is free, continue walk from there until $O(\frac{1}{\epsilon})$ steps
- Length of the walk is an important parameter
- Update time for a single walk: $O(\frac{1}{\epsilon})$



Random Walk-based Algorithm

- By itself this does not even guarantee a maximal matching!
- Fixing by undoing all changes
- Alternative: Δ-Settling: Scan through neighbours of visited vertices to find a free vertex
- Stops if either free vertex found or after $\frac{1}{\epsilon}$ steps
- If Random Walk was unsuccessful, try to match the last vertex touched by scanning all its neighbours
- Requires $O(\Delta)$ additional time per visited vertex



Random Walk-based Algorithm

Edge Insertion:

Random Walk-based Algorithm

Edge Insertion:

1. If both endpoints u, v are free, match (u, v)

Random Walk-based Algorithm

Edge Insertion:

1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing

Random Walk-based Algorithm

Edge Insertion:

1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing
3. Else: Unmatch $v, mate(v) = w$, match (u, v) and start Random Walk from w

Random Walk-based Algorithm

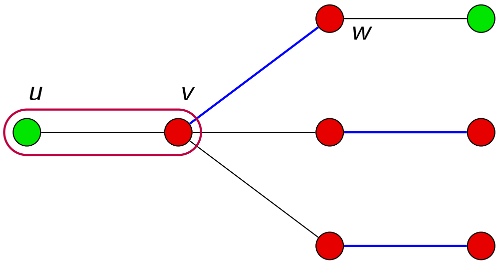
Edge Insertion:

1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing
3. Else: Unmatch $v, mate(v) = w$, match (u, v) and start Random Walk from w
4. If Random Walk is unsuccessful, undo all changes and restore matching to the state before unmatching v, w

Random Walk-based Algorithm

Edge Insertion:

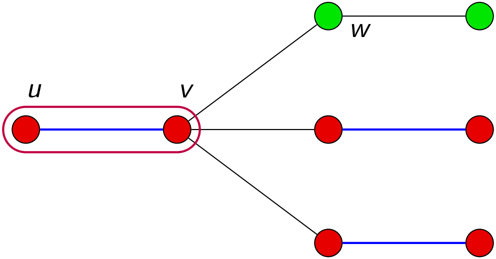
1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing
3. Else: Unmatch $v, mate(v) = w$, match (u, v) and start Random Walk from w
4. If Random Walk is unsuccessful, undo all changes and restore matching to the state before unmatching v, w



Random Walk-based Algorithm

Edge Insertion:

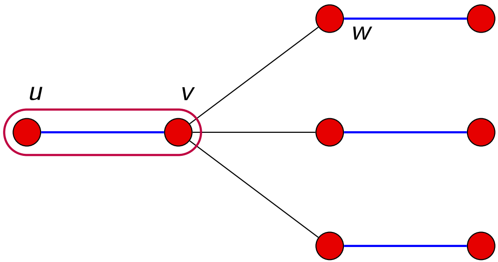
1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing
3. Else: Unmatch $v, mate(v) = w$, match (u, v) and start Random Walk from w
4. If Random Walk is unsuccessful, undo all changes and restore matching to the state before unmatching v, w



Random Walk-based Algorithm

Edge Insertion:

1. If both endpoints u, v are free, match (u, v)
2. Else if both are matched: do nothing
3. Else: Unmatch $v, mate(v) = w$, match (u, v) and start Random Walk from w
4. If Random Walk is unsuccessful, undo all changes and restore matching to the state before unmatching v, w



Random Walk-based Algorithm

Edge Deletion:

Random Walk-based Algorithm

Edge Deletion:

1. If (u, v) was unmatched: do nothing

Random Walk-based Algorithm

Edge Deletion:

1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)

Random Walk-based Algorithm

Edge Deletion:

1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)
3. If u and/or v cannot be matched and $\mathcal{M}$ was maximal before deletion: $\mathcal{M}$ is still maximal

Random Walk-based Algorithm

Edge Deletion:

1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)
3. If u and/or v cannot be matched and $\mathcal{M}$ was maximal before deletion: $\mathcal{M}$ is still maximal
4. However: free vertices may be the start of an augmenting path!

Random Walk-based Algorithm

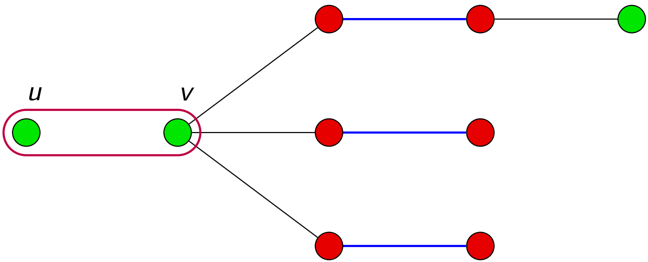
Edge Deletion:

1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)
3. If u and/or v cannot be matched and $\mathcal{M}$ was maximal before deletion: $\mathcal{M}$ is still maximal
4. However: free vertices may be the start of an augmenting path!
5. Start Random Walk from u and/or v

Random Walk-based Algorithm

Edge Deletion:

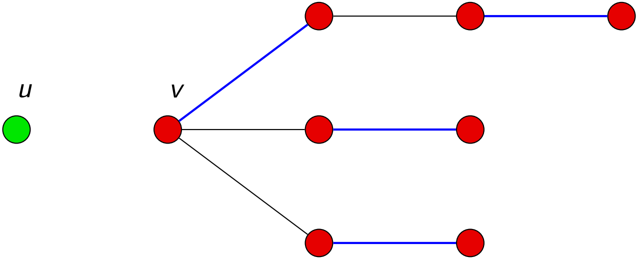
1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)
3. If u and/or v cannot be matched and $\mathcal{M}$ was maximal before deletion: $\mathcal{M}$ is still maximal
4. However: free vertices may be the start of an augmenting path!
5. Start Random Walk from u and/or v



Random Walk-based Algorithm

Edge Deletion:

1. If (u, v) was unmatched: do nothing
2. Else: Scan neighbours of u, v, try to match them (takes $O(\Delta)$ time)
3. If u and/or v cannot be matched and $\mathcal{M}$ was maximal before deletion: $\mathcal{M}$ is still maximal
4. However: free vertices may be the start of an augmenting path!
5. Start Random Walk from u and/or v



Random Walk-based Algorithm

Analysis:

Random Walk-based Algorithm

Analysis:

1. Maintains $(1+\epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often

Random Walk-based Algorithm

Analysis:

1. Maintains $(1 + \epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often
2. Path Length: $\frac{2}{\epsilon} - 1$, Repetitions: $\Delta^{\frac{2}{\epsilon} - 1} \log(n)$

Random Walk-based Algorithm

Analysis:

1. Maintains $(1+\epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often
2. Path Length: $\frac{2}{\epsilon}-1$, Repetitions: $\Delta^{\frac{2}{\epsilon}-1}\log(n)$
3. No augmenting path $\leq \frac{2}{\epsilon}-1 = 2(\frac{1}{\epsilon}+1)-3$, $k = \frac{1}{\epsilon}+1 \overset{H.\&K.}{\Longrightarrow} \mathcal{M}$ is a $(1+\epsilon)$-approximation of $\mathcal{M}_{\text{opt}}$

Random Walk-based Algorithm

Analysis:

1. Maintains $(1+\epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often
2. Path Length: $\frac{2}{\epsilon} - 1$, Repetitions: $\Delta^{\frac{2}{\epsilon}-1}\log(n)$
3. No augmenting path $\leq \frac{2}{\epsilon} - 1 = 2(\frac{1}{\epsilon}+1) - 3$, $k = \frac{1}{\epsilon} + 1 \overset{H.\&K.}{\Longrightarrow} \mathcal{M}$ is a $(1+\epsilon)$-approximation of $\mathcal{M}_{\mathrm{opt}}$
4. If there is such a path, the probability of finding it is $\geq (\frac{1}{\Delta})^{\frac{2}{\epsilon}-1}$

Random Walk-based Algorithm

Analysis:

1. Maintains $(1+\epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often
2. Path Length: $\frac{2}{\epsilon}-1$, Repetitions: $\Delta^{\frac{2}{\epsilon}-1}\log(n)$
3. No augmenting path $\leq \frac{2}{\epsilon}-1 = 2(\frac{1}{\epsilon}+1)-3$, $k=\frac{1}{\epsilon}+1 \overset{H.\&K.}{\Longrightarrow} \mathcal{M}$ is a $(1+\epsilon)$-approximation of $\mathcal{M}_{\mathrm{opt}}$
4. If there is such a path, the probability of finding it is $\geq (\frac{1}{\Delta})^{\frac{2}{\epsilon}-1}$
5. Probability that λ walks do not find such a path: $\leq (1-\frac{1}{\Delta^{\frac{2}{\epsilon}-1}})^{\lambda}$

Random Walk-based Algorithm

Analysis:

1. Maintains $(1+\epsilon)$ approximation if Random Walks are of appropriate length and repeated sufficiently often
2. Path Length: $\frac{2}{\epsilon}-1$, Repetitions: $\Delta^{\frac{2}{\epsilon}-1}\log(n)$
3. No augmenting path $\leq \frac{2}{\epsilon}-1 = 2(\frac{1}{\epsilon}+1)-3$, $k=\frac{1}{\epsilon}+1 \overset{H.\&K.}{\Longrightarrow} \mathcal{M}$ is a $(1+\epsilon)$-approximation of $\mathcal{M}_{\text{opt}}$
4. If there is such a path, the probability of finding it is $\geq (\frac{1}{\Delta})^{\frac{2}{\epsilon}-1}$
5. Probability that λ walks do not find such a path: $\leq (1-\frac{1}{\Delta^{\frac{2}{\epsilon}-1}})^{\lambda}$
6. Hence if $\lambda \geq \Delta^{\frac{2}{\epsilon}-1}\log(n)$:

$$(1-\frac{1}{\Delta^{\frac{2}{\epsilon}-1}})^{\Delta^{\frac{2}{\epsilon}-1}\log(n)} \leq e^{-\frac{1}{\Delta^{\frac{2}{\epsilon}-1}}\Delta^{\frac{2}{\epsilon}-1}\log(n)} = e^{-\log(n)} = \frac{1}{n}$$

Blossom-based Algorithm

Blossom-based Algorithm

- Theoretical bound for Random Walk-based Algorithm is fairly pessimistic

Blossom-based Algorithm

- Theoretical bound for Random Walk-based Algorithm is fairly pessimistic
- Stops after one augmenting path has been found which can be shorter

Blossom-based Algorithm

- Theoretical bound for Random Walk-based Algorithm is fairly pessimistic
- Stops after one augmenting path has been found which can be shorter
- Idea: Depth-bounded augmenting path search via BFS instead of Random Walks

Blossom-based Algorithm

- Theoretical bound for Random Walk-based Algorithm is fairly pessimistic
- Stops after one augmenting path has been found which can be shorter
- Idea: Depth-bounded augmenting path search via BFS instead of Random Walks
- One search stops as soon as augmenting path was found and has running time $\Theta(n' + m')$ where n', m' are the number of vertices and edges touched by the BFS

Blossom-based Algorithm

- Theoretical bound for Random Walk-based Algorithm is fairly pessimistic
- Stops after one augmenting path has been found which can be shorter
- Idea: Depth-bounded augmenting path search via BFS instead of Random Walks
- One search stops as soon as augmenting path was found and has running time $\Theta(n' + m')$ where n', m' are the number of vertices and edges touched by the BFS
- First BFS needs an additional $O(n + m)$ time to initialize the data structures, all others do book keeping of the changes they made and undo them afterwards

Blossom-based Algorithm

Edge insertion:

Blossom-based Algorithm

Edge insertion:

1. If u, v are free: match (u, v)

Blossom-based Algorithm

Edge insertion:

1. If u, v are free: match (u, v)
2. Else if only one of them is free: Start BFS from u

Blossom-based Algorithm

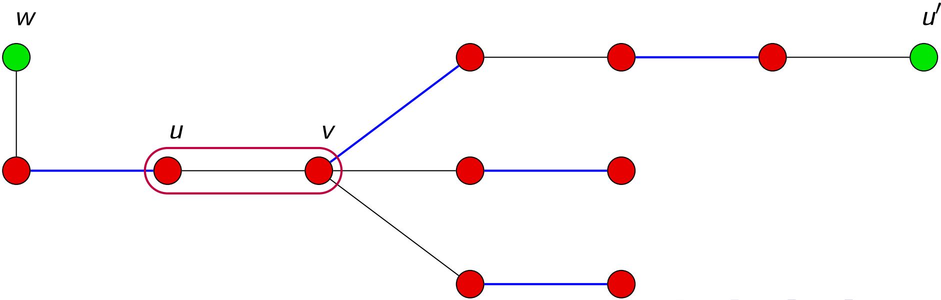
Edge insertion:

1. If u, v are free: match (u, v)
2. Else if only one of them is free: Start BFS from u
3. Else: Start BFS from u to find a free node w via an alternating path

Blossom-based Algorithm

Edge insertion:

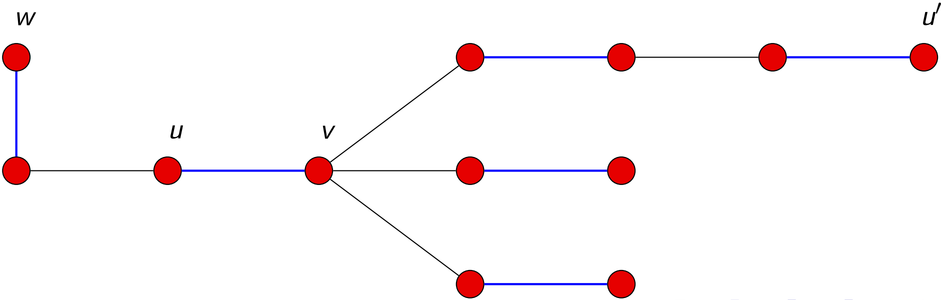
1. If u, v are free: match (u, v)
2. Else if only one of them is free: Start BFS from u
3. Else: Start BFS from u to find a free node w via an alternating path
4. Start another BFS from w to find augmenting path



Blossom-based Algorithm

Edge insertion:

1. If u, v are free: match (u, v)
2. Else if only one of them is free: Start BFS from u
3. Else: Start BFS from u to find a free node w via an alternating path
4. Start another BFS from w to find augmenting path



Blossom-based Algorithm

Optimizations:

Blossom-based Algorithm

Optimizations:

1. *unsafe*: In case both u and v are not free: do nothing

Blossom-based Algorithm

Optimizations:

1. *unsafe*: In case both u and v are not free: do nothing
2. *Lazy augmenting path search*: Start search from u only if at least $\frac{m'}{2}$ edges have been inserted or deleted since the last search from u or no search has been started

Blossom-based Algorithm

Optimizations:

1. *unsafe*: In case both u and v are not free: do nothing
2. *Lazy augmenting path search*: Start search from u only if at least $\frac{m'}{2}$ edges have been inserted or deleted since the last search from u or no search has been started
3. Depth-binding paths to length $\frac{2}{\epsilon} - 1$ ensures deterministic $(1 + \epsilon)$-approximate matching algorithm

Blossom-based Algorithm

Optimizations:

1. *unsafe*: In case both u and v are not free: do nothing
2. *Lazy augmenting path search*: Start search from u only if at least $\frac{m'}{2}$ edges have been inserted or deleted since the last search from u or no search has been started
3. Depth-binding paths to length $\frac{2}{\epsilon} - 1$ ensures deterministic $(1+\epsilon)$-approximate matching algorithm
4. Worst case complexity of optimum version: $O(n+m)$, bounded version: $O(\Delta^{\frac{2}{\epsilon}-1})$
5. Edge Deletions: Start BFS from any free endpoint u or v, combinable with LP and DB

Experimental Setup

Experimental Setup

1. Ten repetitions per instance, taking geometric mean

Experimental Setup

1. Ten repetitions per instance, taking geometric mean
2. Example graphs include static graphs as well as dynamic ones

Experimental Setup

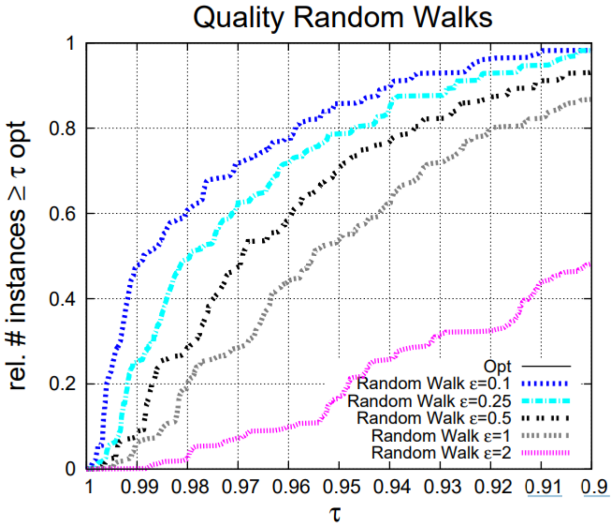
1. Ten repetitions per instance, taking geometric mean
2. Example graphs include static graphs as well as dynamic ones
3. Two types of experiments: start with empty (static graph) and do insertions only as well as real dynamic graphs

Experimental Setup

1. Ten repetitions per instance, taking geometric mean
2. Example graphs include static graphs as well as dynamic ones
3. Two types of experiments: start with empty (static graph) and do insertions only as well as real dynamic graphs
4. Most of the dynamic graph instances only use insertions, deletions are constructed by undoing insertions

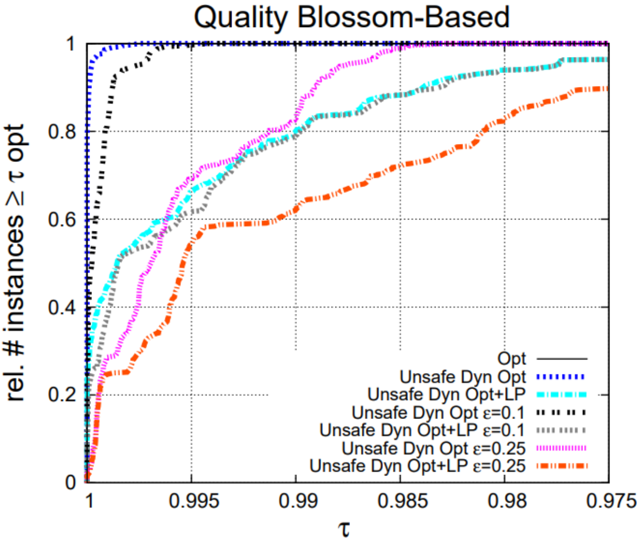
Experiments: Random Walk-based Algorithm

Experiments: Random Walk-based Algorithm



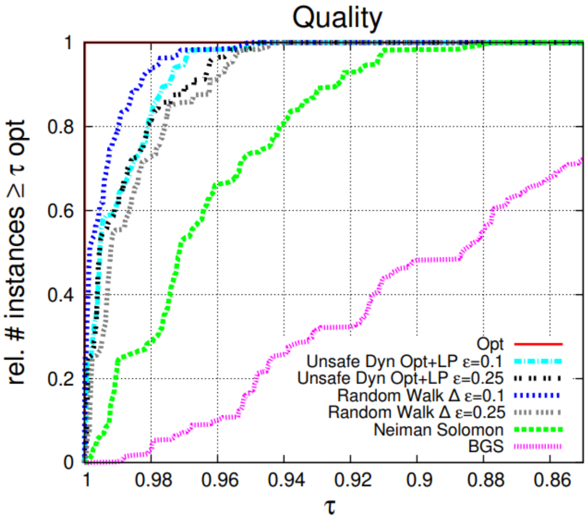
Experiments: Blossom-based Algorithm

Experiments: Blossom-based Algorithm



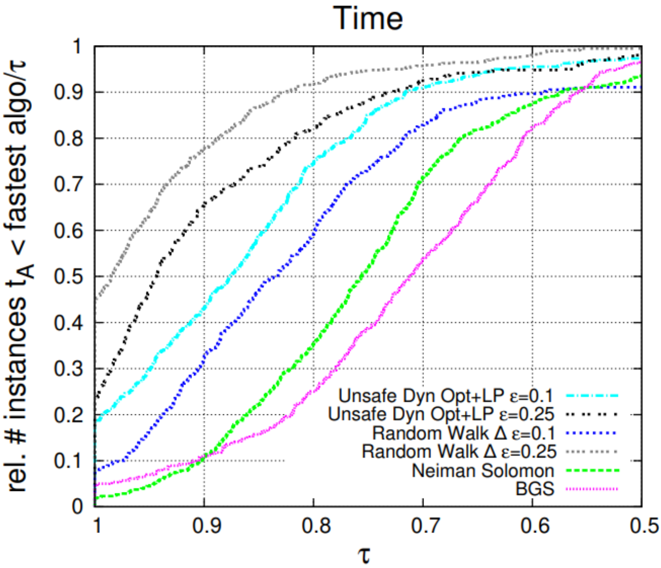
Experiments: Quality comparison of all algorithms

Experiments: Quality comparison of all algorithms



Experiments: Runtime comparison of all algorithms

Experiments: Runtime comparison of all algorithms



Conclusion

1. Maintaining optimum matchings can be done much more efficiently than the naive approach to compute matchings from scratch after every dynamic change in an unweighted graph
2. All approximative algorithms that we have seen are able to maintain near-optimum matchings in practice while being significantly faster
3. Random-Walks with Delta Settling enabled will be the method of choice in practice
4. Open questions: Weighted case, dynamic multilevel algorithms, parallelization potential, real world dynamic graph instances with both insertions and deletions

Thank you for your attention!