Streaming Algorithms for Matching Problems

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering

Fakultät für Informatik

TU Dortmund

Selected Topics on Combinatorial Optimization

Vienna Graduate School on Computational Optimization

Lectures 3+5 (Part 1+3), November 7+9, 2018

Outline

Literature for Data Stream Algorithms

- A. McGregor: Graph Stream Algorithms: A Survey, SIGMOD Rec, vol. 43 (1), ACM, 2014, 9–20
- M. Henzinger, P. Raghavan, S. Rajagopalan: Computing on Data Streams, in: External Memory Algorithms, American Mathematical Society, 1999, 107–118
- S. Muthukrishnan: Data Streams: Algorithms and Applications, Found. Trends Theor. Comput. Sci., 2005, 117–236

Specific open questions concerning data streams: `sublinear.info`

Literature for Matching Algorithms in the Streaming Model

- A. McGregor: Finding Graph matchings in Data Streams, APPROX'05/RANDOM'05, LNCS, 2005, 170–181
- C. Konrad, F. Magniez, C. Mathieu: Maximum matching in semi-streaming with few passes, APPROX/RANDOM 2012, 231–242 (also ArXiv:1112.0184v3, 2014)
- C. Konrad: A simple augmentation method for matchings with applications to streaming algorithms, MFCS 2018, LIPICS 74:1–74:16
- S. Eggert, L. Kliemann, P. Munstermann, A. Srivastav: Bipartite graph matchings in the semi-streaming model, Algorithmica 63 (1), 490–508, 2012
- L. Kliemann: Engineering a bipartite matching algorithm in the semi-streaming model, Algorithm Engineering, LNCS 9220, 2016, 352–378

Motivation for Data Stream Algorithms

- Web graph: 10^{10} nodes
- Human brain models: 10^{10} nodes
- IPv6 supports 2^{128} nodes

Classes of considered problems

- connectivity properties
- approximating graph distances (e.g., centrality indices in social networks, graph classification)
- frequency-counting of subgraphs (graph classification)
- approximate matchings (controllability of networks, assignment problems)

Data Stream Algorithms

- A data stream is a (very big) amount of data that cannot be stored, since available memory is much less than the stream size
- **Goal:** process data in data stream model

Origins in 70s but has become popular in last fifteen years

Focus on early work:

Process numerical data such as estimating quantiles, heavy hitters or the number of distinct elements in the stream

Current applications:

Routers (server logs) for network monitoring, satellites, facebook actions, user clicks, search queries, sensor networks aggregation, CERN data, neurons and synapses

Data Stream Model

Definition (Our data stream model)

We consider the following data stream model:

- the input stream $a_1, a_2, \ldots$ arrives sequentially, item by item
- for finite input streams: m elements from universe $0, 1, \ldots, n$

Goal: compute a function of the stream data, e.g., median, number of distinct elements, longest increasing sequence,...

Restrictions:

1. limited working memory (sublinear in size of input) measured in bits
2. access data sequentially
3. process each element quickly
4. number of passes P over the data stream

Data Stream Algorithms

Analysis of Data Stream Algorithms

- time complexity for each element
- overall time complexity
- space complexity

- restricted number of passes
- space-time tradeoffs
- maintaining a dynamic data structure but with only sublinear storage space
- techniques: approximation, sampling, statistics or sketching

Models of Computations

RAM Model of Computation

The Random Access Machine (RAM) denotes a model of computation for algorithm analysis in which we have

- each memory access takes exactly one time step
- each simple operation (e.g., $+, -, *, =, \mathit{if}, \mathit{call}$) takes exactly 1 time step
- we have as much memory as we need

This does no more apply for data streams.
Numbers need $\log n$ bits to be stored and processed.
$\rightarrow$ we measure bit complexity

Warming Up: Find max

Task

Find maximum element in a finite data stream of numbers: $a_1, a_2, \ldots, a_m$ from domain $\{1, 2, \ldots n\}$

Simple algorithm

- max $= 0$;
- for each i in $\{1, \ldots, m\}$:
- if max $< a_i$ then set max $= a_i$
- return max

Analysis

- time complexity for each element: $O(\log n)$
- overall time complexity: $O(m \log n)$
- space complexity $O(\log n)$

Data Stream Models for Graphs

Earliest work to explicitly consider graph problems was the influential paper by Henzinger, Raghavan, Rajagopalan 1999: Computing on data streams

Theorem (Henzinger et al. (1999), Theorem 6)

In P passes, computing the connected components or testing planarity in an n node graph requires $\Omega(n/P)$ space. Finding the sinks in a directed graph requires $\theta(n/P)$ space.

There are one-pass algorithms for connected components and planarity testing that use $O(n \log n)$ space (e.g., incremental).

One-Pass Algorithm for Testing Connectedness

Simple algorithm that constructs a spanning forest:

- Maintain a set of edges H and add the next edge $\{u, v\}$ in the stream to H if there is no path from u to v in H.

Lemma

The above algorithm correctly decides in one pass of the data stream if the given graph is connected. The time complexity for each element is $O(\log n)^2$. *The overall time complexity is* $O\left((n + |E| \log^* n) \log n\right)$, *the space complexity is* $O(n \log n)$.

Proof.

Implementation via Union-Find, similar to the subtask in the Kruskal algorithm (for solving the minimum spanning tree problem) of deciding if a considered edge closes a cycle in the currently computed spanning forest. The find-operation needs time $O(\log n)$ in the RAM model and amortized in a sequence of $|E|$ operations $|E|O(\log^* n)$, whereas $\log^* n := \min\{k \mid Z(k) \geq n\}$ and $Z(0) := 1$, $Z(i) := 2^{Z(i-1)}$ für $i \in \mathbb{N}$. □

Semi-Streaming Model

- Most problems are provably intractable if the available space is sublinear in $n = |V|$
- Many problems become feasible if memory is roughly proportional to n
- $\rightarrow$ most of the work focuses on semi-streaming model

Definition (Semi-streaming model for $G = (V, E)$)

- stream consists of a sequence of $|E|$ unordered pairs $e = \{u, v\}$, where $u, v \in \{1, 2, \ldots, n\}$
- space restricted to $O(n \text{ polylog } n)$ space

- stream $S = (e_1, e_2, \ldots, e_m)$ defines an undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$
- stream can also have weights/costs, then the input elements are, e.g., $(e = \{u, v\}, w(e))$

Basic Results in the Semi-Streaming Model

- Breadth-First-Search cannot be realized in the semi-streaming model
- in order to construct the first k layers of the BFS-tree, either $\Omega(k)$ passes or $\Omega(n^{1+1/k})$ bits storage are necessary [McGregor 2009]

General techniques for graph streams: Spanners

Definition (Spanner)

A subgraph H of G is an α-spanner for G if for all $u, v \in V$:

$$d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v),$$

where d_G and d_H are lengths of the shortest paths in G and H, resp..

Input: Stream of edges for graph G, semi-streaming model

- Construction: add an edge if it does not complete a short cycle

General techniques for graph streams: Sparsification

Idea: Subgraphs of G keeping (approximately) certain key properties

Definition (Cut Sparsification)

A weighted subgraph H of G is a $(1+\epsilon)$ cut sparsification for G if

$$\lambda_H(A) = (1 \pm \epsilon)\lambda_G(A) \quad \forall A \subset V,$$

where $\lambda_G(A)$ and $\lambda_H(A)$ denote the weight of the cut $(A, V \setminus A)$ in G.

Definition (Spectral Sparsification)

- A weighted subgraph H of G is a $(1+\epsilon)$ spectral sparsification if

$$x^T L_H x = (1 \pm \epsilon) x^T L_G x \qquad \forall x \in R^n,$$

where L_G and L_H are the Laplacians of G and H.

Spectral sparsifiers work very well with the merge-and-reduce paradigm.

General techniques for graph streams: Sketches

Idea: Construct a sketch for G keeping (approximately) certain key properties

Definition (Linear sketches)

They maintain a random linear projection of the input such that

- relevant properties of the input can be inferred from the sketch
- the sketch only needs small space

Sketches also work for dynamic data streams where edges can be added and removed.

Methods follow the following scheme:

- Design sketches such that it is possible to emulate the basic graph algorithms on them

Basics for the matching problem

See documents for lecture Part 2 for Matching

What we need:

- Theoretical background of M-augmenting paths and maximum matching
- Basic M-augmenting path algorithm for bipartite graphs (including analysis)
- Algorithm by Hopcroft and Karp for bipartite graphs (including analysis)
- Awareness of problems for general graphs and solution ideas

Algorithms for Matching in Data Streaming

Matchings problems are widely studied in the data streaming model:

- bipartite matching
- matching in general graphs
- weighted matching
- multiple passes
- insertion/deletion streams
- sparse graphs
- other variants

For an overview, see, e.g., Konrad 2018.
There are also many results concerning randomized algorithms.
We focus here on deterministic algorithms (excluding random order streams).

Greedy Single-Pass Algorithm for Matching

Algorithm

1. $M := \emptyset$
2. For each $e \in S$ do
3. If $M \cup \{e\}$ is matching, then $M = M \cup \{e\}$
4. Return M

- This Greedy algorithm is a 1/2-approximation for the maximum matching problem.
- So far it is the best approximation known for single-pass and the considered semi-streaming model,
- even if space $O(n^{2-\delta})$ is allowed for any $\delta > 0$
- There does not exist an algorithm with approximation ratio larger than $1 - 1/e$ which is roughly 0.6321 (Kapralov 2013)

Multi-Pass Algorithms for Matching

- Finding a maximum matching using p passes requires $O(n^{1+\Omega(1/p)}/p^{O(1)})$ space.
- No exact algorithm achieving this is known.
- There exists an $(1/(1+\epsilon))$-approximation algorithm that uses $O(p/\epsilon)$ passes and $O(n^{1+1/p})$ space (primal dual, Ahn and Guha 2013)
- There exists an $(1/(1+\epsilon))$-approximation algorithm that uses $O(1/\epsilon^8)$ passes and $O(n \text{ polylog } n)$ bits (Eggert et al. 2012)
- There is a $(1/2+0.02)$-approximation algorithm using two passes (Konrad et al. 2012).
- This has been improved to $1/2+0.0625$ (Kale and Tirodkar 2017)
- and again improved to $1/2+0.083$ (Esfandiari et al. 2016)

General framework for many algorithms

General framework for many algorithms

- During 1*st* pass: Compute a Greedy matching
- During 1*st* pass: Sometimes adding additional set of edges
- During passes: Repeatedly computing a set of vertex-disjoint M-augmenting paths and augmenting these paths
- Stop when M-augmenting paths exceed a certain length (length depends approximation factor)
- Sometimes: Postprocessing via Hopcroft-Karp on some set of edges in the storage

Two-Pass Algorithm by Konrad et al. 2012

Let $G = (A, B, E)$ the bipartite Graph with vertex sets A and B

Definition (Semi-matching)

A semi-matching matches all A vertices to B vertices without limmitations on the degree of a B vertex. A vertices have degree 1.

Definition (Incomplete λ-bounded semi-matching)

An incomplete λ-bounded semi-matching is a subset $S \subseteq E$ such that the degrees of all A-vertices in the semi-matching is bounded by 1 and the degree of all the B-vertices bounded by λ.

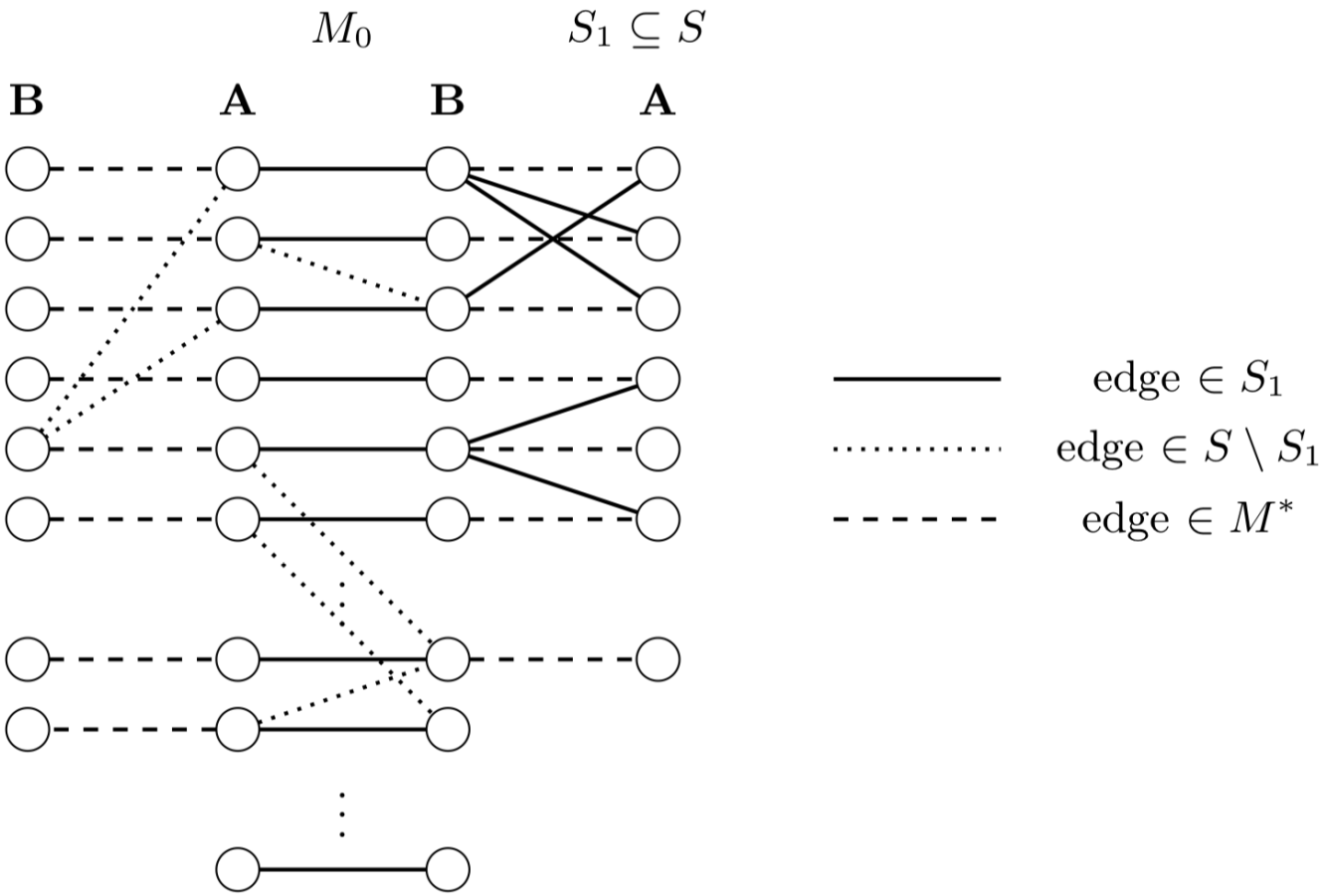
Two-Pass Algorithm by Konrad et al. 2012

Let $G = (A, B, E)$ the bipartite Graph with vertex sets A and B

Algorithm

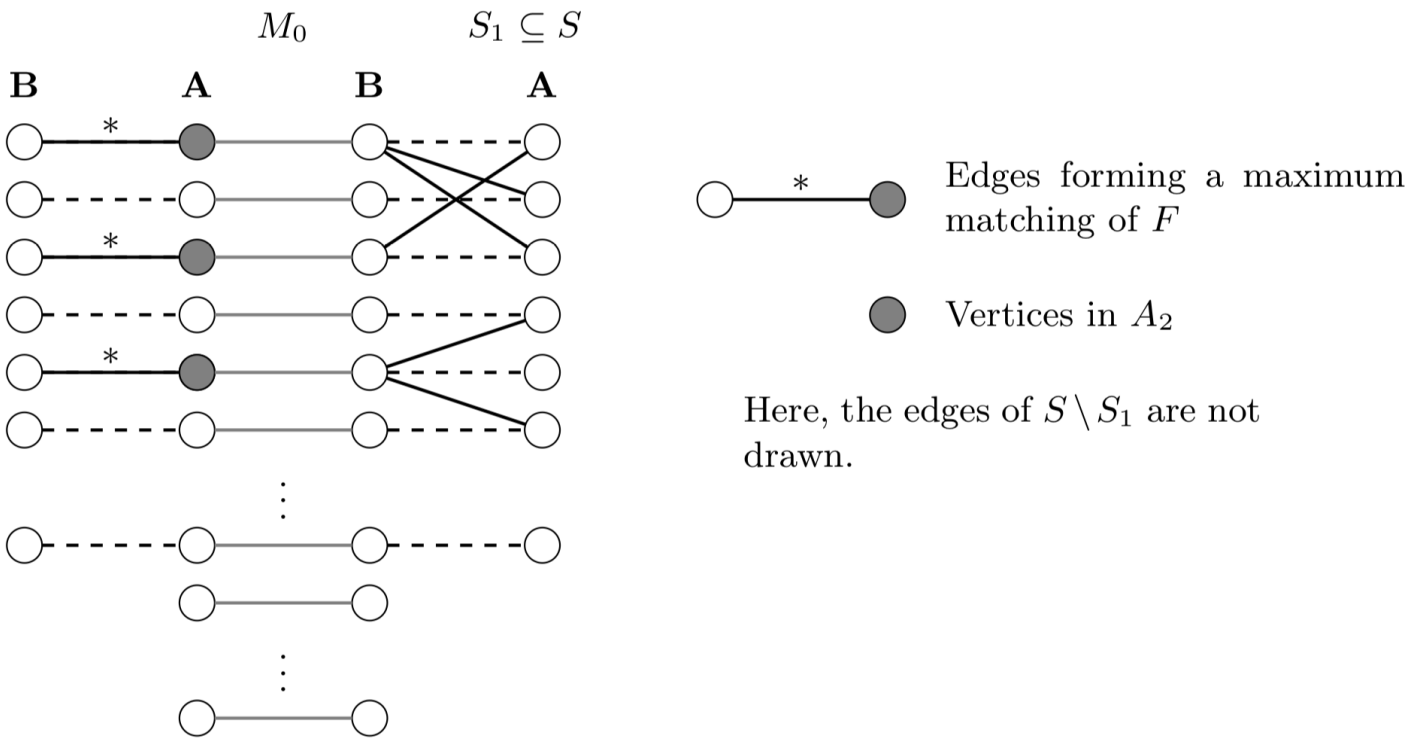
- During 1*st* pass: Compute a Greedy matching M_0
- During 1*st* pass: Compute an incomplete λ-bounded semi-matching S
- Build edge set S_1 consisting of all edges in S having one endvertex in A but not in M_0 and the other one in B and in M_0
- Build vertex set A_2 consisting of all vertices $a \in A$ and M_0 having a path of length 2 starting in a using a matched edge (by M_0) followed by an edge in S_1
- During 2*nd* pass: Let edge set F consisting of all edges incident to a vertex in A_2 and ending in a M_0-free vertex.
- During 2*nd* pass: Compute a Greedy matching M_2 on the set F
- Augment M_0 by edges in S_1 and M_2 (augmenting paths of length 3)

Illustration of First Pass for $\lambda = 2$



Some A vertices not matched in M_0 are matched in S. The edges incident to those define S_1

Illustration of Second Pass for $\lambda = 2$



$M_0 \cup S_1$ has 5 paths of length 2. These paths are not disjoint. But since the maximal degree in S is 2, it has at least $1/2 \cdot 5$ disjoint paths, and hence $|A_2| = 3 \geq 1/2 \cdot 5$.

A maximum matching in F is of size 3, and in the second pass, Greedy will find at least half of them leading to at least two augmenting paths of length 3.

Analysis of the algorithm

Let M^* be the value of the maximum matching.

Lemma

Let S be the calculated S by the algorithm for some λ. Then S is an incomplete λ-bounded semi-matching such that $|A(S)| \geq \frac{\lambda}{\lambda+1}|M^*|$.

Proof: Let $a \in A(M^*) \setminus A(S)$ and let b be its partner in M^*. The algorithm did not add the optimal edge (a, b) when it arrived. This implies that b was already matched to λ other vertices. Hence, $|A(M^*) \setminus A(S)| \leq \frac{1}{\lambda}|A(S)|$. Then the results follows by combining this inequality with $|M^*| - |A(S)| \leq |A(M*) \setminus A(S)|$.

Analysis of the algorithm

Let M^* be the value of the maximum matching.

Theorem

The algorithm approximates the optimum value with ratio $1/2 + 0.019$.

Proof (Very rough sketch): We have $|M| \geq |M_0| + \frac{1}{2}|opt(F)|$. One can show that $|opt(F) \geq |A_2| - 4\epsilon|M^*|$. Furthermore, $|A_2| \geq \frac{1}{\lambda}|S_1|$. Using the above lemma and further investigations we get to a factor of $1/2 + 1/52$.

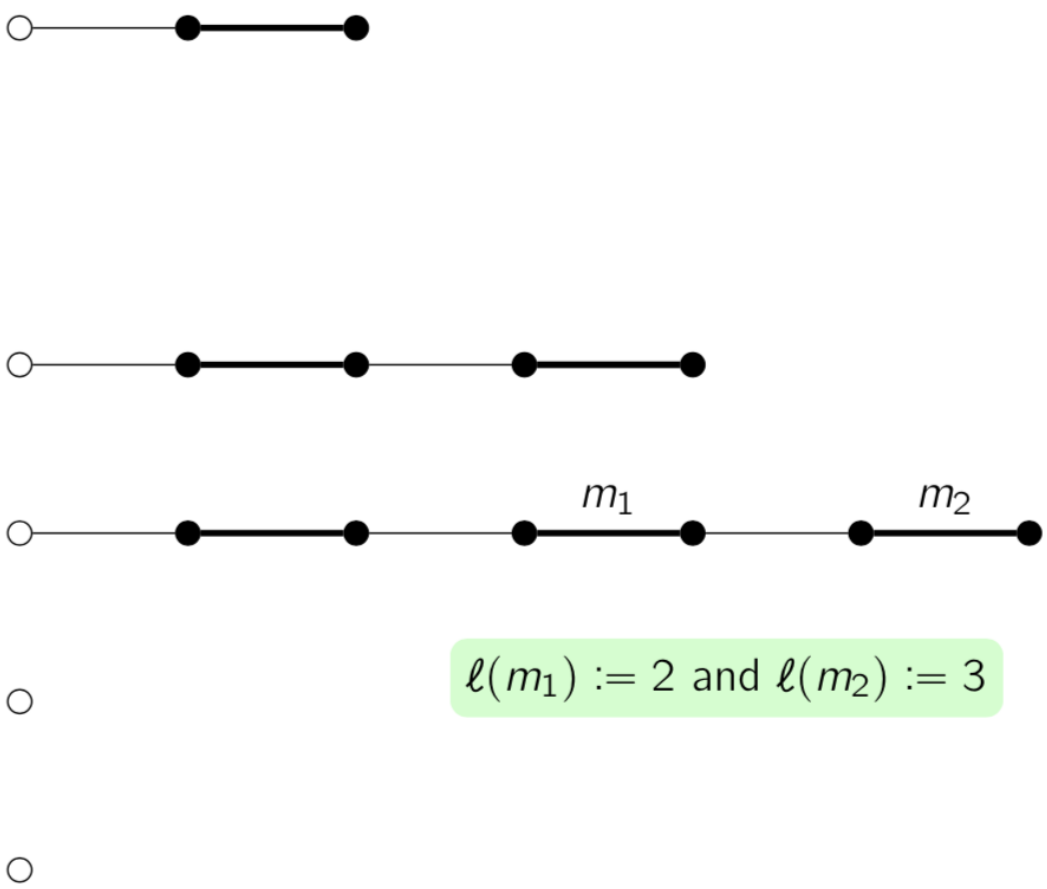
$\frac{1}{1+\epsilon}$-Approximation by Eggert et al. 2012

Let $G = (A, B, E)$ the bipartite Graph with vertex sets A and B

Algorithmic steps

- Goal: Find many pairwise vertex-disjoint M-augmenting paths of length at most $2k + 1$ ($k = 1/\epsilon$)
- Maintain a set of incomplete paths $\mathcal{I}$ and augmenting paths $\mathcal{A}$.
- Any two constructed paths are vertex disjoint.
- Incomplete paths end with a matching edge and with a vertex from A.
- In each incomplete path each vertex from B has a matching edge to its right.
- Matching edges have a position limit $l(m)$, which is $k + 1$ in the beginning and may decrease during the run of the algorithm.
- After each pass, the position number $i \in \{1, \ldots, k\}$ is increased or set back to 1 if $k + 1$ is reached.

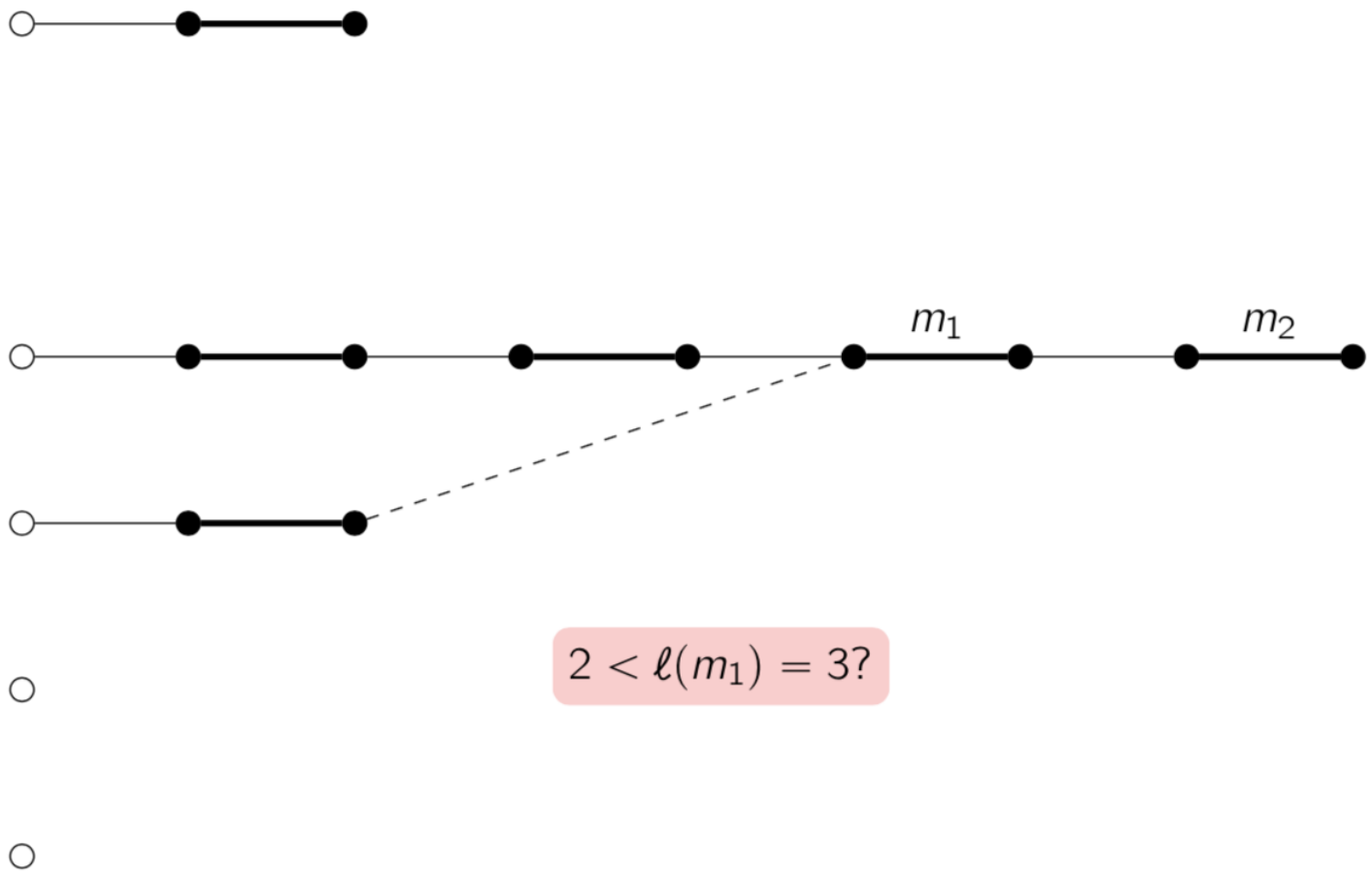
Illustration



For position number $i = 1, \ldots, k$: find many incomplete paths of length $2i$ or augmenting path of length $2i - 1$

[Source: Kliemann 2012]

Illustration



Incomplete paths can be rebuild

[Source: Kliemann 2012]

Analysis of the algorithm

Let M^* be the value of the maximum matching and $k = \frac{1}{\epsilon}$.

Lemma (from Hopcroft and Karp)

Let M and N two matchings with $|N| > |M|$ then the symmetric difference of them contains at least $|N| - |M|$ vertex disjoint M-augmenting paths.

Corollary

Let M be a matching such that the shortest M-augmenting path has length $2k+1$. Then M is a $\frac{k}{k+1}$-approximation of a maximum matching.

Proof: Since there are $|M^*| - |M|$ vertex disjoint M-augmenting paths with length at least $2k+1$, and each of these contains at least k matching edges, we know that there are at least $k(|M^*| - |M|)$ matching edges in M. We have $|M| \geq k|M^*| - k|M|$ which is equivalent to $|M|(k+1) \geq k|M^*|$ and thus $|M| \geq \frac{k}{(k+1)} \cdot |M^*|$.

$\frac{1}{1+\epsilon}$-Approximation by Eggert et al. 2012

Let $e = a, b$ with $a \in A$ and $b \in B$ be the current edge.
e can extend an incomplete path if

1. $P(\alpha)$ has exactly length $2(i-1)$
2. $P(\alpha)$ has a as its endpoint

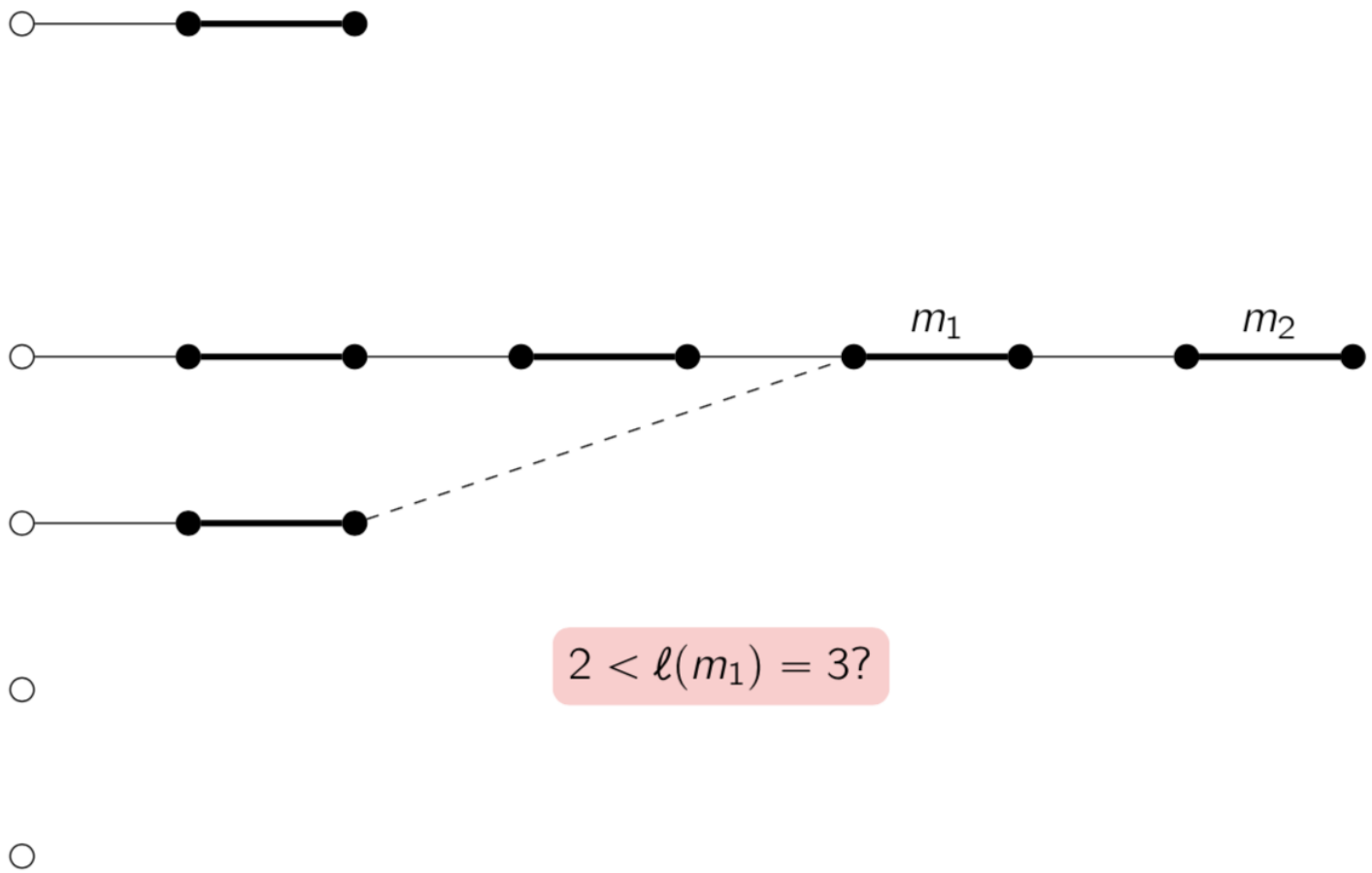
Since all paths are vertex-disjoint, there can be at most one such path.
If there is none, we ignore e. Otherwise:

$\frac{1}{1+\epsilon}$-Approximation by Eggert et al. 2012

Let $e = a, b$ with $a \in A$ and $b \in B$ be the current edge.
e may extend an incomplete path $P(\alpha)$.
If there is none, we ignore e. Otherwise:

- If b is a free vertex, we have found an augmenting path and we move the former incomplete path to the set of augmenting paths.
- If b is matched with edge m, we consider its position limit.
- If $l(m) \leq i$ we do nothing and ignore e.
- If $l(m) > i$ we have two cases:
- If the matching edge is not part of an incomplete path, we append e and its matching edge to the path $P(\alpha)$
- Otherwise m is already part of another incomplete path; then we move b and its right wing to the end of $P(\alpha)$ (append it there); in this case the matching edge gets a new position limit

Illustration



Incomplete paths can be rebuild

[Source: Kliemann 2012]

$\frac{1}{1+\epsilon}$-Approximation by Eggert et al. 2012

- When the pass is over (for an i), we test for $i = 1$ if we can finish.
- We finish if the number of incomplete paths does not exceed $\delta|M|$.
- One can show that this algorithm cannot find more then $2\delta|M|$ additional augmenting paths.
- When the pass is over and $i = k + 1$, we do backtracking:
- Remove the last two edges from each incomplete path
- The removed edge will never be put back to the same position, because of their position limit $l(m)$
- During a pass with position i, incomplete paths of length at most $2(i-2)$ or exactly $2i$ are not changed. Only longer paths will be reduced (if some part is taken away)

Analysis of the algorithm

Let M^* be the value of the maximum matching and $\epsilon > 0$.

Theorem

The algorithm finds a matching M with $(1+\epsilon)|M| \geq |M^*|$ and needs at most $O((\frac{1}{\epsilon})^8)$ passes.

There are practical improvements, e.g., instead of maintaining M-alternating paths one can do the same with M-alternating trees.

Practical results, e.g., in Kliemann 2016.

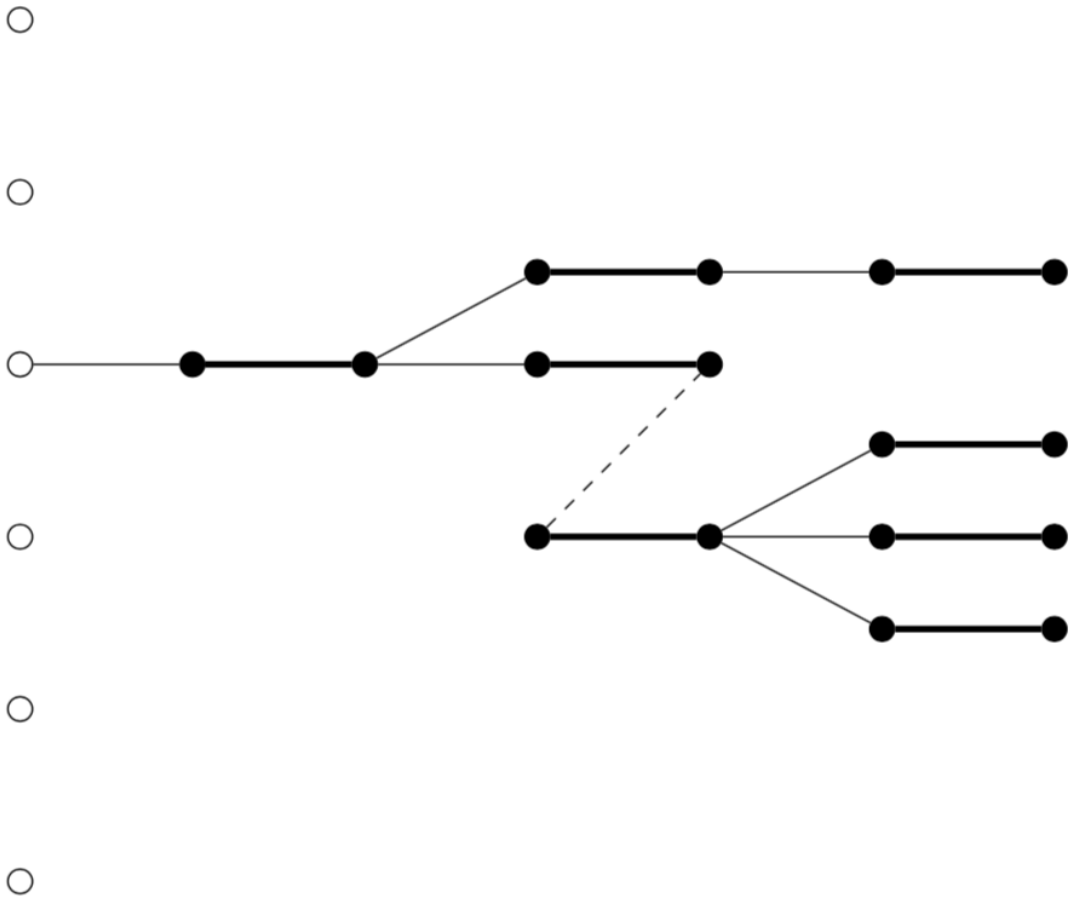
Experimental Results for Paths (Kliemann[1])

- Fix $k = 9$, that is, we guarantee a 90% approximation.
- $n = 40{,}000, 41{,}000, \ldots, 50{,}000$
- Density is limited by $D_{\max} = 1/10$.
- Number of edges ranges up to about $|E| = 62 \times 10^6$.

	maximum					mean				
	rand	degm	hilo	rbg	rope	rand	degm	hilo	rbg	rope
$O(k^5)$	11,886	14,180	7,032	4,723	2,689	107	145	3,337	257	378
$O(k^6)$	7,817	31,491	7,971	4,383	3,843	80	127	2,071	500	541
$O(k^7)$	7,121	32,844	9,106	5,687	5,126	74	166	2,033	844	790

[1] Talk at Dortmund in 2012

Generalization to Incomplete Trees



Experimental Results for Trees (Kliemann[2])

n	rand	degm	hilo	rbg	rope	$O(kn)$
8,192	35	21	51	43	53	18,433
16,384	25	26	52	44	57	36,865
32,768	37	19	54	48	59	73,729
65,536	39	23	55	50	64	147,457
131,072	28	28	57	51	65	294,913
262,144	39	42	58	53	64	589,825
524,288	27	26	58	55	61	1,179,649
1,048,576	44	26	62	57	53	2,359,297
2,097,152	41	31	60	57	59	4,718,593
4,194,304	30	29	62	58	54	9,437,185

Data based on roughly 800,000 instances in total.

[2] Talk at Dortmund in 2012