Processing Huge Graphs with STXXL

joint work with

D. Ajwani, L. Kettner, U. Meyer, P. Sanders, D. Schultes, J. Sibeyn

Roman Dementiev

Algorithmik II, Fakultät für Informatik, University of Karlsruhe

dementiev@ira.uka.de

Where do we have HUGE graph instances?

Motivation
- Where do we have HUGE graph instances?
- The Problem: RAM Model vs. Real Computer
- I/O Model

STXXL Library

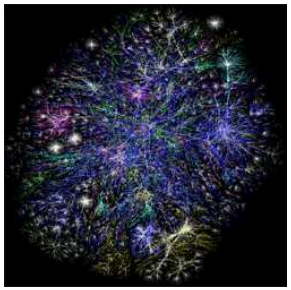
BFS

MST

Coloring

Challenges:

- Network Analysis
  - World Wide Web: clusters, cliques, transitivity coefficient
  - Phone calls: terrorist networks (CIA)!
  - . . .



- Route planning with PDAs
- Single agent search, action planning, state exploration in Artificial Intelligence, BDD reduction
- Approximation and heuristic algorithms for very large optimization problems
- . . .

The Problem: RAM Model vs. Real Computer

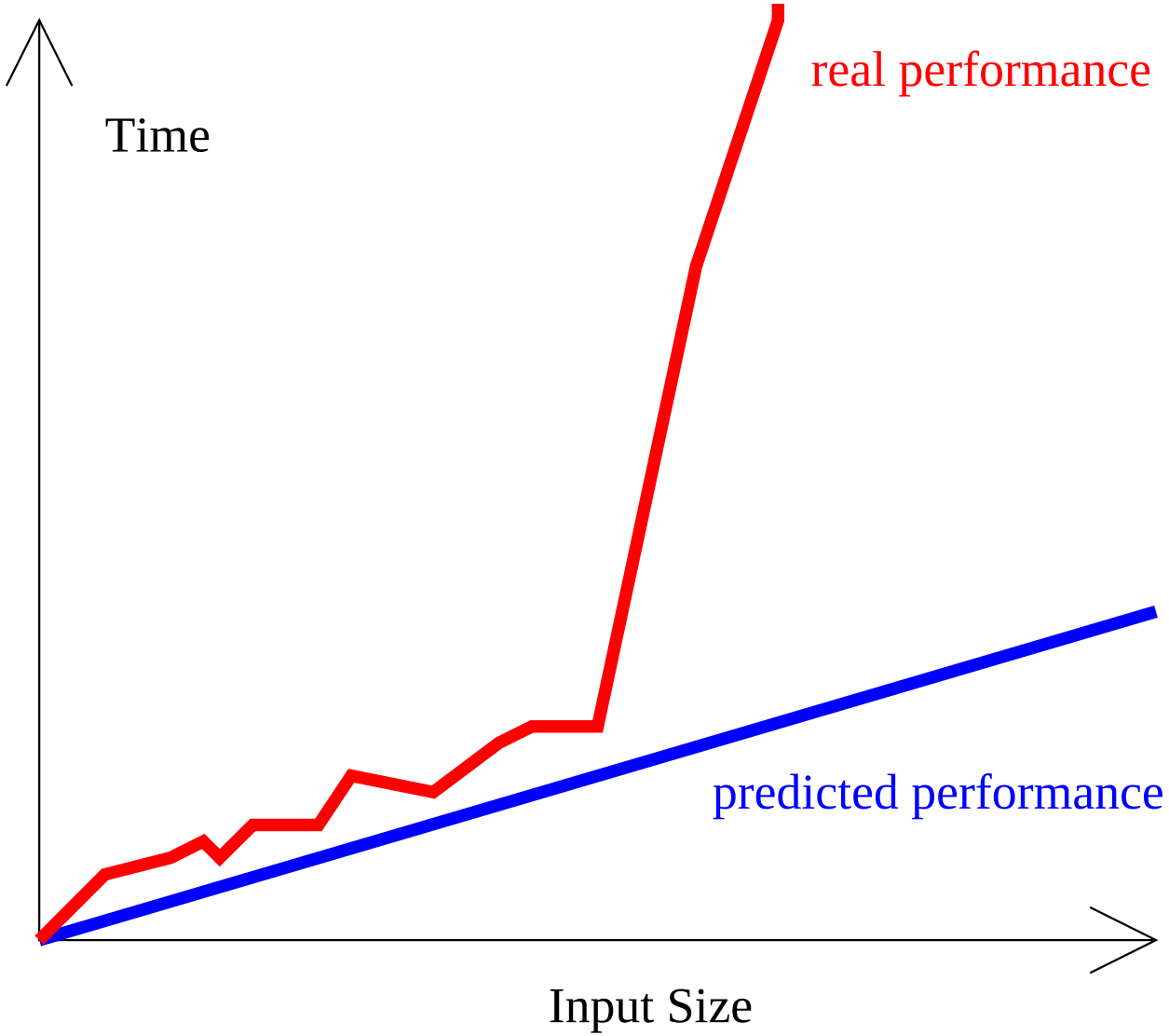
Motivation
- Where do we have HUGE graph instances?
- The Problem: RAM Model vs. Real Computer
- I/O Model

STXXL Library

BFS

MST

Coloring



I/O Model

Motivation
- Where do we have HUGE graph instances?
- The Problem: RAM Model vs. Real Computer
- I/O Model

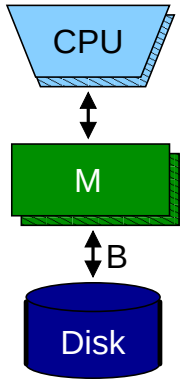
STXXLLibrary

BFS

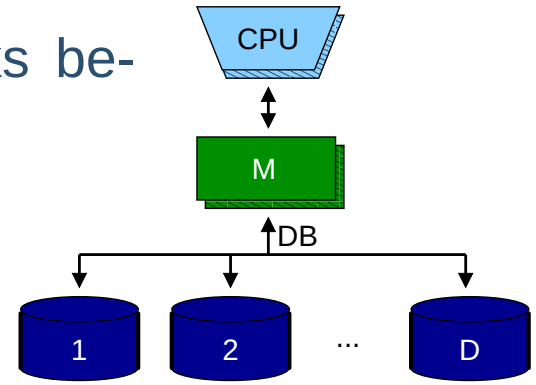
MST

Coloring

- Aggarwal–Vitter I/O model
  - N — size of input
  - M — size of main memory ($M \ll N$)
  - B — size of transfer block (128KB .. 2 MB)
  - Cost measure – number of I/Os
  - I/O-efficient alg. $\equiv$ External memory alg. $\equiv$ Secondary memory alg.



- Add disks to improve bandwidth: Parallel Disk Model (PDM) by Vitter–Shriver
  - D independent disks
  - In an I/O step try transfer D blocks between main memory and disks
  - Goal: minimize number of I/O steps



I/O-Efficient Tools

Motivation
STXXL Library
I/O-Efficient Tools
STXXL: a Collection of I/O-Efficient Tools
STXXL Features
BFS
MST
Coloring

- Scanning: $\text{scan}(N) = \mathcal{O}\left(\frac{N}{DB}\right)$ I/Os.
- Sorting: $\text{sort}(N) = \mathcal{O}\left(\frac{N}{DB}\lceil\log_{M/B}\frac{N}{M}\rceil\right), \approx \frac{2\,N}{DB}$.
- Special I/O-efficient data structures:
  - priority queues – $\mathcal{O}\left(\frac{1}{N}\text{sort}(N)\right)$ per operation
  - stacks, FIFO queues – $\mathcal{O}\left(\frac{1}{DB}\right)$ per operation
  - search trees – $\mathcal{O}\left(\log_{M/B}\frac{N}{M}\right)$ per operation
  - ...

STXXL: a Collection of I/O-Efficient Tools

Motivation
STXXL Library
- I/O-Efficient Tools
- STXXL: a Collection of I/O-Efficient Tools
- STXXL Features

BFS

MST

Coloring

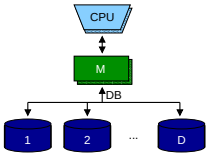
- STL – C++ Standard Template Library, implements basic containers (maps, sets, priority queues, etc.) and algorithms (quicksort, mergesort, selection, etc.)
- STXXL: Standard Template Library for XXL Data Sets
  `http://stxxl.sourceforge.net`
  containers and algorithms that can process huge volumes of data that only fit on disks (I/O-efficient tools)
  - Compatible with STL
  - Performance–oriented

STXXL Features

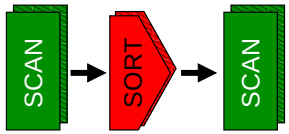
Motivation
STXXL Library
- I/O-Efficient Tools
- STXXL: a Collection of I/O-Efficient Tools
- STXXL Features

BFS
MST
Coloring

- Parallel disk support



- Handles very large problems (up to terabytes)
- Pipelining saves many I/Os
  - Feed output of I/O-eff. alg. to input of another I/O-eff. alg. directly



- Explicitely overlaps I/O and computation
- Avoids superfluous copying
  - in OS I/O subsystem and the library itself
- Compatible with STL – C++ Standard Template Library
  - Short development times
  - Reuse of STL code (e.g. STL selection algorithm)

Breadth First Search

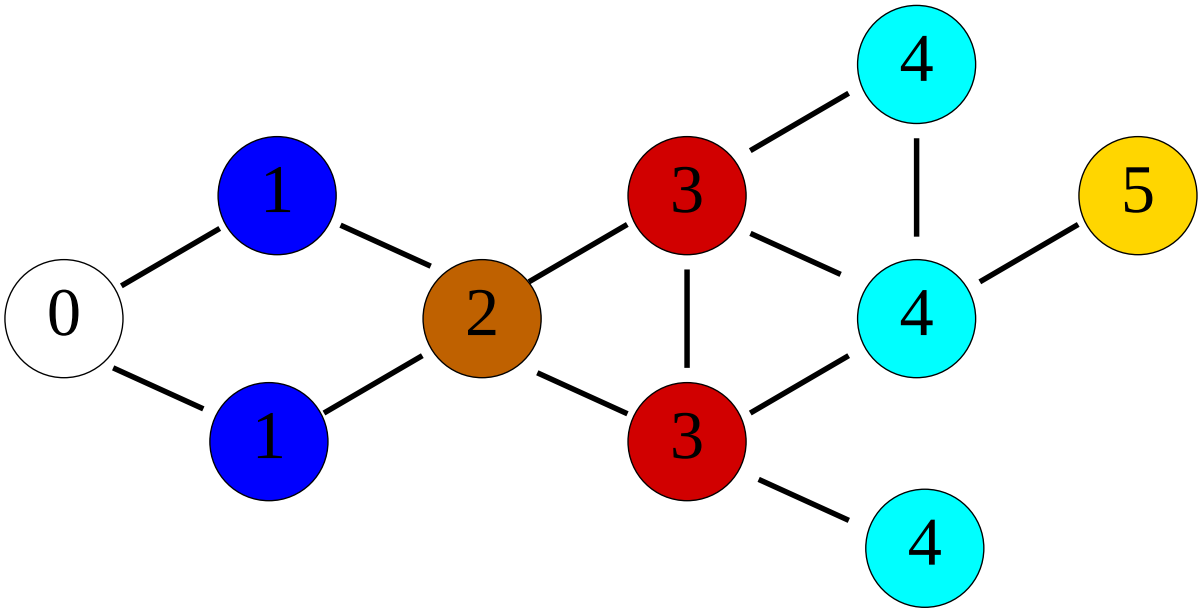
Motivation

STXXLLibrary

BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST

Coloring



- Applications: state exploration, shortest paths, crawling WWW, . . .

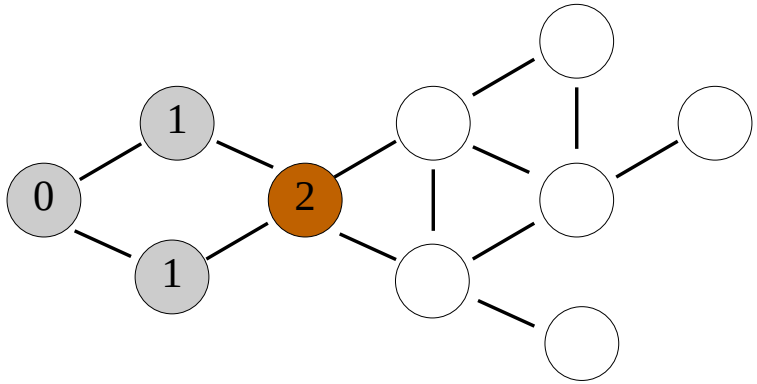
BFS: RAM algorithm

Motivation
STXXLLibrary
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST
Coloring

Internal memory algorithm:

```
Q: FIFO queue of nodes
Q.push(s)
while Q.notEmpty()
  u := Q.pop();
  visit u
  foreach unmarked neighbor v
    mark v
    Q.push(v)
```



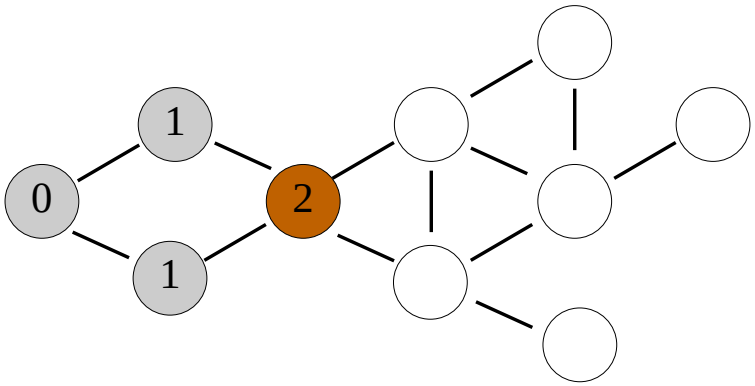
BFS: RAM algorithm

Motivation
STXXLLibrary
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST
Coloring

Internal memory algorithm:

```
Q: FIFO queue of nodes
Q.push(s)
while Q.notEmpty()
  u := Q.pop();
  visit u
  foreach unmarked neighbor v
    mark v
    Q.push(v)
```



- Marking nodes: $\mathcal{O}(m)$ I/Os
- Finding neighbors (adj. lists): $\mathcal{O}(n)$ I/Os

Algorithm of Munagala and Ranade

Motivation

STXXLLibrary

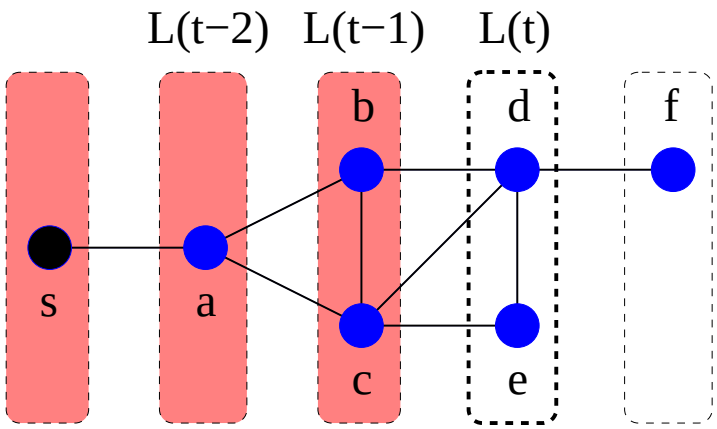
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

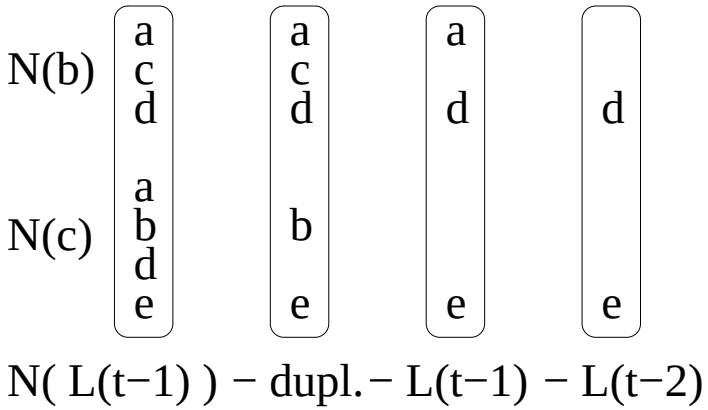
MST

Coloring

Creating $L(t)$: all reached neighbors of nodes in $L(t-1)$ belong to $L(t-2)$ or $L(t-1)$.

1. $N(L(t-1))$ = all neighbours of $L(t-1)$ — $\mathcal{O}(|L(t-1)| + \frac{|N(L(t-1))|}{D \cdot B})$ I/Os.
2. eliminate duplicates in $N(L(t-1))$ by sorting — $\mathcal{O}(\text{sort}(|N(L(t-1))|))$ I/Os.
3. eliminate nodes already in $L(t-1)$ by scanning — $\mathcal{O}(\text{scan}(|L(t-1)|))$ I/Os.
4. eliminate nodes already in $L(t-2)$ by scanning — $\mathcal{O}(\text{scan}(|L(t-2)|))$ I/Os.





$\sum_i |N(L(i))| \leq 2 \cdot m$ and $\sum_i |L(i)| \leq n \Rightarrow \mathcal{O}(n + \text{sort}(n+m))$ I/Os in total.

Algorithm of Mehlhorn and Meyer

Motivation
STXXLLibrary
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST
Coloring

Preprocessing: $\mathcal{O}(\text{sort}(n+m))$ I/Os

- partition nodes into $\mathcal{O}(n/\mu)$ subsets (clusters) s.t. any two nodes in same cluster have distance at most μ in G.
- store adjacency lists of nodes in the same cluster consecutively

BFS Phase: Refined Algorithm of Munagala-Ranade

- extract neighbors of $L(t)$ by scanning sorted external data structure $\mathcal{H}$ (hot pool) – prevents the $\mathcal{O}(n)$ accesses.
- if first node in a cluster is reached, add all adjacency lists of the cluster to $\mathcal{H}$.
- each adjacency list stays in $\mathcal{H}$ for at most μ iterations.
- $\mathcal{O}\left(n/\mu + \mu \cdot \text{scan}(n+m) + \text{sort}(n+m)\right)$ I/Os.
- Balancing: $\mathcal{O}\left(\sqrt{nm/B} + \text{sort}(n+m)\right)$ I/Os.

BFS Experiments: Random Graphs $m = 4n$

Motivation

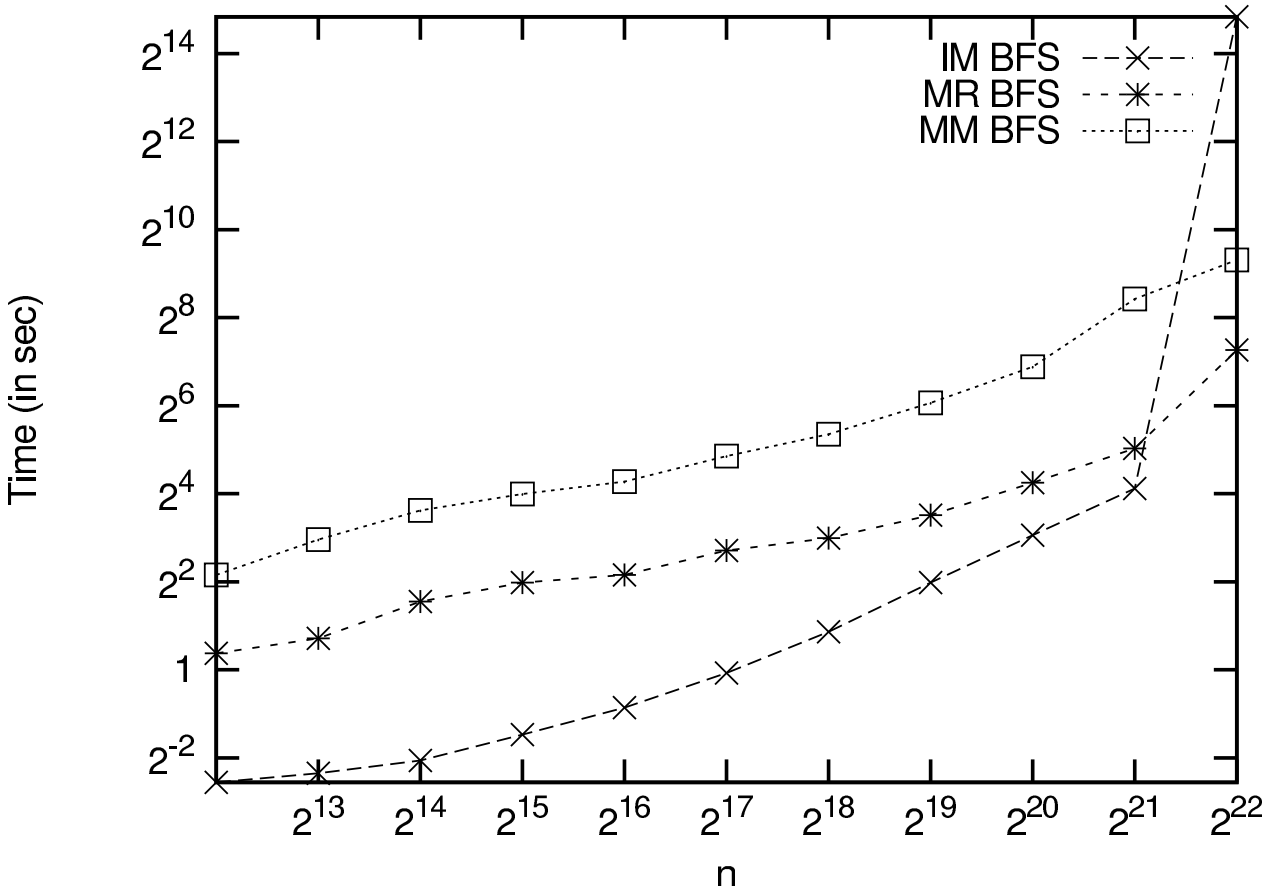
STXXL Library

BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST

Coloring

- Pipelined STXXL implementations of MR-BFS and MM-BFS
- Linux, 2GHz Xeon, 1 GB RAM, Seagate Barracuda hard disks (65 MB/s max, 9 ms seek time)



BFS Experiments

Motivation

STXXLLibrary

BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST

Coloring

MM-BFS versus MR-BFS

- Random graphs: MR-BFS 3.8 times faster — small diameter
- Grid graphs: MM-BFS 87 times faster — large diameter

BFS Experiments

Motivation
STXXLLibrary
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST
Coloring

MM-BFS versus MR-BFS

- Random graphs: MR-BFS 3.8 times faster — small diameter
- Grid graphs: MM-BFS 87 times faster — large diameter

Parallel disks ($D = 4$):

- Speedup is about two
- Become more CPU bound: may benefit from parallel processing in STXXL sorting

BFS Experiments

Motivation
STXXLLibrary
BFS
- Breadth First Search
- BFS: RAM algorithm
- BFS: RAM algorithm
- Algorithm of Munagala and Ranade
- Algorithm of Mehlhorn and Meyer
- BFS Experiments: Random Graphs $m = 4n$
- BFS Experiments
- BFS Experiments
- BFS Experiments

MST
Coloring

MM-BFS versus MR-BFS

- Random graphs: MR-BFS 3.8 times faster — small diameter
- Grid graphs: MM-BFS 87 times faster — large diameter

Parallel disks ($D = 4$):

- Speedup is about two
- Become more CPU bound: may benefit from parallel processing in STXXL sorting

Real graph instance:
Web crawl with n=130 million, m=1.4 billion

- Core: 10–12 levels, the rest: thousands of levels with 2-3 nodes per level
- MR-BFS (2.3 hours) beats MM-BFS (4.5 hours)
- External BFS computation is now feasible!

Minimum Spanning Trees and Forests

Motivation

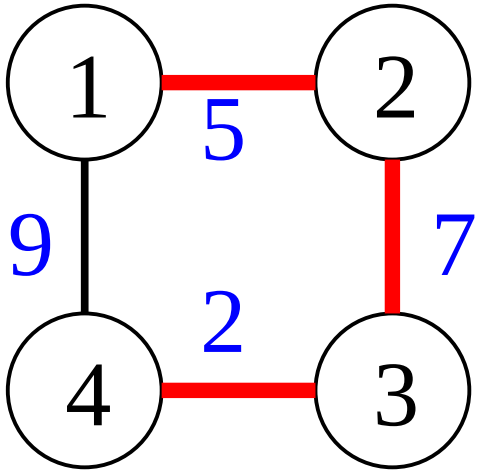
STXXLLibrary

BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring



- Applications: network design, clusteriung, subroutine in many algorithms for optimization problems, approximation of TSP
- Challenge: Can we compute MSTs for really huge graphs?

Semiexternal MST

Motivation
STXXLLibrary
BFS
MST
- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring

Kruskal's Algorithm [1956]

Idea: grow a forest

$T := \emptyset$ // subforest of the MST
foreach $(u, v) \in E$ in ascending order of weight **do**
  if u and v are in different subtrees of T **then**
    $T := T \cup \{(u, v)\}$ // Join two subtrees
return T

Problem: Random accesses to a union-find data structure

Semiexternal MST [Abello Buchsbaum Westbrook 02]

Assume $n \leq M - 2B$:
run Kruskal's algorithm using I/O-efficient `stxxl::sort`

Fast Node Reduction $n \longrightarrow n'$

Motivation

STXXLLibrary

BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

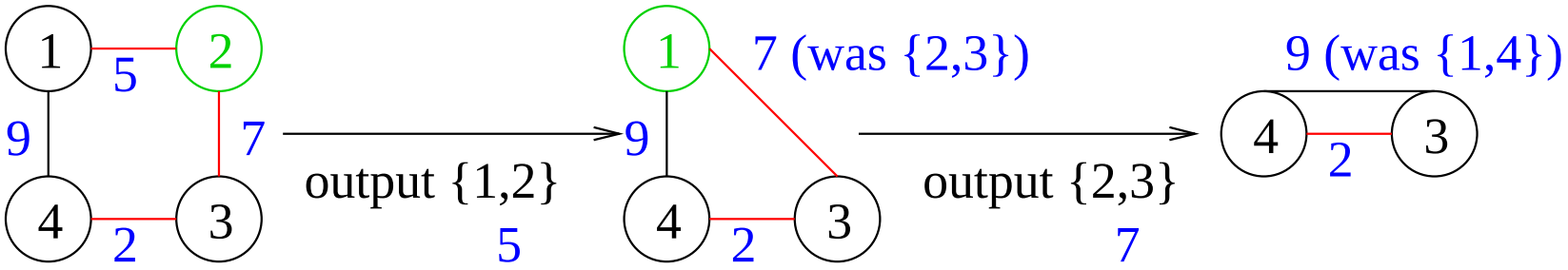
Coloring

for i := n **downto** $n' + 1$ **do**

pick a random node v

find the lightest edge (u, v) out of v and output it

contract (u, v)



#edges processed: $2m \ln \frac{n}{n'}$

External Implementation: Sweeping

Motivation

STXXLLibrary

BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring

π : random permutation $V \rightarrow V$

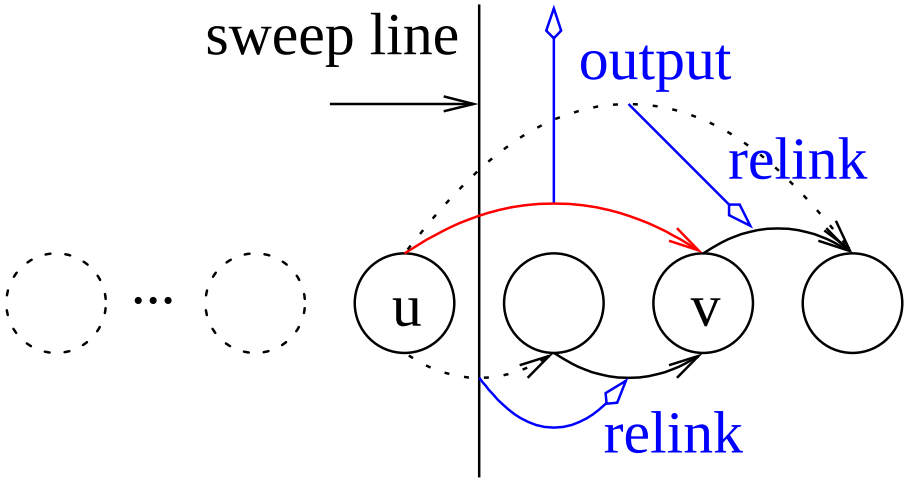
sort edges (u, v) by $\min(\pi(u), \pi(v))$

for i := n **downto** $n' + 1$ **do**

pick the node v with $\pi(v) = i$

find the lightest edge (u, v) out of v and output it

contract (u, v)



Problem: how to implement relinking?

Relinking Using Priority Queues

Motivation

STXXL Library

BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring

Q: STXXL priority queue // Order: max node, then min edge weight

foreach $(\{u,v\},c) \in E$ **do** Q.insert$((\{\pi(u),\pi(v)\},c,\{u,v\}))$

current := $n+1$

loop

  $(\{u,v\},c,\{u_0,v_0\})$:= Q.deleteMin()

  if current$\neq \max\{u,v\}$**then**

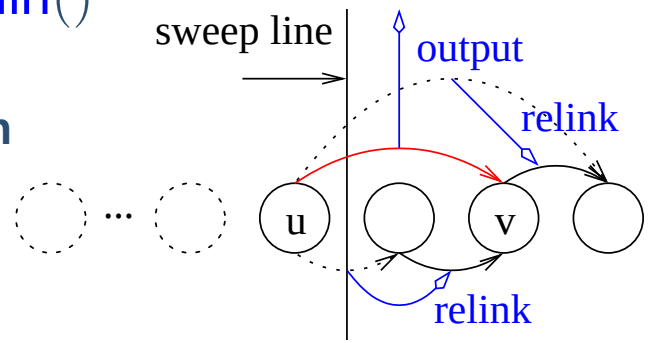
    if current$= M+1$ **then return**

    output $\{u_0,v_0\},c$

    current := $\max\{u,v\}$

    connect := $\min\{u,v\}$

  else Q.insert$((\{\min\{u,v\}, connect\},c,\{u_0,v_0\}))$



$\approx sort(10m \ln \frac{n}{M})$ I/Os with STXXL priority queues [Sanders 00]

Problem: Superfluous internal work overhead

$\Rightarrow$ for parallel disks CPU bound!

Sweeping with linear internal work

Motivation

STXXLLibrary

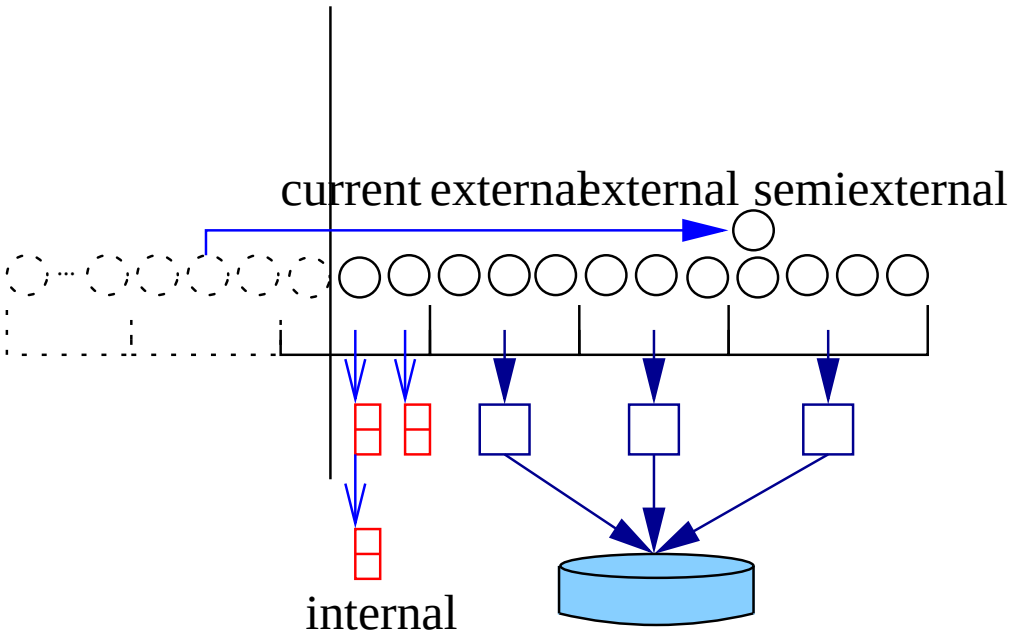
BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring

- Assume $m = \mathcal{O}(M^2/B)$
- $k = \Theta(M/B)$ external buckets with n/k nodes each
- M nodes for last "semiexternal" bucket



$m \approx 2n$

Motivation

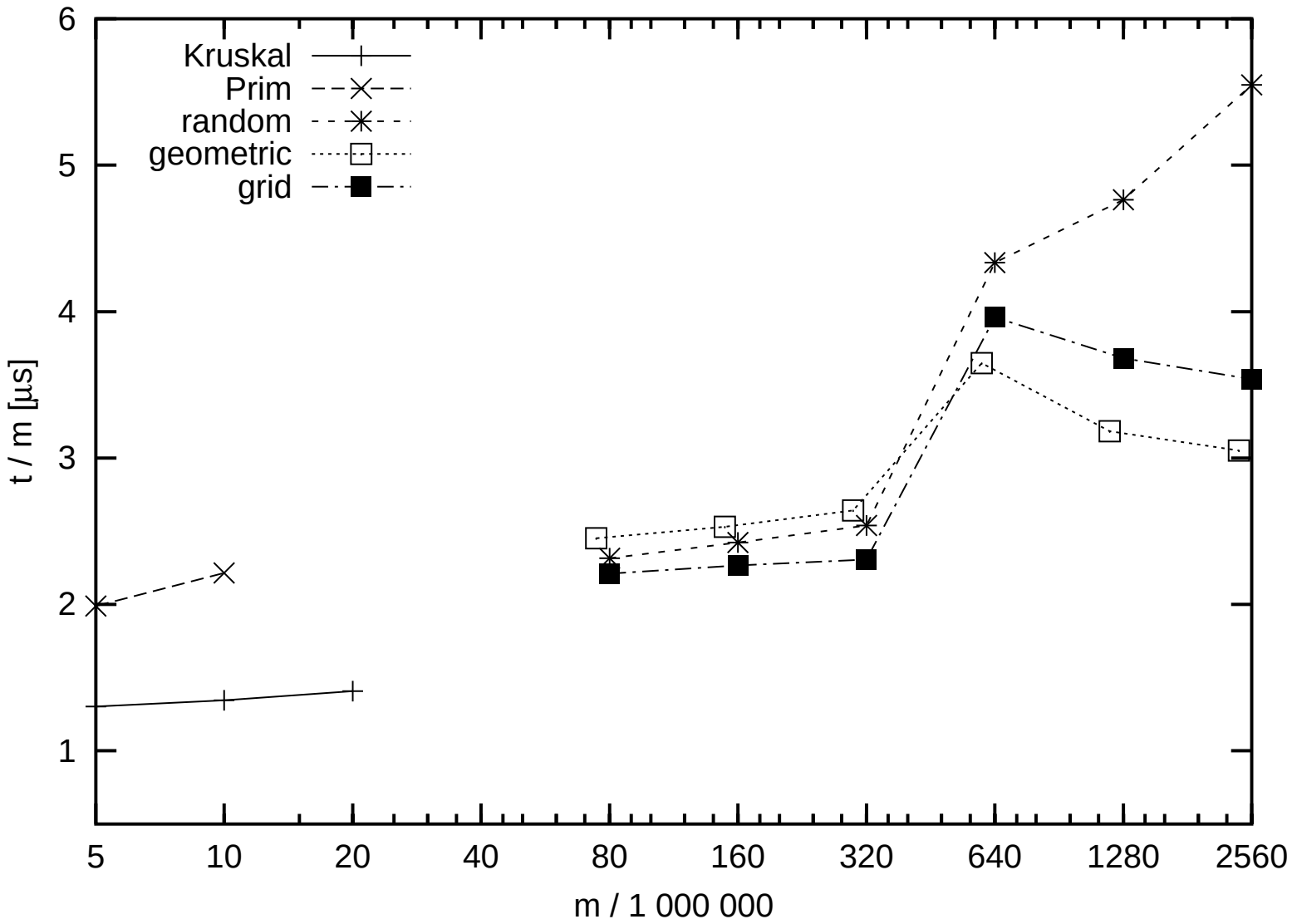
STXXLLibrary

BFS

MST

- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring



MST Experiments

Motivation
STXXLLibrary
BFS
MST
- Minimum Spanning Trees and Forests
- Semiexternal MST
- Fast Node Reduction $n \longrightarrow n'$
- External Implementation: Sweeping
- Relinking Using Priority Queues
- Sweeping with linear internal work
- $m \approx 2n$
- MST Experiments

Coloring

Huge inputs:

- grid graph with $n = 2^{32}$ (input = 96 GBytes) $\Rightarrow$ only 8h 40min

Random graphs with $n = 320 \cdot 10^6$ and $m = 640 \cdot 10^6$ (time per edge):

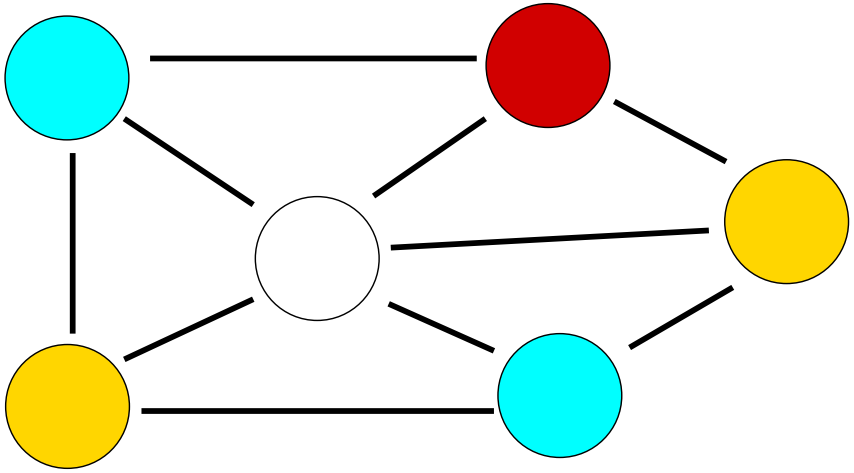
	1 disk	4 disks
bucket implementation	6.7 μs	4.3 μs
priority queue implementation	11.0 μs	8.9 μs

- Bucket version: going from one to four disks brings speedup 1.56 ($\Rightarrow$ not I/O bound)
- PQ version is CPU-bound but performs reasonably well
  - handles also graphs with high degree ($> M$) nodes

$\Rightarrow$ External MST computation is feasible!!

Graph Coloring

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary



- Applications: solving sparse linear systems of equations, resource allocation, scheduling, the construction and testing of VLSI circuits
- NP-complete problem
- Heuristics do not guarantee the quality, but very fast
- I/O-efficient coloring heuristics?

Greedy Coloring [Zeh]

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

Algorithm:

- Process nodes in arbitrary order
- When a node v is visited, it is assigned the smallest color, which has not been assigned to any of already visited neighbors of v.

I/O-Efficient priority queue needed ($\mathcal{O}(\text{sort}(m))$ I/Os):

Function $GreedyColoring(E)$
  ExternalPriorityQueue: Q // stores neighbor colors
  while $E \neq \emptyset \vee Q \neq \emptyset$ **do** (0)
    $v := \min\{E.front().src, Q.min().node\}$ (1)
    $U := \emptyset$ // used colors (2)
    while $v = Q.min().node$ **do** $U.append(Q.delMin().color)$ **od** (3)
    sort U (4)
    scan U and assign node v color $C := \min_{c \notin U} \{c\}$ (5)
    while $v = E.front().src$ **do** $Q.insert((E.popFront().dst, C))$ **od** (6)

Highest Degree First (HDF)

Motivation

STXXL Library

BFS

MST

Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

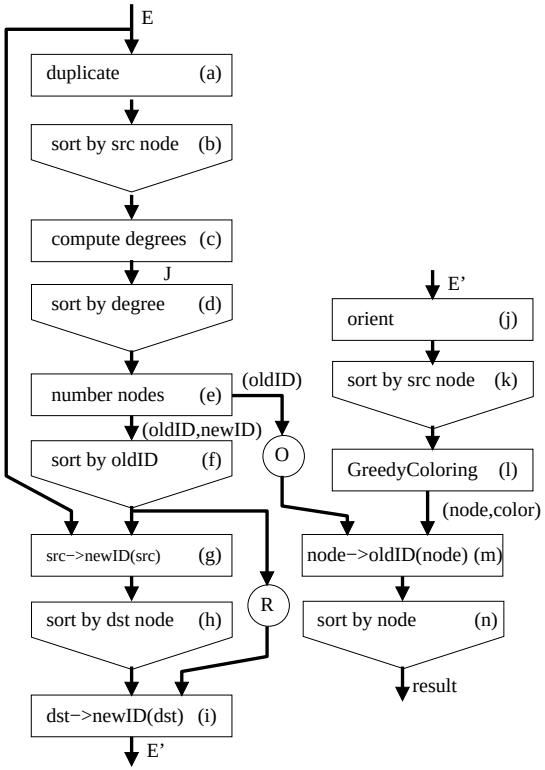
Visit and color the nodes in the order of their degree (difficult nodes first):

(1) Rename nodes in E such that $deg(v) > deg(u) \Rightarrow newName(v) < newName(u)$.

(2) Call $GreedyColoring(E)$ computing a coloring.

(3) Restore the old names in the coloring.

- Steps (1) and (3) can be implemented using sorting.
- Non-pipelined implementation max. I/O volume: $224m + 120n$ bytes
- STXXL pipelined implementation transfers at most: $112m + 72n$ bytes



Smallest Degree Last (SDL)

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

- (Phase 1) recursively remove a node with the smallest degree from the graph.
- (Phase 2) greedily color the nodes in the *backward* removal order.

Random accesses to the adjacency lists during node removal $\Rightarrow \Omega(m)$ I/Os.

Batched Smallest Degree Last (BSDL)

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

- Remove many nodes (batch) in each recursion, each recursion costs $\mathcal{O}(\text{sort}(m))$ I/Os
- Take nodes with degrees at most $\delta(V, E)$, where δ maps an integer in range $[\min_{v \in V} \{deg(v)\}, \infty)$
- Possible maximum batch degree (δ) functions:
  - average degree $\delta(V, E) = \lceil 2m/n \rceil$
  - median degree
  - node c-quantile $\delta(V, E) = \min \{k \ : \ |V_{1..k}| \geq cn\}$
  - edge c-quantile $\delta(V, E) = \min \left\{k : \sum_{v \in V_{1..k}} deg(v) \geq 2mc\right\}$

BSDL: Planar Graphs

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

- On planar graphs BSDL with $\delta(V,E) = \lceil 2m/n \rceil$ reduces a constant fraction of edges in each recursion, guarantees:
  - Needs at most $\mathcal{O}(\text{sort}(m))$ I/Os
  - Since $\lceil 2m/n \rceil \leq 6 \Rightarrow$ BSDL produces a 7-coloring
- To reduce more nodes (faster) we can set $\delta(V,E) = 6$ and obtain the same guarantees (EM7 algorithm)

Experiments: Small Inputs

Motivation

STXXLLibrary

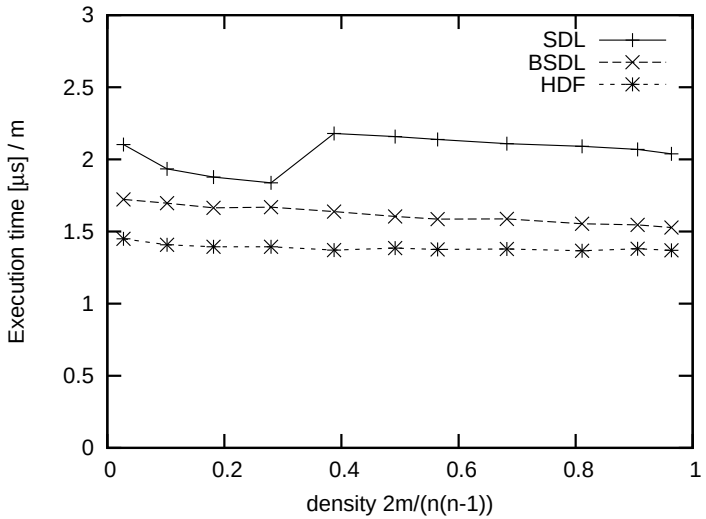
BFS

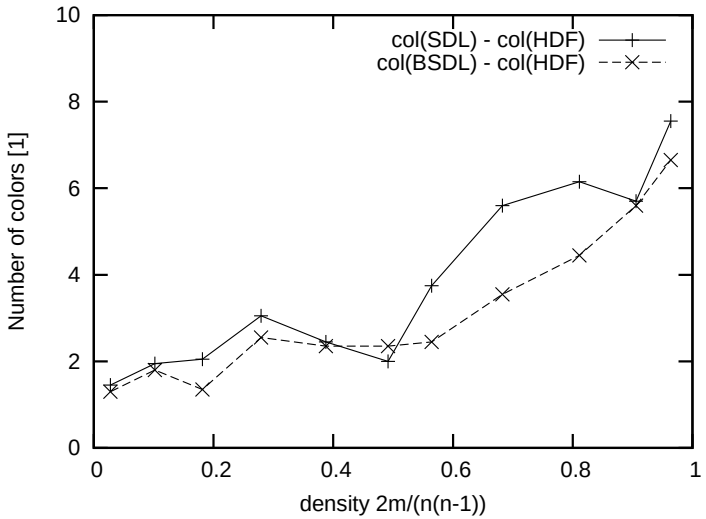
MST

Coloring

- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

- Linux, Opteron 2GHz, 1-4 IDE Disks
- SDL: an implementation from the BOOST library
- Random and real-world graphs
- Difference in the number of colors: $< 1.3\%$





Experiments: Large Inputs

Motivation
STXXLLibrary
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

- SDL can not run
- Random graphs: HDF is slightly faster
- Difference in the number of colors: $< 1.3\%$
- Minutes/#colors

Instance	$n/10^6$	$m/10^6$	HDF	BSDL
rand.planar1	8.4	16.8	0.55/6.0	0.57/5.1
rand.planar2	8.4	25.2	0.72/7.0	0.77/6.1
delaunay1	8.5	25.5	0.69/7	0.67/7
delaunay2	84.7	254.0	6.95/7	6.57/7
delaunay3	503.0	1509.0	49.87/7	43.19/7
webgraph	135.0	1079.9	35.86/246	43.57/246

- BSDL is slightly faster than HDF on delaunay graphs (they have many low-degree nodes)
- BSDL produces better colorings

Experiments: EM7 on Huge Planar Graphs

Motivation
STXXL Library
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

Instance	$n/10^6$	$m/10^6$	C	minutes
rand.planar1	8.4	16.8	5.5	0.49
rand.planar2	8.4	25.2	6.05	0.71
delaunay1	8.5	25.5	7	0.62
delaunay2	84.7	254	7	5.80
delaunay3	503	1509	7	42.1

- EM7 outperforms BSDL and HDF (can reduce more nodes)

$\Rightarrow$ Huge real-world graphs can be colored in an hour on a PC!

Summary

Motivation
STXXL Library
BFS
MST
Coloring
- Graph Coloring
- Greedy Coloring [Zeh]
- Highest Degree First (HDF)
- Smallest Degree Last (SDL)
- Batched Smallest Degree Last (BSDL)
- BSDL: Planar Graphs
- Experiments: Small Inputs
- Experiments: Large Inputs
- Experiments: EM7 on Huge Planar Graphs
- Summary

STXXL graph implementations not covered here:

- Triangle counting (clustering coefficient computation)

Open problems:

- Shortest paths
- Graph diameter (BFS-related)
- Better I/O-efficient coloring heuristics