1 Minimum Spanning Trees

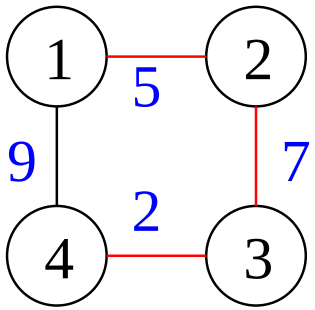
undirected Graph $G = (V, E)$.

nodes V, $n = |V|$, e.g., $V = \{1, \ldots, n\}$

edges $e \in E$, $m = |E|$, two-element subsets of V.

edge weight $c(e)$, $c(e) \in \mathbb{R}_+$.

G is connected, i.e., $\exists$ path between any two nodes.



Find a tree (V, T) with minimum weight $\sum_{e \in T} c(e)$ that connects all nodes.

MST: Overview

- Basics: Edge property and cycle property
- Jarník-Prim Algorithm
- Kruskals Algorithm
- Some tricks and comparison
- Advanced algorithms using the cycle property
- External MST

Applications: Clustering; subroutine in combinatorial optimization, e.g., Held-Karp lower bound for TSP. Challenging real world instances???

Anyway: almost ideal “fruit fly” problem

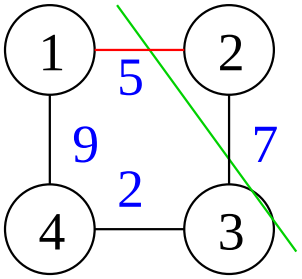
Selecting and Discarding MST Edges

The Cut Property

For any $S \subset V$ consider the cut edges

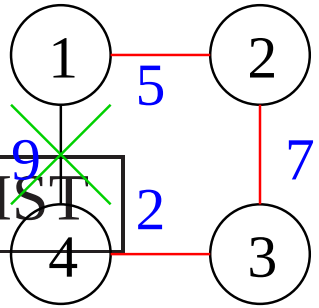
$C = \{\{u,v\} \in E : u \in S, v \in V \setminus S\}$

The lightest edge in C can be used in an MST.



The Cycle Property

The heaviest edge on a cycle is not needed for an MST.



The Jarník-Prim Algorithm [Jarník 1930, Prim 1957]

Idea: grow a tree

$T := \emptyset$

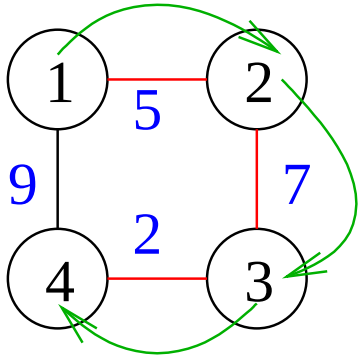
$S := \{s\}$ for arbitrary start node s

repeat $n-1$ times

find (u,v) fulfilling the cut property for S

$S := S \cup \{v\}$

$T := T \cup \{(u,v)\}$



Implementation Using Priority Queues

Function jpMST(V, E, w) : Set **of** Edge

dist$=[\infty,\ldots,\infty]$: **Array** $[1..n]$// dist$[v]$ is distance of v from the tree

pred : **Array of** Edge// pred$[v]$ is shortest edge between S and v

q : PriorityQueue **of** Node with dist$[\cdot]$ as priority

dist$[s]$:= 0; q.insert(s) for any $s \in V$

for $i := 1$ **to** $n-1$ **do do**

u := q.deleteMin() // new node for S

dist$[u]$:= 0

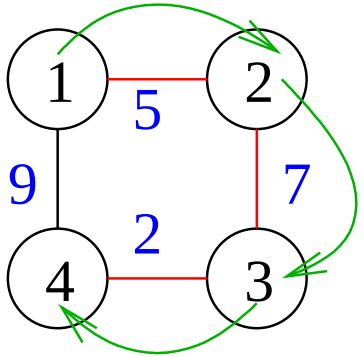
foreach $(u,v) \in E$ **do**

if $c((u,v)) <$ dist$[v]$ **then**

dist$[v]$:= $c((u,v))$; pred$[v]$:= (u,v)

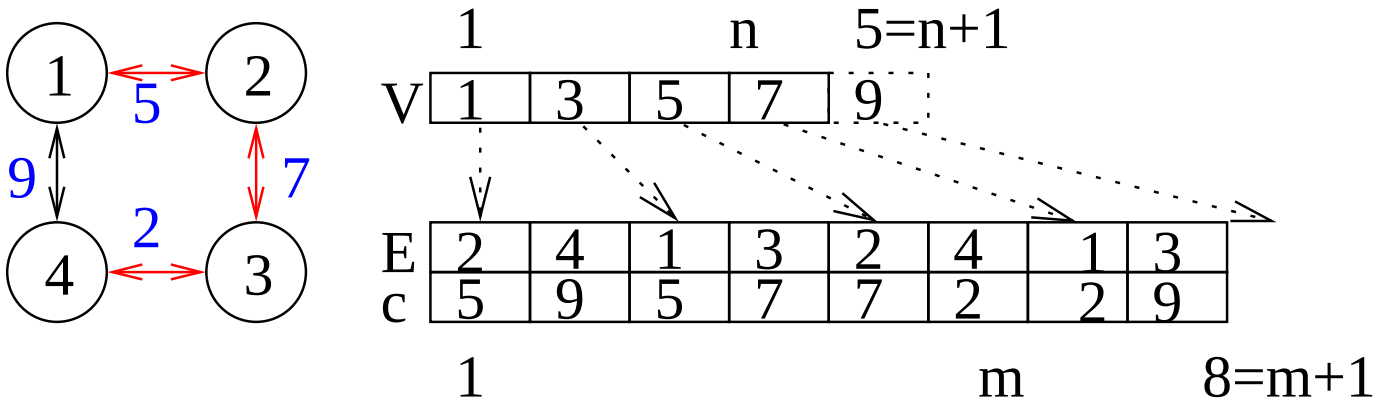
if $v \in q$ **then** q.decreaseKey(v) **else** q.insert(v)

return $\{\text{pred}[v] : v \in V \setminus \{s\}\}$



Graph Representation for Jarník-Prim

We need node → incident edges



+ fast (cache efficient)
+ more compact than linked lists
– difficult to change
– Edges are stored twice

Analysis

- $O(m+n)$ time outside priority queue
- n deleteMin (time $O(n \log n)$)
- $O(m)$ decreaseKey (time $O(1)$ amortized)

$\rightsquigarrow$ $O(m+n\log n)$ using Fibonacci Heaps

practical implementation using simpler pairing heaps.

But analysis is still partly open!

Kruskal's Algorithm [1956]

$T := \emptyset$ // 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

The Union-Find Data Structure

Class UnionFind($n : \mathbb{N}$) // Maintain a partition of $1..n$

parent$=[n+1,\ldots,n+1]$: **Array** $[1..n]$ **of** $1..n+\lceil \log n \rceil$

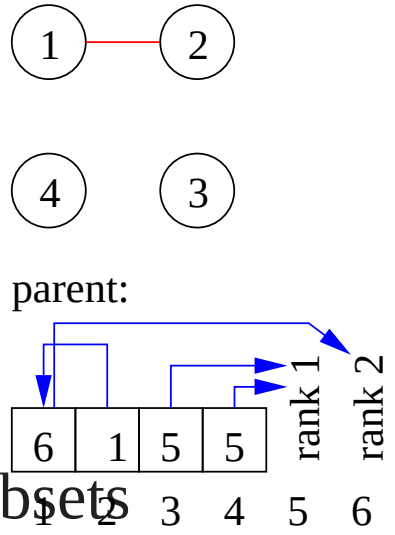
Function find($i : 1..n$) : $1..n$

if parent$[i] > n$ **then return** i

else $i' :=$ find(parent$[i]$)

parent$[i] := i'$

return i'



Procedure link($i, j : 1..n$)

assert i and j are leaders of different subsets

if parent$[i] <$ parent$[j]$ **then** parent$[i] := j$

else if parent$[i] >$ parent$[j]$ **then** parent$[j] := i$

else parent$[j] := i$; parent$[i]$++ // next generation

Procedure union(i, j) **if** find(i) $\neq$ find(j) **then** link(find(i), find(j))

Kruskal Using Union Find

T : UnionFind(n)

sort E in ascending order of weight

kruskal(E)

Procedure kruskal(E)

foreach $(u,v) \in E$ **do**

$u' := T.\text{find}(u)$

$v' := T.\text{find}(v)$

if $u' \neq v'$ **then**

output (u,v)

$T.\text{link}(u',v')$

Graph Representation for Kruskal

Just an edge sequence (array) !

- $+$ very fast (cache efficient)
- $+$ Edges are stored only once
- $\rightsquigarrow$ more compact than adjacency array

Analysis

$O(\text{sort}(m) + m\alpha(m,n)) = O(m \log m)$ where α is the inverse Ackermann function

Kruskal versus Jarník-Prim I

- Kruskal wins for very sparse graphs
- Prim seems to win for denser graphs
- Switching point is unclear
  - How is the input represented?
  - How many decreaseKeys are performed by JP?
    (average case: $n \log \frac{m}{n}$ [Noshita 85])
  - Experimental studies are quite old [**?**],
    use slow graph representation for both algs,
    and artificial inputs

Better Version For Dense Graphs ?

Procedure quickKruskal(E : Sequence **of** Edge)

  if $m \leq \beta n$ **then** kruskal(E) // for some constant β

  else

    pick a pivot $p \in E$

    $E_{\leq} := \langle e \in E : e \leq E \rangle$ // partitioning a la

    $E_{>} := \langle e \in E : e > E \rangle$ // quicksort

    quickKruskal($E_{\leq}$)

    $E'_{>} :=$ filter($E_{>}$)

    quickKruskal($E'_{>}$)

Function filter(E)

  make sure that leader$[i]$ gives the leader of node i // $O(n)$!

  return $\langle (u,v) \in E$: leader$[u] \neq$ leader$[v] \rangle$

1.1 Attempted Average-Case Analysis

Assume different random edge weights, arbitrary graphs

Assume pivot p has median weight

Let $T(m)$ denote the expected execution time for m edges

$m \leq \beta n$: $O(n \log n)$

Partitioning, Filtering: $O(m+n)$

$m > \beta n$: $T(m) = \Omega(m) + \mathrm{T(m/2)+T(2n)}$ [Chan 98]

Solves to $O\left(m + n\log(n) \cdot \log\frac{m}{n}\right) \leq O(m + n\log(n) \cdot \log\log n)$

Open Problem: I know of no graph family with

$T(n) = \omega(m + n\log(n))$

Kruskal versus Jarník-Prim II

Things are even less clear.

Kruskal may be better even for dense graphs

Experiments would be interesting.
Even for artificial graphs.

1.2 Filtering by Sampling Rather Than Sorting

R:= random sample of r edges from E

F :=MST(R) // Wlog assume that F spans V

L:= $\emptyset$ // "light edges" with respect to R

foreach $e \in E$ **do** // Filter

    C := the unique cycle in $\{e\} \cup F$

    if e is not heaviest in C **then**

        L:= $L \cup \{e\}$

return MST$((L \cup F))$

1.2.1 Analysis

[Chan 98, KKK 95]

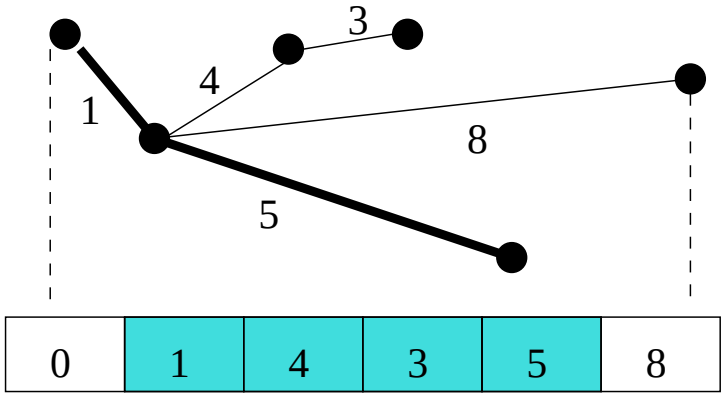
Observation: $e \in L$ only if $e \in \mathrm{MST}(R \cup \{e\})$.

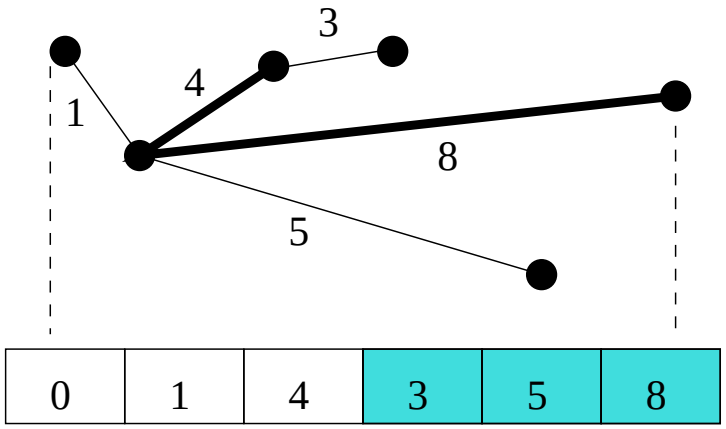
(Otherwise e could replace some heavier edge in F).

Lemma 1. $\mathrm{E}[|L \cup F|] \leq \frac{mn}{r}$

MST Verification by Interval Maxima

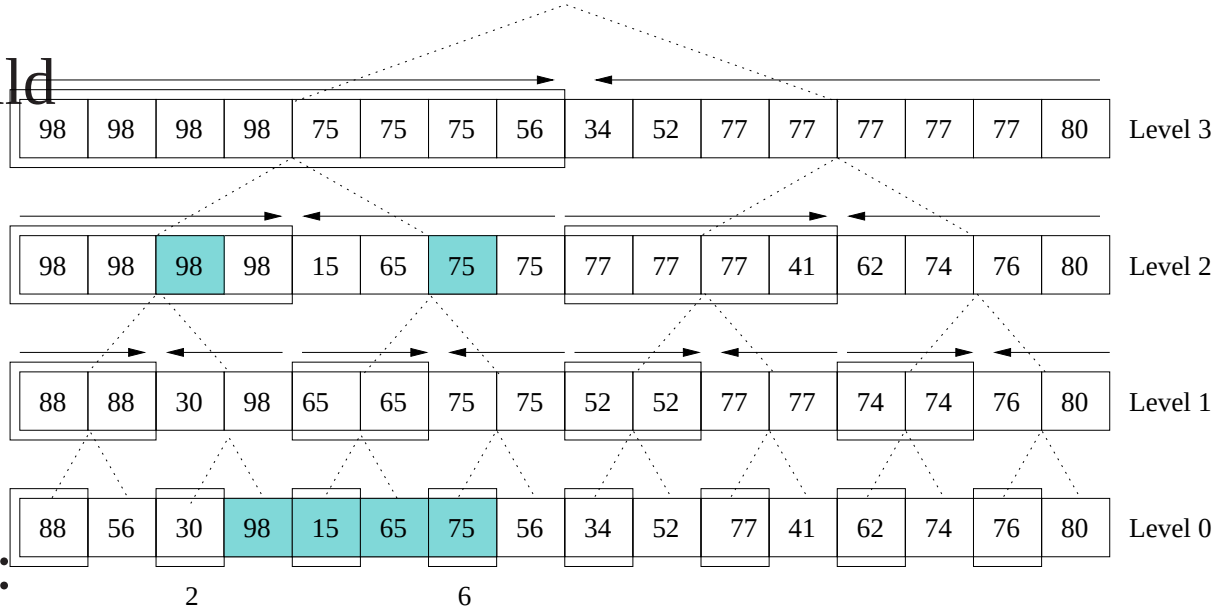
- Number the nodes by the order they were added to the MST by Prim's algorithm.
- w_i = weight of the edge that inserted node i.
- Largest weight on path$(u,v) = \max\{w_j \| u < j \leq v\}$.





Interval Maxima

Preprocessing: build $n \log n$ size array PreSuf.



To find $\max a[i..j]$:

- Find the level of the LCA: $\ell = \lfloor \log_2(i \oplus j) \rfloor$.
- Return $\max(\text{PreSuf}[\ell][i], \text{PreSuf}[\ell][j])$.
- Example: $2 \oplus 6 = 010 \oplus 110 = 100 : \ell = 2$

A Simple Filter Based Algorithm

Choose $r = \sqrt{mn}$.

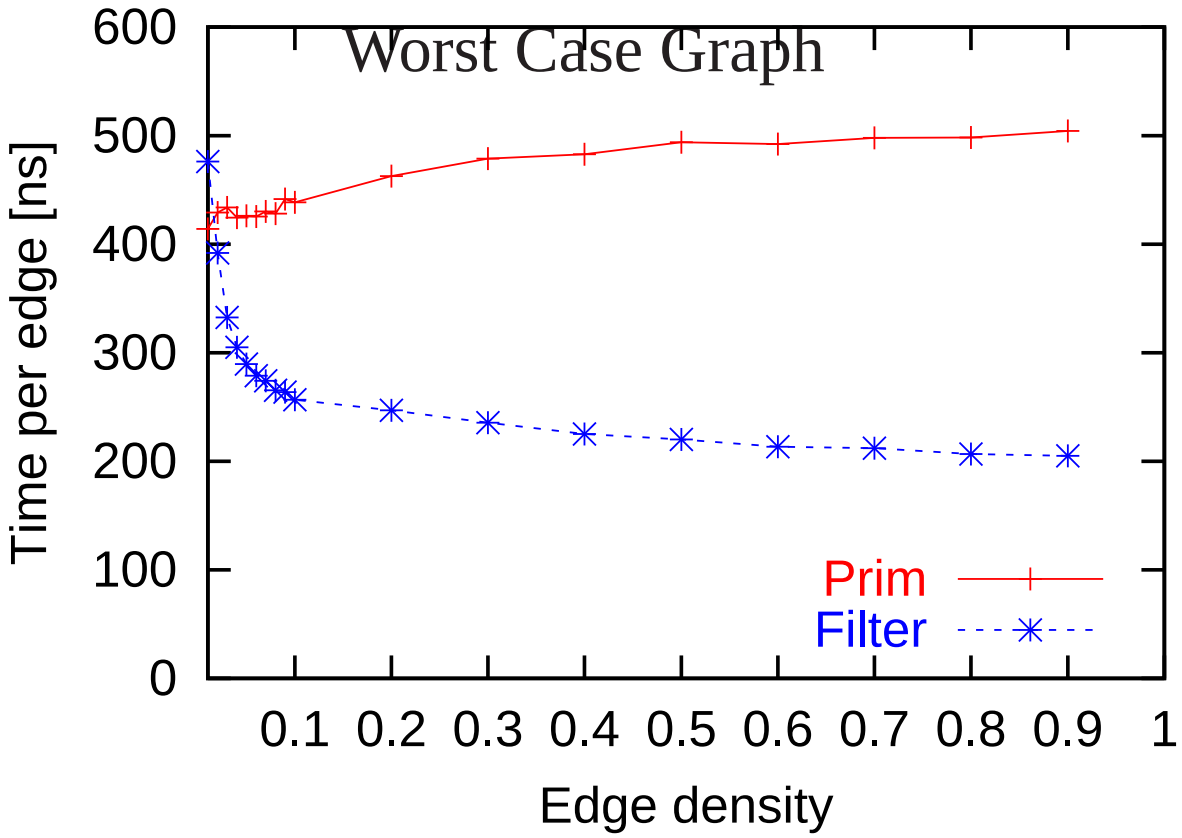
We get expected time

$$T_{\text{Prim}}(\sqrt{mn}) + O(n \log n + m) + T_{\text{Prim}}(\tfrac{mn}{\sqrt{mn}}) = O(n \log n + m)$$

The constant factor in front of the m is very small.

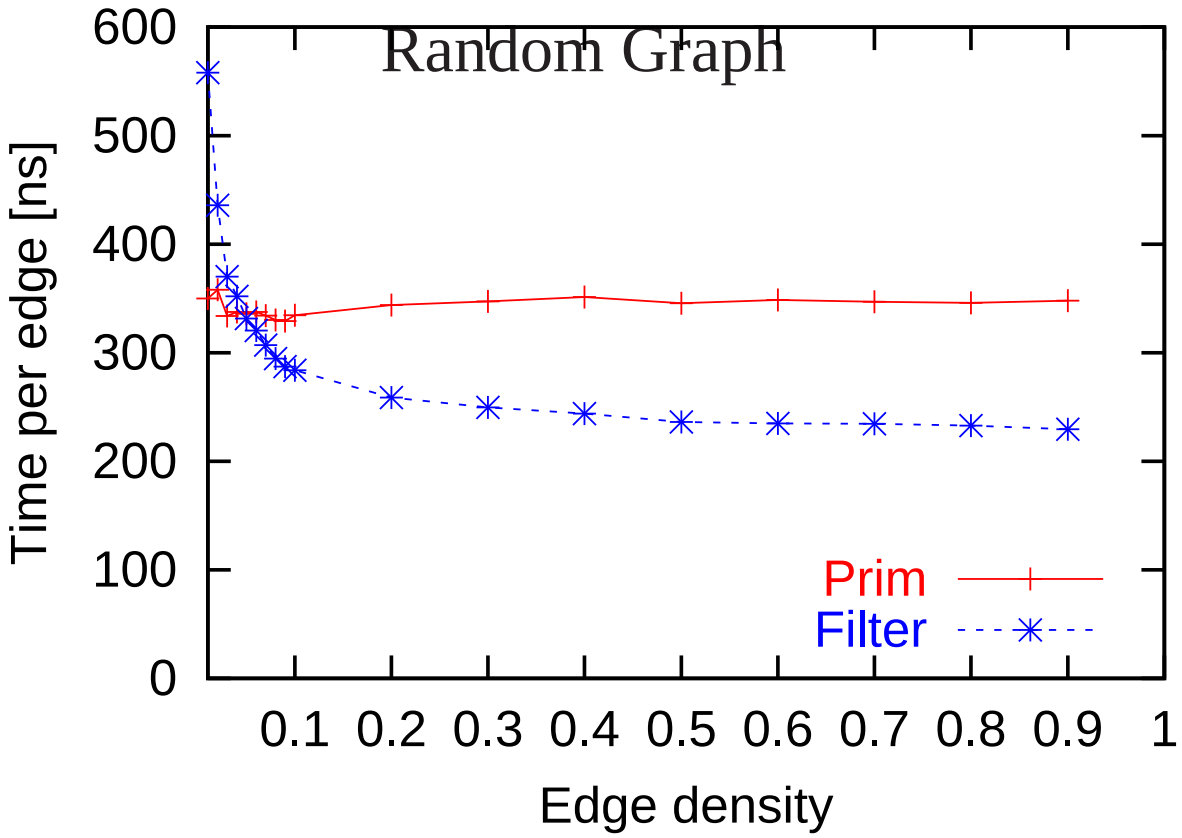
Results

10 000 nodes, SUN-Fire-15000, 900 MHz UltraSPARC-III+

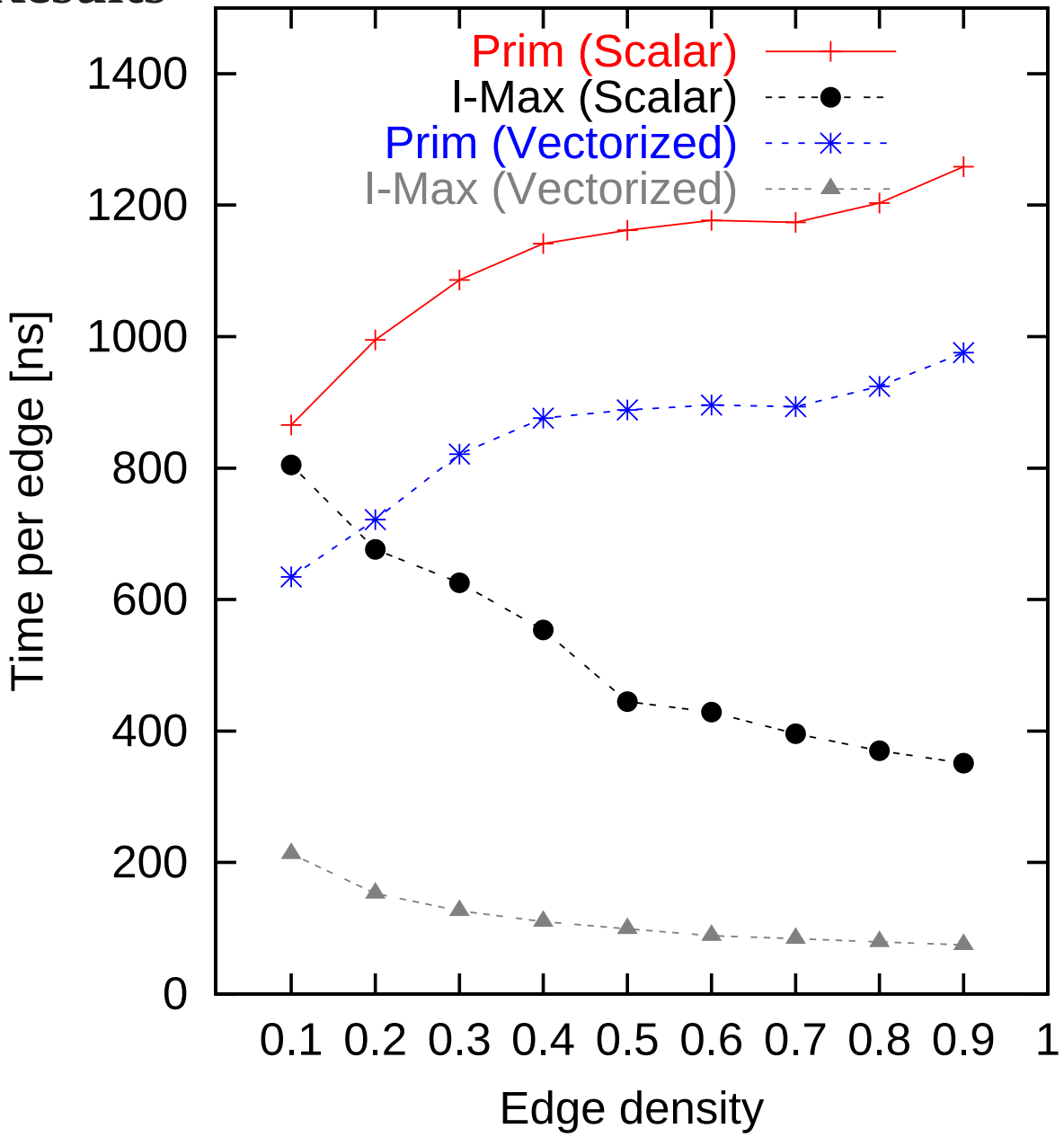


Results

10 000 nodes, SUN-Fire-15000, 900 MHz UltraSPARC-III+



Results



10 000 nodes,
NEC SX-5
Vector Machine
“worst case”

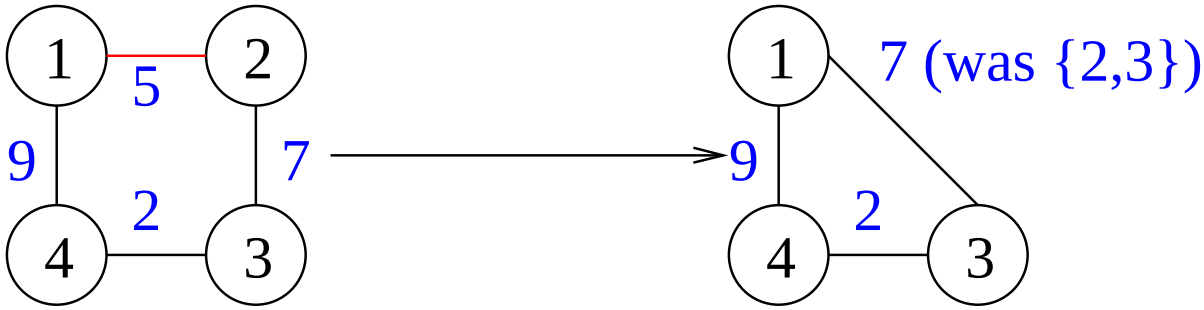
Edge Contraction

Let $\{u,v\}$ denote an MST edge.

Eliminate v:

forall $(w,v) \in E$ **do**

$E := E \setminus (w,v) \cup \{(w,u)\}$ // but remember orignal terminals

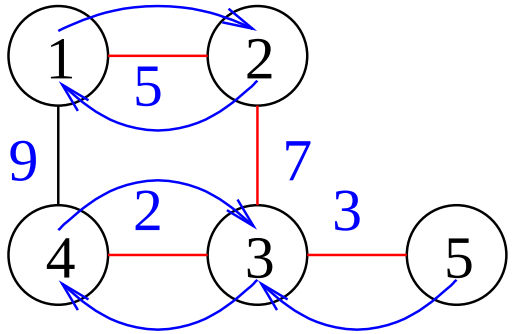


Boruvka's Node Reduction Algorithm

For each node find the lightest incident edge.
Include them into the MST (cut property)
contract these edges,

Time $O(m)$

At least halves the number of remaining nodes



External Implementation of Boruvka's Reduction

1. Sort E by first incident vertex. ($16m/B$ I/Os.)
2. Scan the result and output the lightest edge $(C(v),v)$ incident to each vertex v. The graph $G'_d = (V, L = \{(C(v),v) : v \in V\})$.
3. Remove one of the cycle-edges in each tree: Sort the edges (u,v) in L lexicographically by $(\min(u,v), \max(u,v))$. ($8n/B$ I/Os.) Scan L. When two antiparallel edges (u,v), (v,u) are found, remove (u,v). Now v is the root of a tree and should become the representative for all nodes in its tree.
4. Make two copies of each edge in L — an A-copy and a B-copy.
5. Sort the resulting list of $2n$ edges such that A-copies are sorted by their first incident node and B-copies are sorted by their second incident node. ($16n/B$ I/Os.) Scan the resulting list and form objects consisting of one A-edge (u,v) followed by consecutive B-edges of the form (v,w).
6. Sort the resulting object list by increasing weight of the A-edge generating list L'. ($16n/B$ I/Os.)
7. Set up a priority queue Q storing triples (c,v,v') of edge weights, node-ids, and their new name. Q initialized by inserting triples (c,w,v) for B-vertices (v,w) that are not preceded by an A-edge of the form (u,v).Scan L'. For an A-edge (u,v) with weight c in L' find a matching entry (c,v,v') and output the pair (v,v') recording that v' is the new name of v. Enqueue triples (c',w,v') for all B-edges of the form (v,w) associated with the A-edge (u,v). ($6n/B$ I/Os.)
8. Sort the resulting renaming sequence R by their first component. ($4n/B$ I/Os.)
9. Scan the edge list E and R simultaneously and rename the first component of each edge in E. This costs $8m/B$ I/Os for reading the sorted list of $2m$ 4-tuples and we have to account another $8m/B$ I/Os for explicitly storing E in in step 1.
10. Sort E by the second component. ($16m/B$ I/Os.)
11. Scan E and R renaming the second component of each edge. ($4n/B$ I/Os).

1.3 Cost of reducing the number of nodes by 2

- need 6 pipelined sorters + priority queue
- three different data types (edges, PQ entries, renaming pairs)

$\rightsquigarrow$ $(48m + 54n)/B$ I/Os to halve the node set

1.4 Simpler and Faster Node Reduction

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

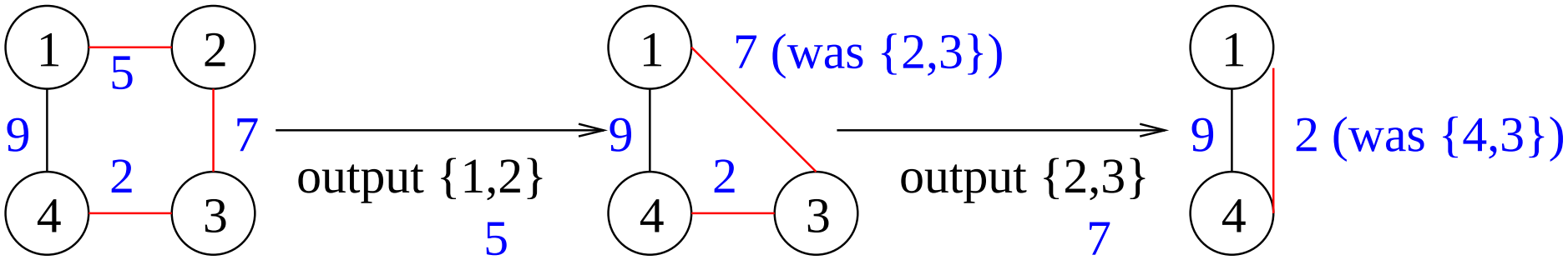
pick a random node v

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

contract (u,v)

$$\mathrm{E}[\mathrm{degree}(v)] \leq 2m/i$$

$$\sum_{n'<i\leq n} \frac{2m}{i} = 2m\left(\sum_{0<i\leq n}\frac{1}{i} - \sum_{0<i\leq n'}\frac{1}{i}\right) \approx 2m(\ln n - \ln n') = 2m\ln\frac{n}{n'}$$



1.5 Randomized Linear Time Algorithm

1. Factor 8 node reduction (3× Boruvka or sweep algorithm)
   $O(m+n)$.
2. $R \Leftarrow m/2$ random edges. $O(m+n)$.
3. $F \Leftarrow MST(R)$ [Recursively].
4. Find light edges L (edge reduction). $O(m+n)$
   $\mathrm{E}[|L|] \leq \frac{mn/8}{m/2} = n/4$.
5. $T \Leftarrow MST(L \cup F)$ [Recursively].

$T(n,m) \leq T(n/8, m/2) + T(n/8, n/4) + c(n+m)$

$T(n,m) \leq 2c(n+m)$ fulfills this recurrence.

1.6 External MSTs

Semiexternal Algorithms

Assume $n \leq M - 2B$:

run Kruskal's algorithm using external sorting

Streaming MSTs

If M is yet a bit larger we can even do it with m/B I/Os:

$T := \emptyset$ // current approximation of MST

while there are any unprocessed edges **do**

load any $\Theta(M)$ unprocessed edges E'

$T := \mathsf{MST}(T \cup E')$ // for any internal MST alg.

Corollary: we can do it with linear expected internal work

Disadvantages to Kruskal:

Slower in practice

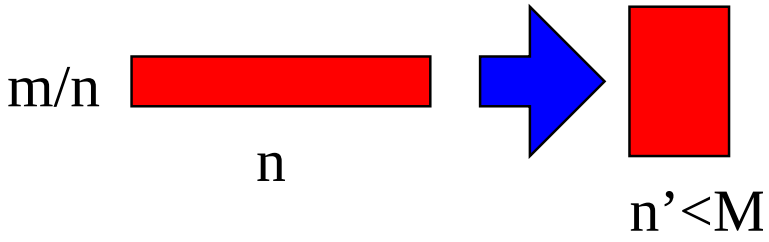
Smaller max. n

General External MST

while $n > M - 2B$ **do**

perform some node reduction

use semi-external Kruskal



Theory: $O(\text{sort}(m))$ expected I/Os by externalizing the linear time algorithm.

(i.e., node reduction + edge reduction)

External Implementation I: Sweeping

π : random permutation $V \rightarrow V$

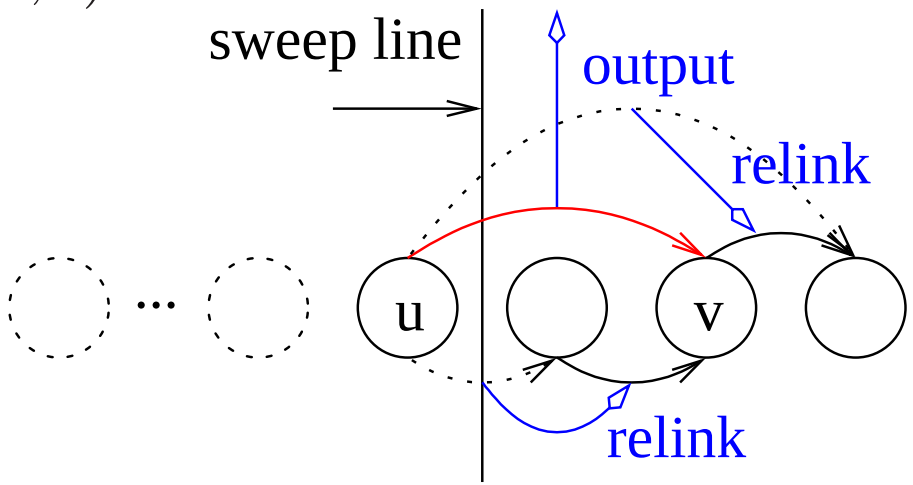
sort edges (u,v) by $\min(\pi(u),\pi(v))$

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)



Fast Random Permutations (no I/O)

$\pi : 0..n-1 \rightarrow 0..n-1$

Feistel permutations:

- assume node i can be represented as a pair (a,b) with $i = a + b\sqrt{n}$
- $\pi_f((a,b)) = (b, a + f(b) \bmod \sqrt{n})$ for random mapping $f : 0..\sqrt{n}-1 \rightarrow 0..\sqrt{n}-1$
- $\sqrt{n}$ is small $\rightsquigarrow$ implement f as a lookup table (for $n = 2^{32}$ need only 128 kB)
- $\pi(x) = \pi_f(\pi_g(\pi_h(\pi_l(x))))$ is good even for cryptography
- we use $\pi(x) = \pi_f(\pi_g(x)))$

External Implementation I: Sweeping

π : random permutation $V \rightarrow V$

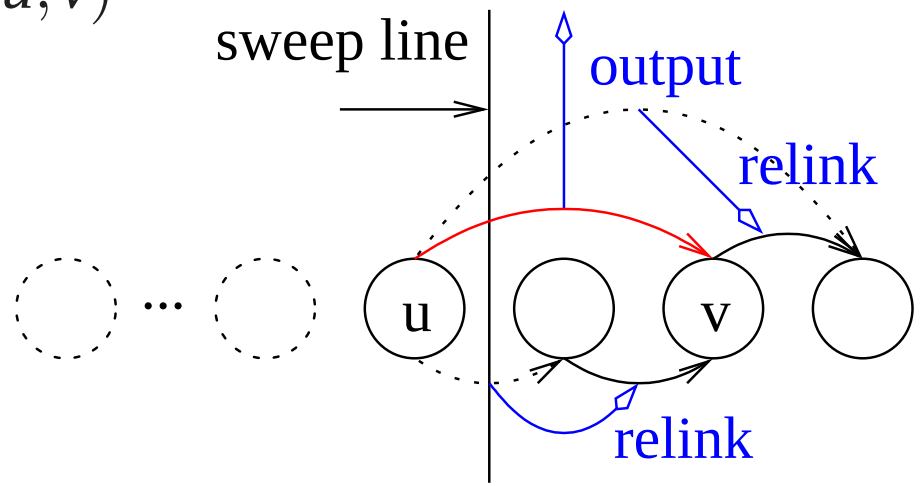
sort edges (u,v) by $\min(\pi(u),\pi(v))$

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

Q: 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**

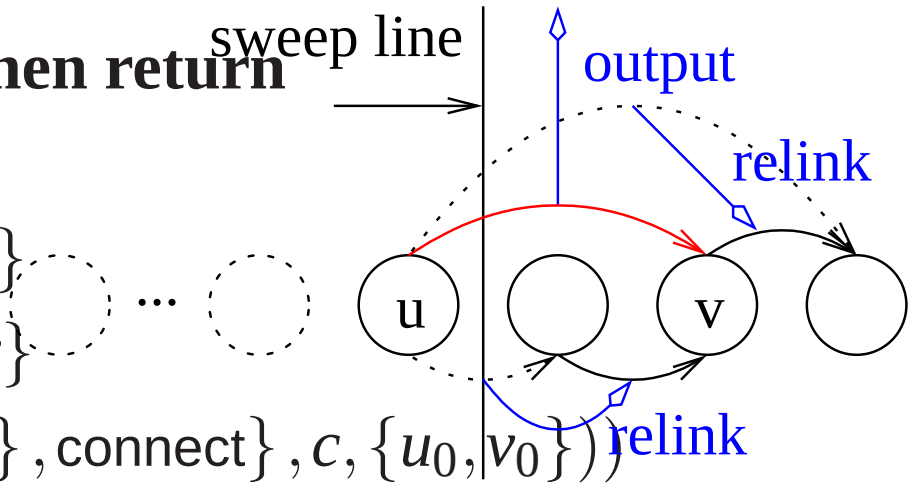
if current$= M+1$ **then return**

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

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

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

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

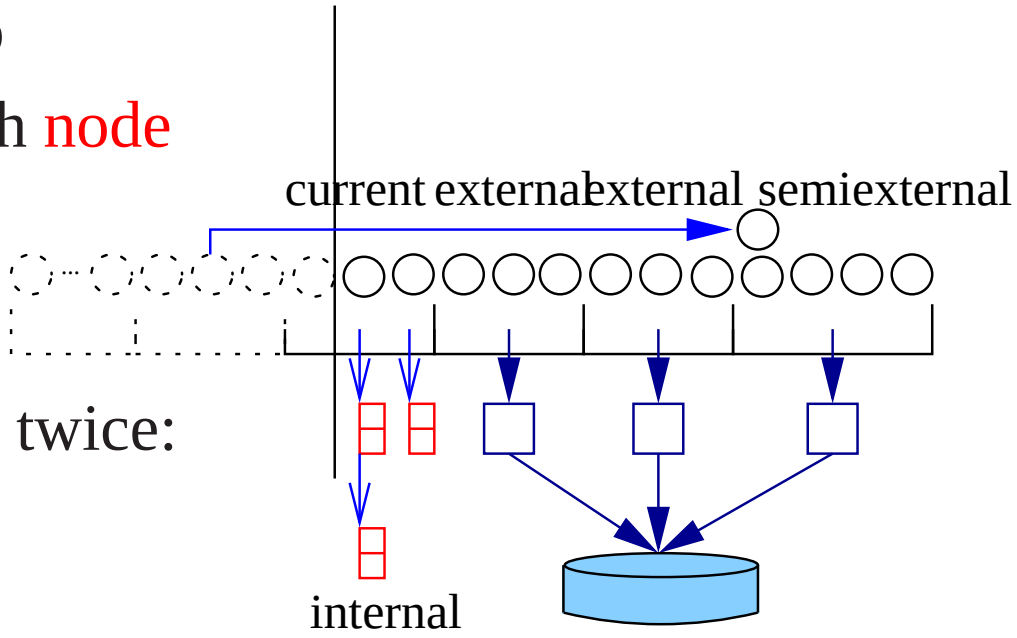


$\approx \text{sort}\left(10m \ln \frac{n}{M}\right)$ I/Os with opt. priority queues [Sanders 00]

Problem: Compute bound

Sweeping with linear internal work

- Assume $m = o\left(M^2/B\right)$
- $k = \Theta(M/B)$ external buckets with n/k nodes each
- M nodes for last "semiexternal" bucket
- split current bucket into internal buckets for each node



Sweeping:

Scan current internal bucket twice:

1. Find minimum
2. Relink

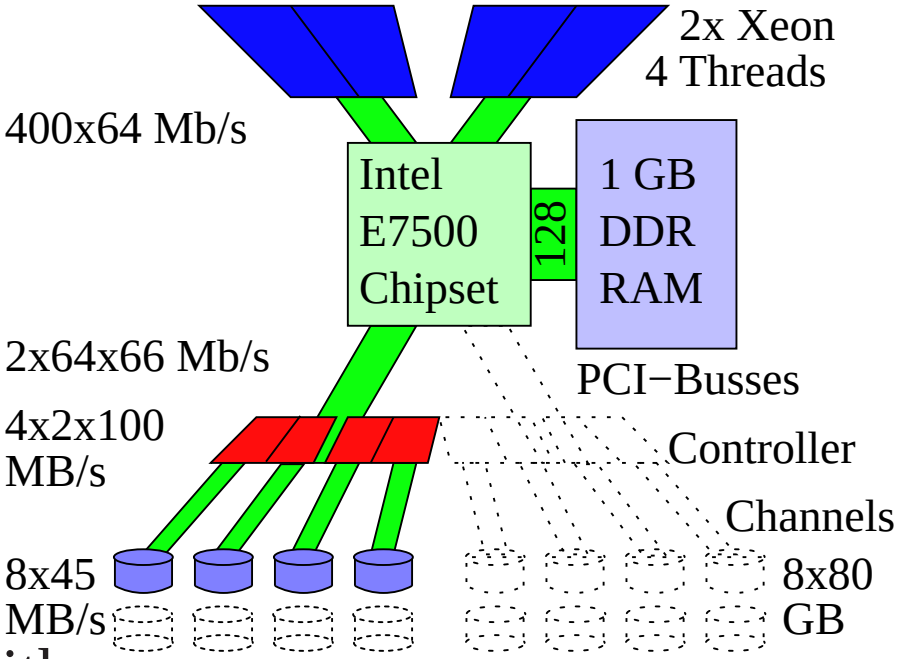
New external bucket: scan and put in internal buckets

Large degree nodes: move to semiexternal bucket

Experiments

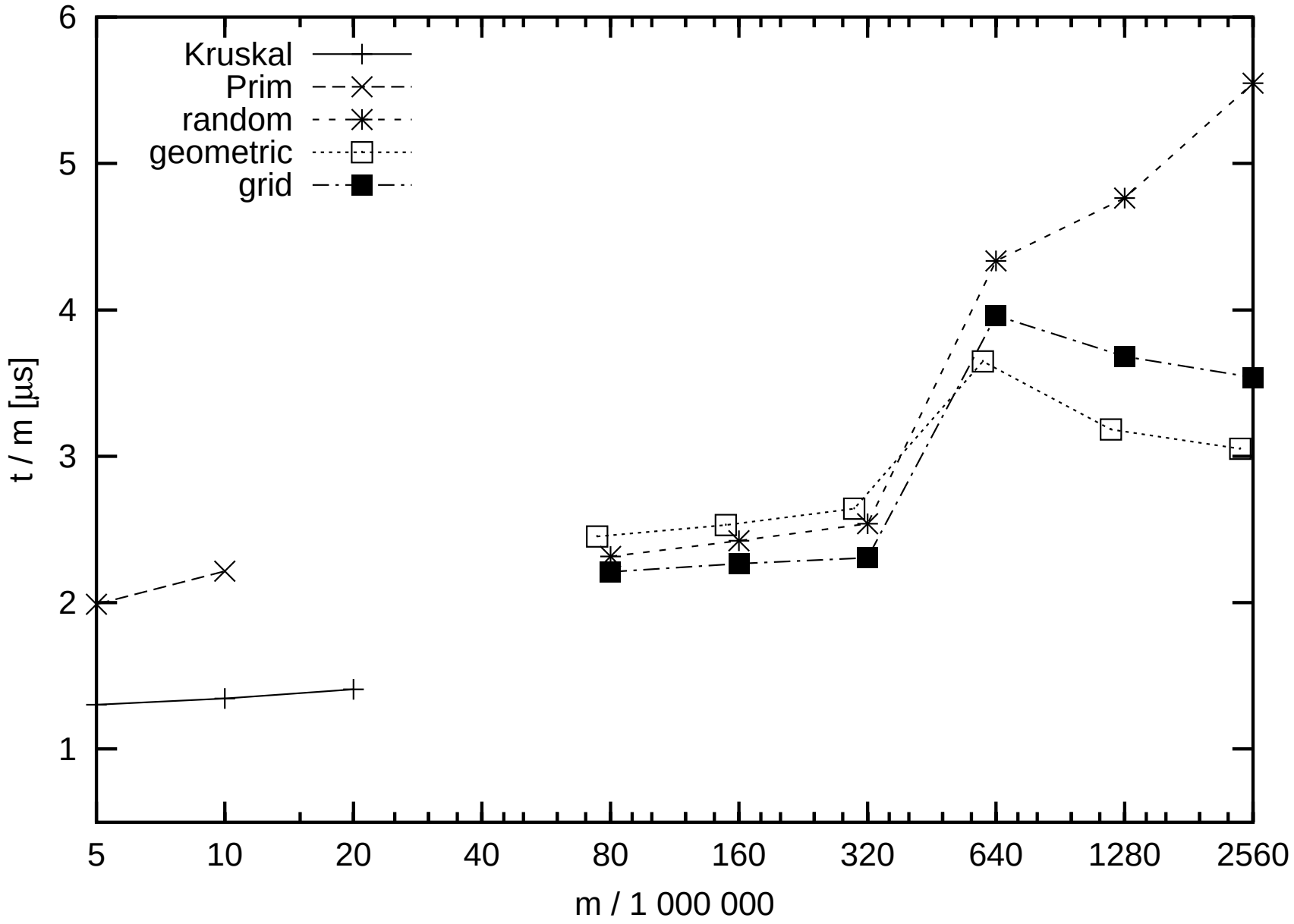
Instances from “classical” MST study [Moret Shapiro 1994]

- sparse random graphs
- random geometric graphs
- grids
  $O(\text{sort}(m))$ I/Os
  for planar graphs by
  removing parallel edges!

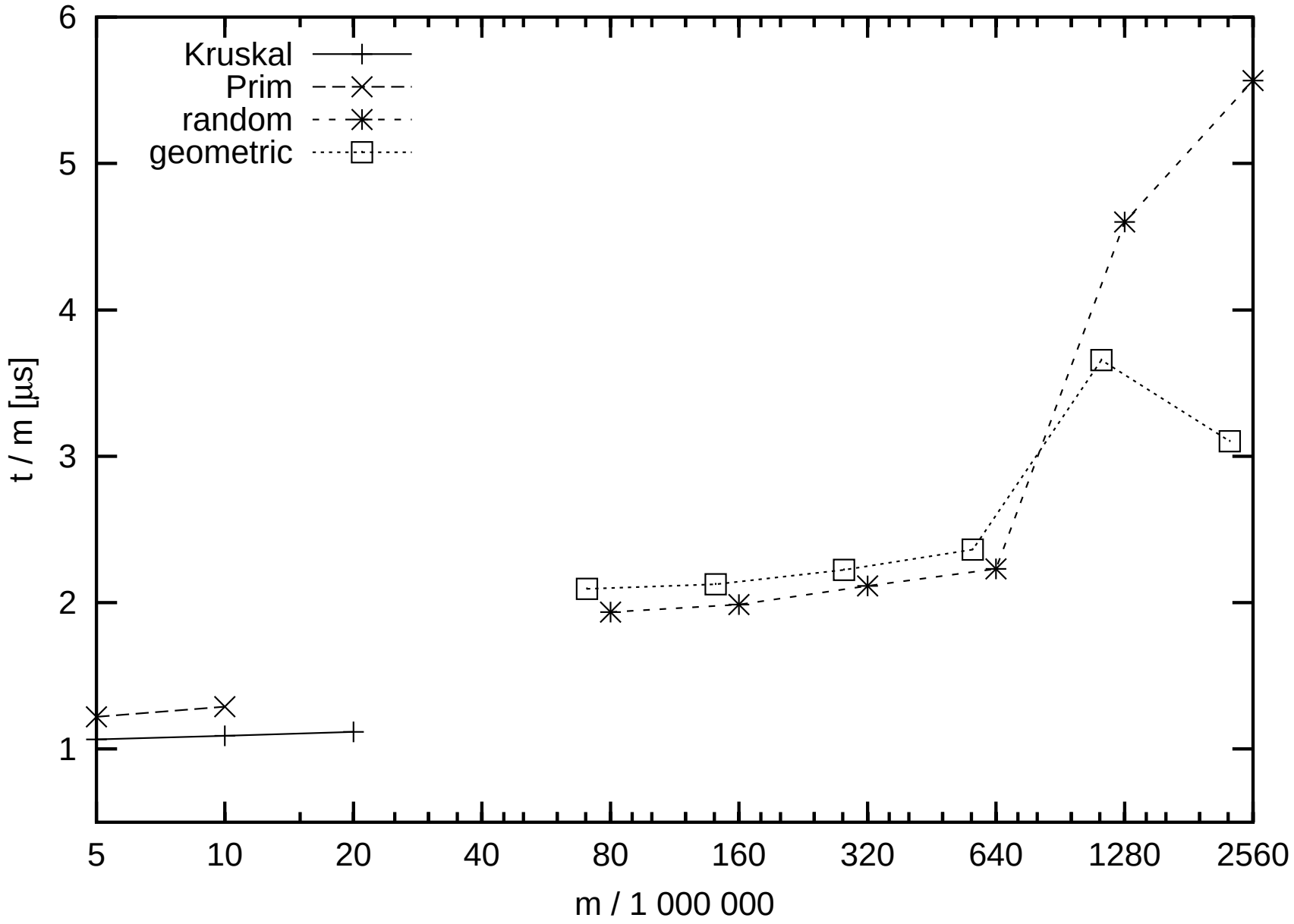


Other instances are rather dense or designed to fool specific algorithms.

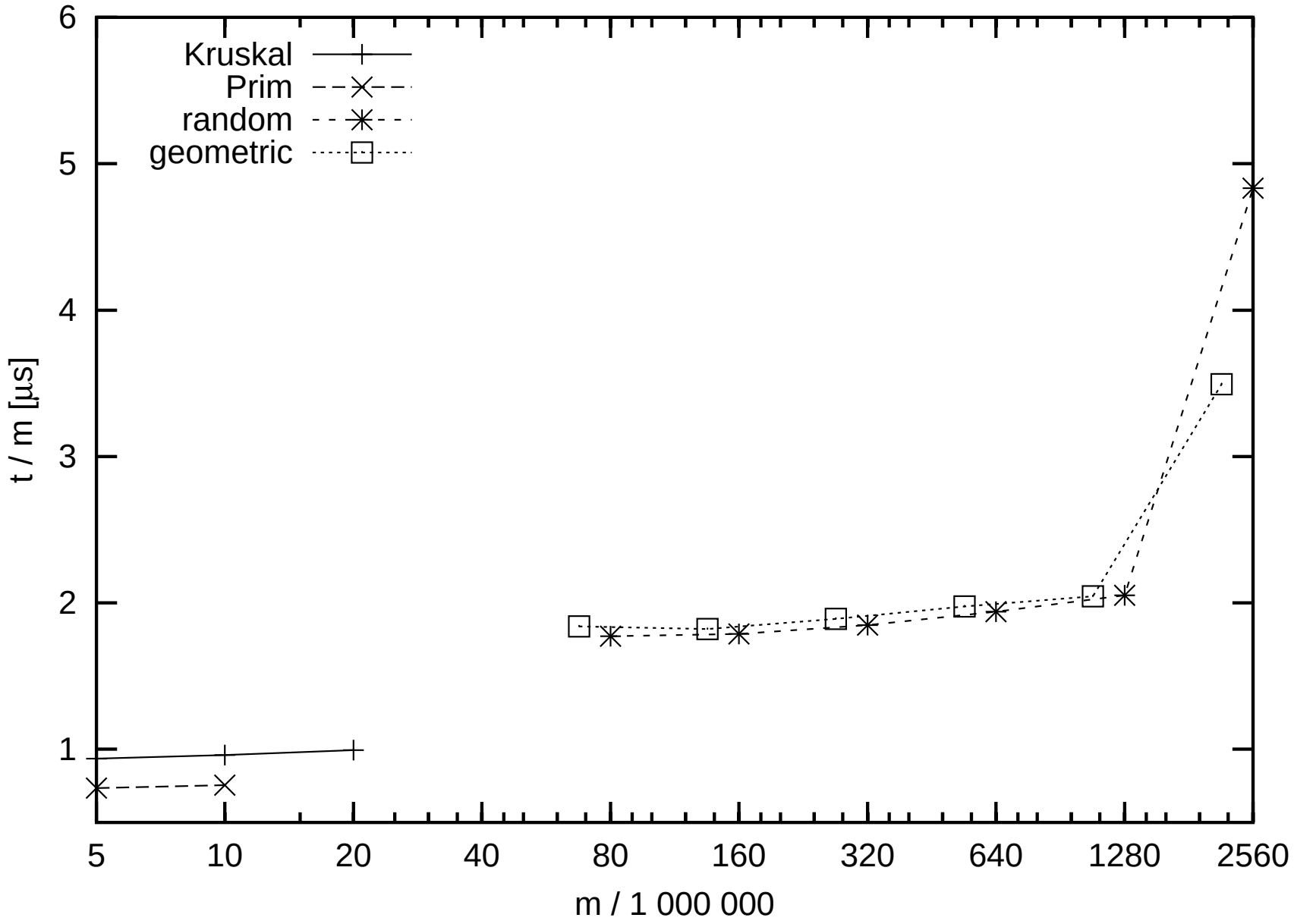
$m \approx 2n$


Kruskal
Prim
random
geometric
grid
t / m [µs]
m / 1 000 000
1
2
3
4
5
6
5
10
20
40
80
160
320
640
1280
2560

$m \approx 4n$


Kruskal
Prim
random
geometric
t / m [µs]
m / 1 000 000
1
2
3
4
5
6
5
10
20
40
80
160
320
640
1280
2560

$m \approx 8n$


Kruskal
Prim
random
geometric
t / m [µs]
m / 1 000 000
1
2
3
4
5
6
5
10
20
40
80
160
320
640
1280
2560

MST Summary

- Edge reduction helps for very dense, “hard” graphs
- A fast and simple node reduction algorithm:
  $(48m + 54n)/B$ **vs.** $\ln(2) \cdot 16m/B \approx 11.1m/B$ I/Os for 2-reduction

$\rightsquigarrow$ $4\times$ less I/Os than previous algorithms

- Refined semiexternal MST, use as base case
- Simple pseudo random permutations (no I/Os)
- A fast implementation
- Experiments with huge graphs (up to $n = 4 \cdot 10^9$ nodes)

External MST is feasible

Open Problems

- New experiments for (improved) Kruskal versus Jarník-Prim
- Realistic (huge) inputs
- Parallel external algorithms
- Implementations for other graph problems

Conclusions

- Even fundamental, “simple” algorithmic problems still raise interesting questions
- Implementation and experiments are important and were neglected by parts of the algorithms community
- Theory an (at least) equally important, essential component of the algorithm design process

More Algorithm Engineering on Graphs

- Count triangles in very large graphs. Interesting as a measure of clusteredness. (Cooperation with AG Wagner)
- External BFS (Master thesis Deepak Ajwani)
- Maximum flows: Is the theoretical best algorithm any good? (Jein)
- Approximate max. weighted matching (Studienarbeit Jens Maue)

Maximal Flows

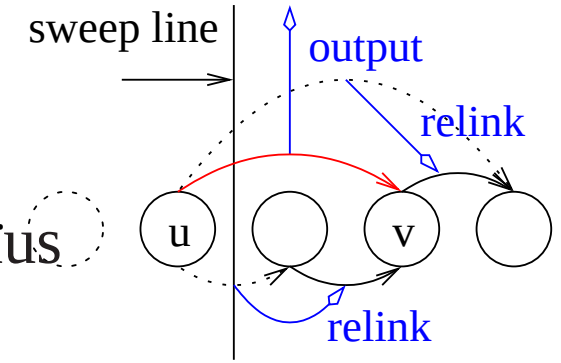
Theory: $O\left(m\Lambda \log(n^2/m) \log U\right)$ binary blocking flow-algorithm mit $\Lambda = \min\{m^{1/2}, n^{2/3}\}$ [Goldberg-Rao-97].

Problem: best case $\approx$ worst case

[Hagerup Sanders Träff WAE 98]:

- Implementable generalization
- best case $\ll$ worst case
- best algorithms for some "difficult" instances

Ergebnis



- Einfach extern implementierbar
- $n' = M \rightsquigarrow$ semiexterner Kruskal Algorithmus
- Insgesamt $O\left(\text{sort}\left(m \ln \frac{n}{m}\right)\right)$ erwartete I/Os
- Für realistische Eingaben mindestens $4\times$ besser als bisher bekannte Algorithmen
- Implementierung in `<stxxl>` mit bis zu 96 GByte gro/3en Graphen läuft „über Nacht“

Presenting Data from Experiments in Algorithmics

Restrictions

- black and white $\leadsto$ easy and cheap printing
- 2D (stay tuned)
- no animation
- no realism desired

Not here

- ensuring reproducibility
- describing the setup
- finding sources of measurement errors
- reducing measuremnt errors (averaging, median,unloaded machine. . .)
- measurements in the creative phase of experimental algorithmics.

The Starting Point

- (Several) Algorithm(s)
- A few quantities to be measured: time, space, solution quality, comparisons, cache faults,... There may also be measurement errors.
- An unlimited number of potential inputs. $\rightsquigarrow$ condense to a few characteristic ones (size, $|V|$, $|E|$, ... or problem instances from applications)

Usually there is not a lack but an abundance of data $\neq$ many other sciences

The Process

Waterfall model?

1. Design
2. Measurement
3. Interpretation

Perhaps the paper should at least look like that.

The Process

- Eventually stop asking questions (Advisors/Referees listen !)
- build measurement tools
- automate (re)measurements
- Choice of Experments driven by risk and opportunity
- Distinguish mode

  explorative: many different parameter settings, interactive, short turnaround times

  consolidating: many large instances, standardized measurement conditions, batch mode, many machines

Of Risks and Opportunities

Example: Hypothesis = my algorithm is the best

big risk: untried main competitor

small risk: tuning of a subroutine that takes 20 % of the time.

big opportunity: use algorithm for a new application

$\rightsquigarrow$ new input instances

Basic Principles

- Minimize nondata ink
  (form follows function, not a beauty contest,...)
- Letter size $\approx$ surrounding text
- Avoid clutter and overwhelming complexity
- Avoid boredom (too little data per m^2.
- Make the conclusions evident

Tables

- \+ easy
- – easy $\rightsquigarrow$ overuse
- \+ accurate values ($\neq$ 3D)
- \+ more compact than bar chart
- \+ good for unrelated instances (e.g. solution quality)
- – boring
- – no visual processing

rule of thumb that “tables usually outperform a graph for small data sets of 20 numbers or less” [Tufte 83]

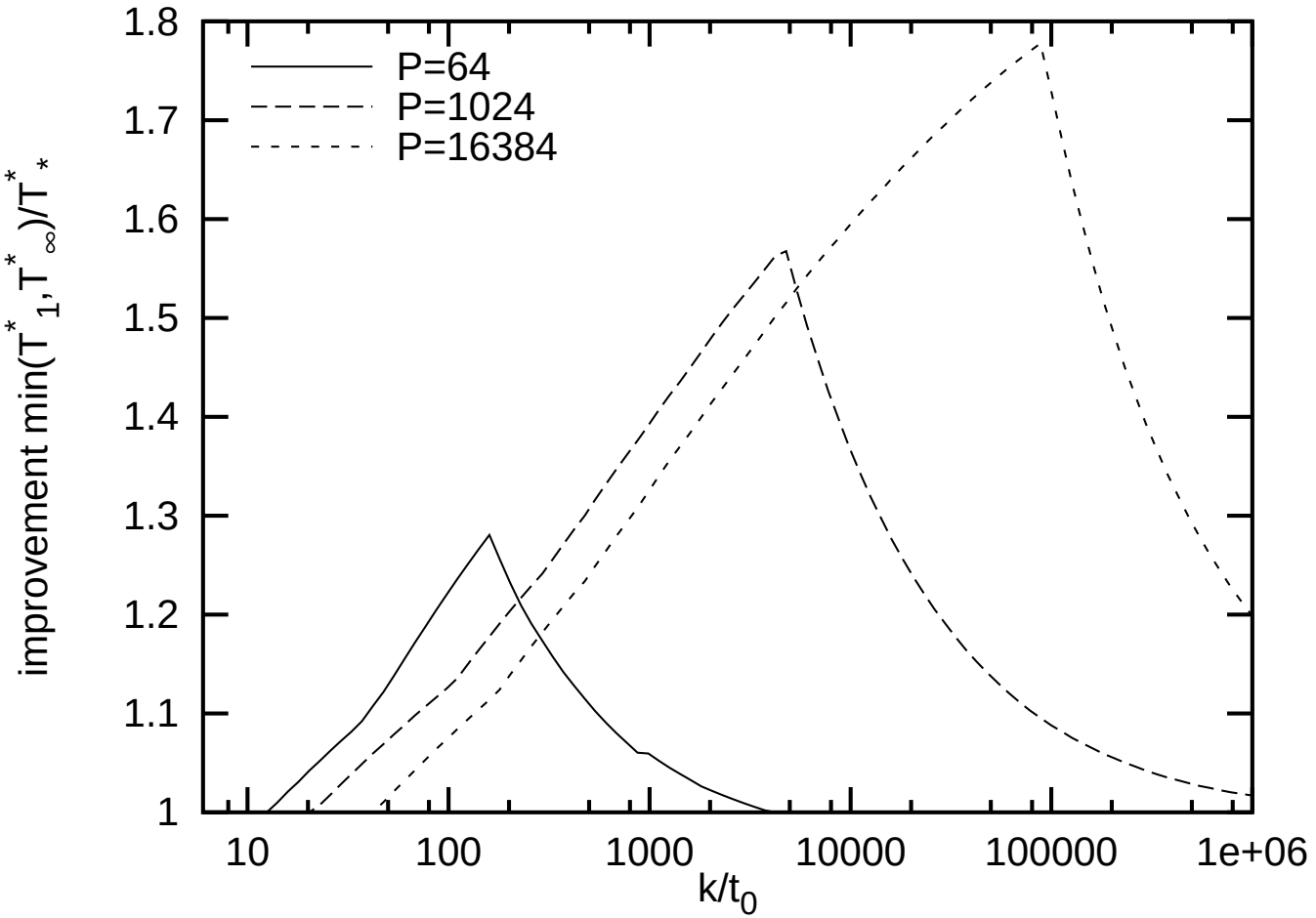
Curves in main paper, tables in appendix?

2D Figures

default: $x =$ input size, $y = f($execution time$)$

x Axis

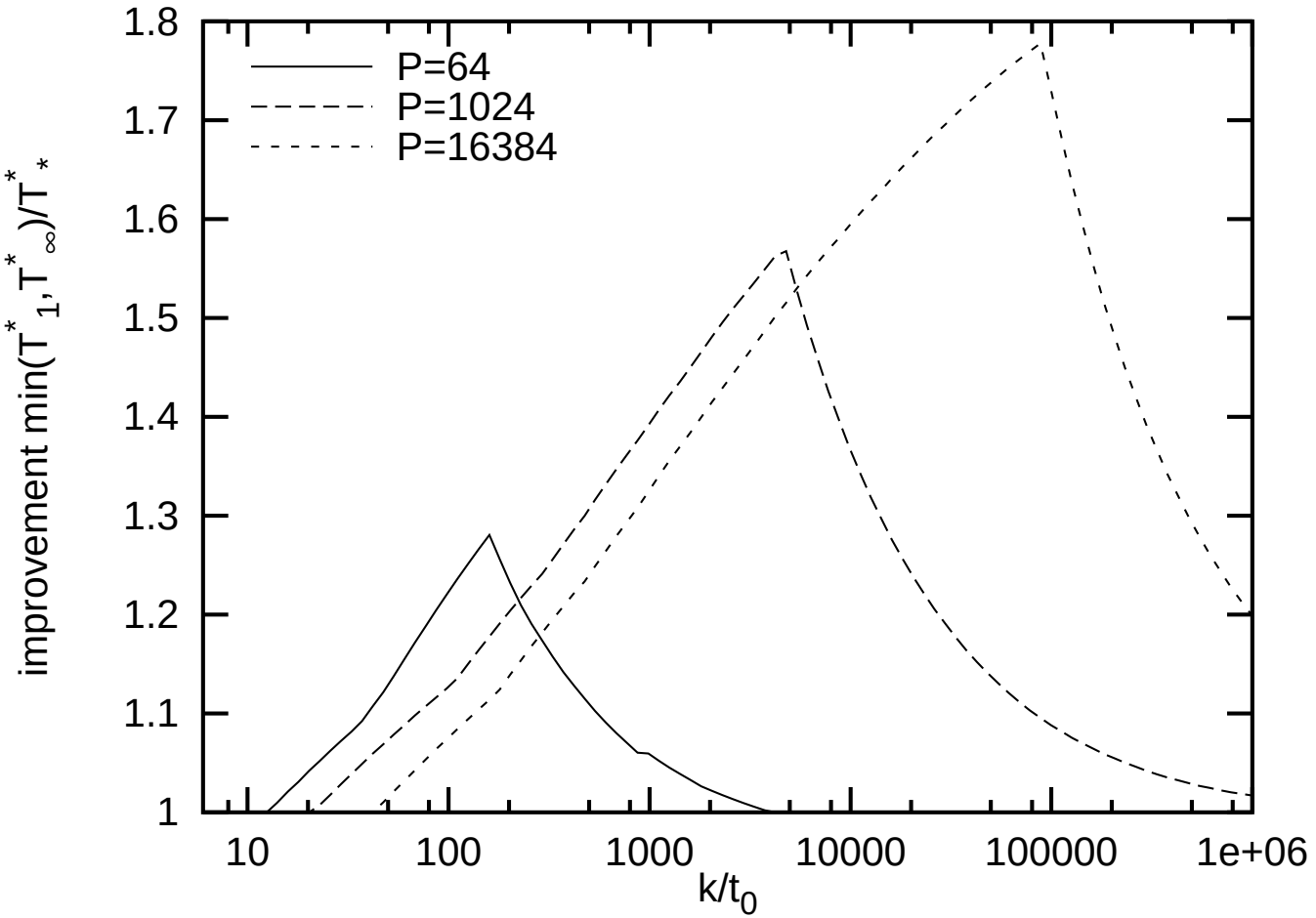
Choose unit to eliminate a parameter?



length k fractional tree broadcasting. latency $t_0 + k$

x Axis

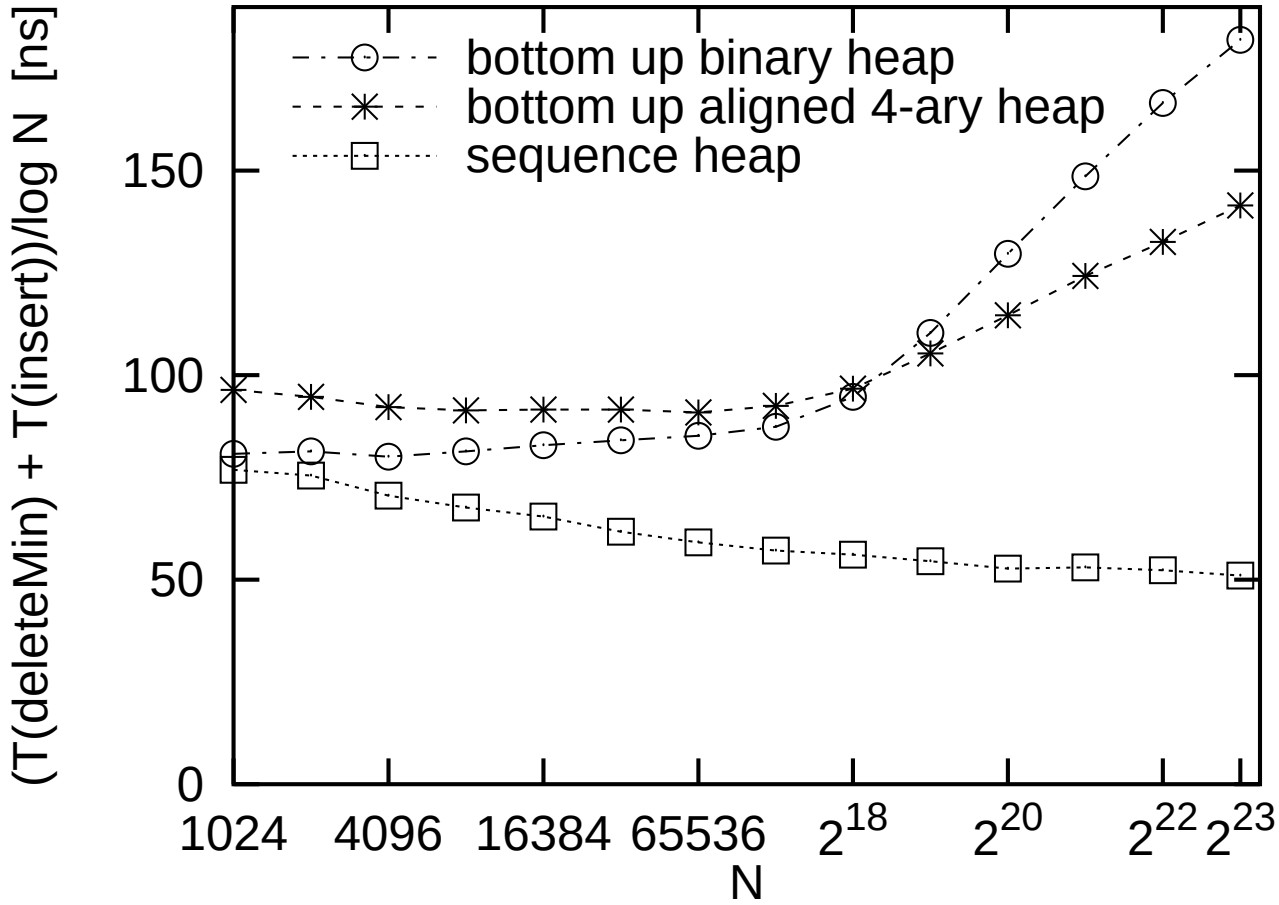
logarithmic scale?



yes if *x* range is wide

x Axis

logarithmic scale, powers of two (or $\sqrt{2}$)



with tic marks, (plus a few small ones)

gnuplot

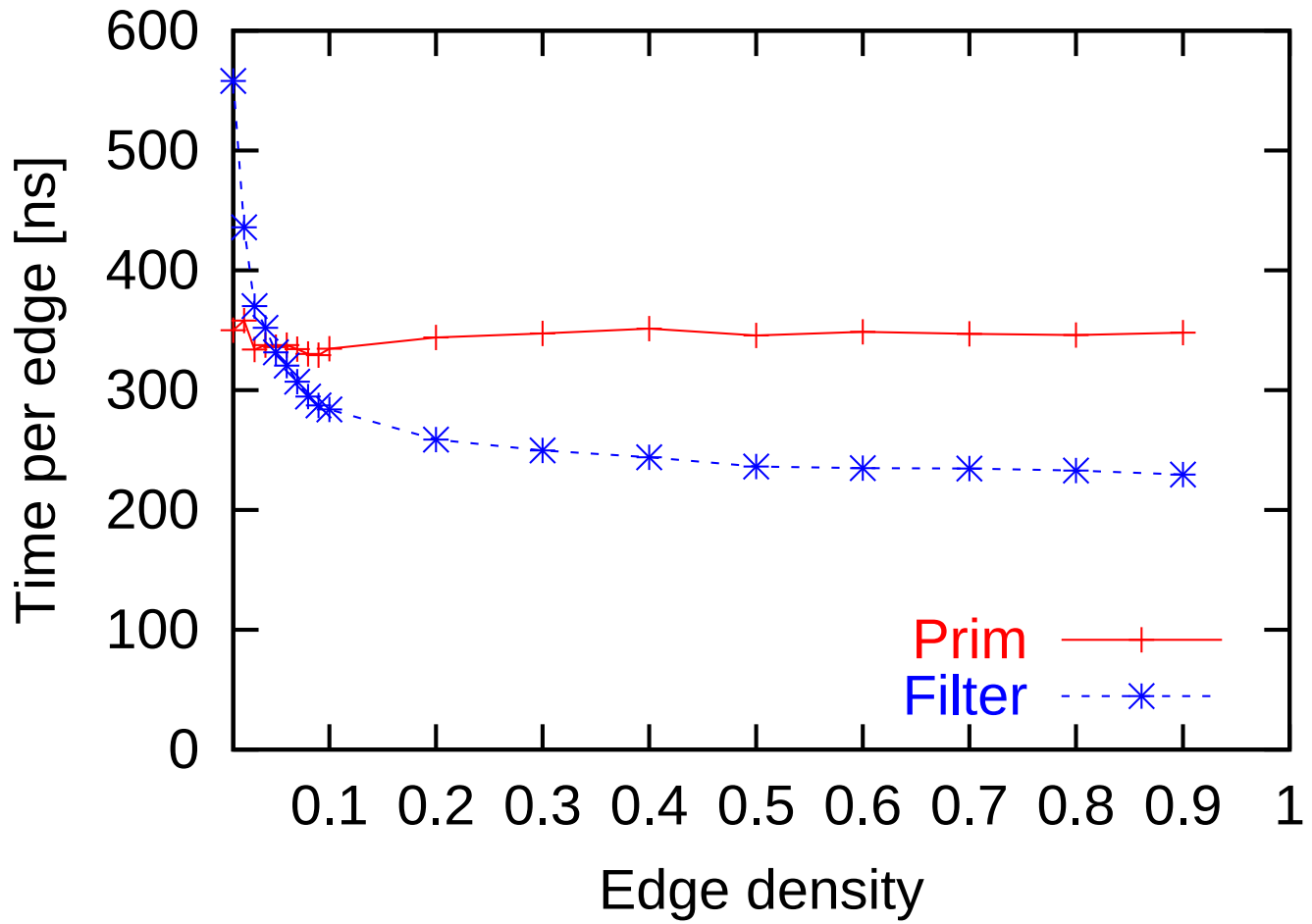
```
set xlabel "N"
set ylabel "(time per operation)/log N  [ns]"
set xtics (256, 1024, 4096, 16384, 65536, "2^{18}" 262144,
set size 0.66, 0.33
set logscale x 2
set data style linespoints
set key left
set terminal postscript portrait enhanced 10
set output "r10000timenew.eps"
plot [1024:10000000][0:220]\
 "h2r10000new.log" using 1:3 title "bottom up binary heap
 "h4r10000new.log" using 1:3 title "bottom up aligned 4-a
 "knr10000new.log" using 1:3 title "sequence heap" with l
```

Data File

```
256 703.125 87.8906
512 729.167 81.0185
1024 768.229 76.8229
2048 830.078 75.4616
4096 846.354 70.5295
8192 878.906 67.6082
16384 915.527 65.3948
32768 925.7 61.7133
65536 946.045 59.1278
131072 971.476 57.1457
262144 1009.62 56.0902
524288 1035.69 54.51
1048576 1055.08 52.7541
2097152 1113.73 53.0349
4194304 1150.29 52.2859
8388608 1172.62 50.9836
```

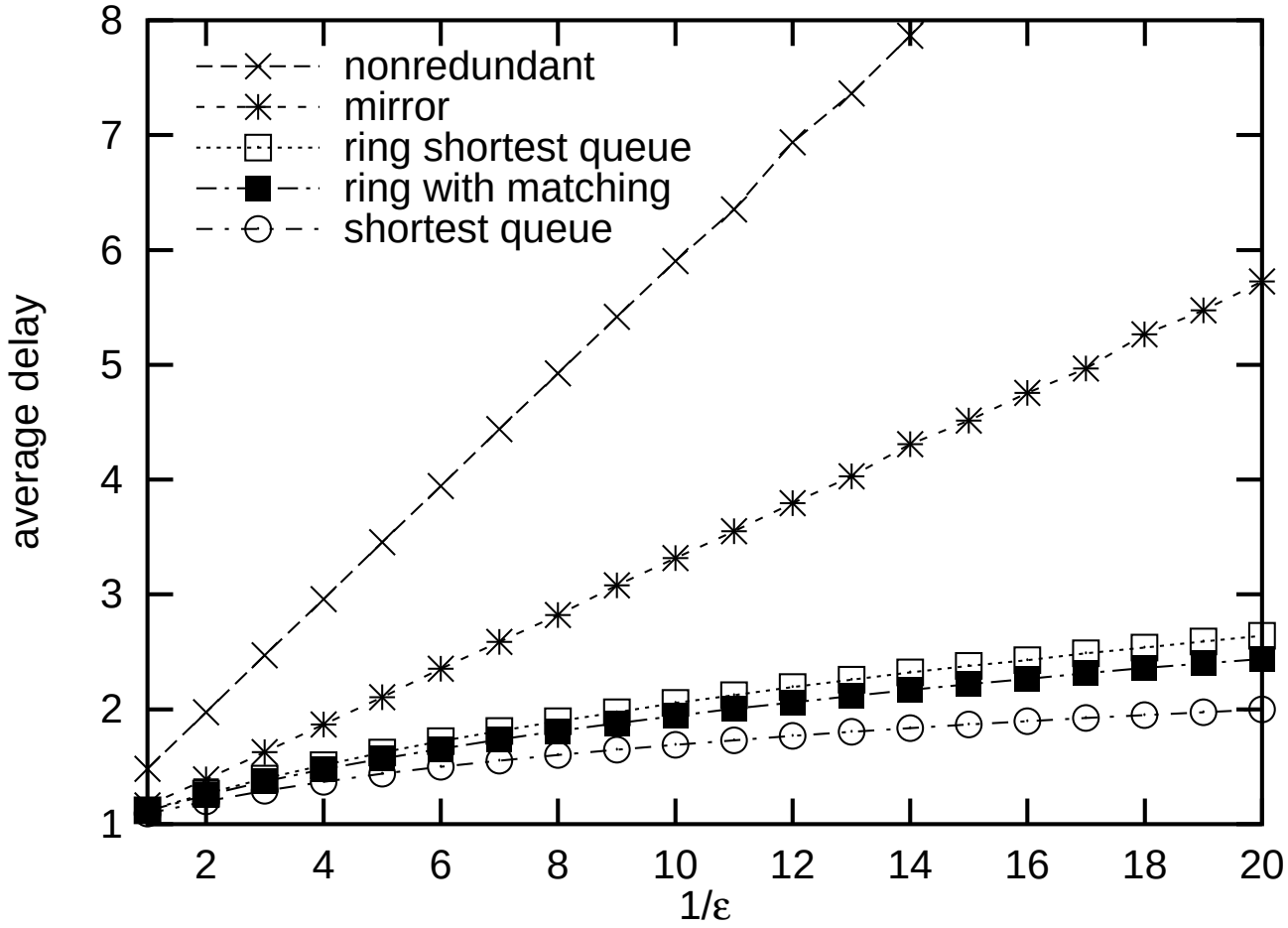
x Axis

linear scale for ratios or small ranges (#processor,. . .)



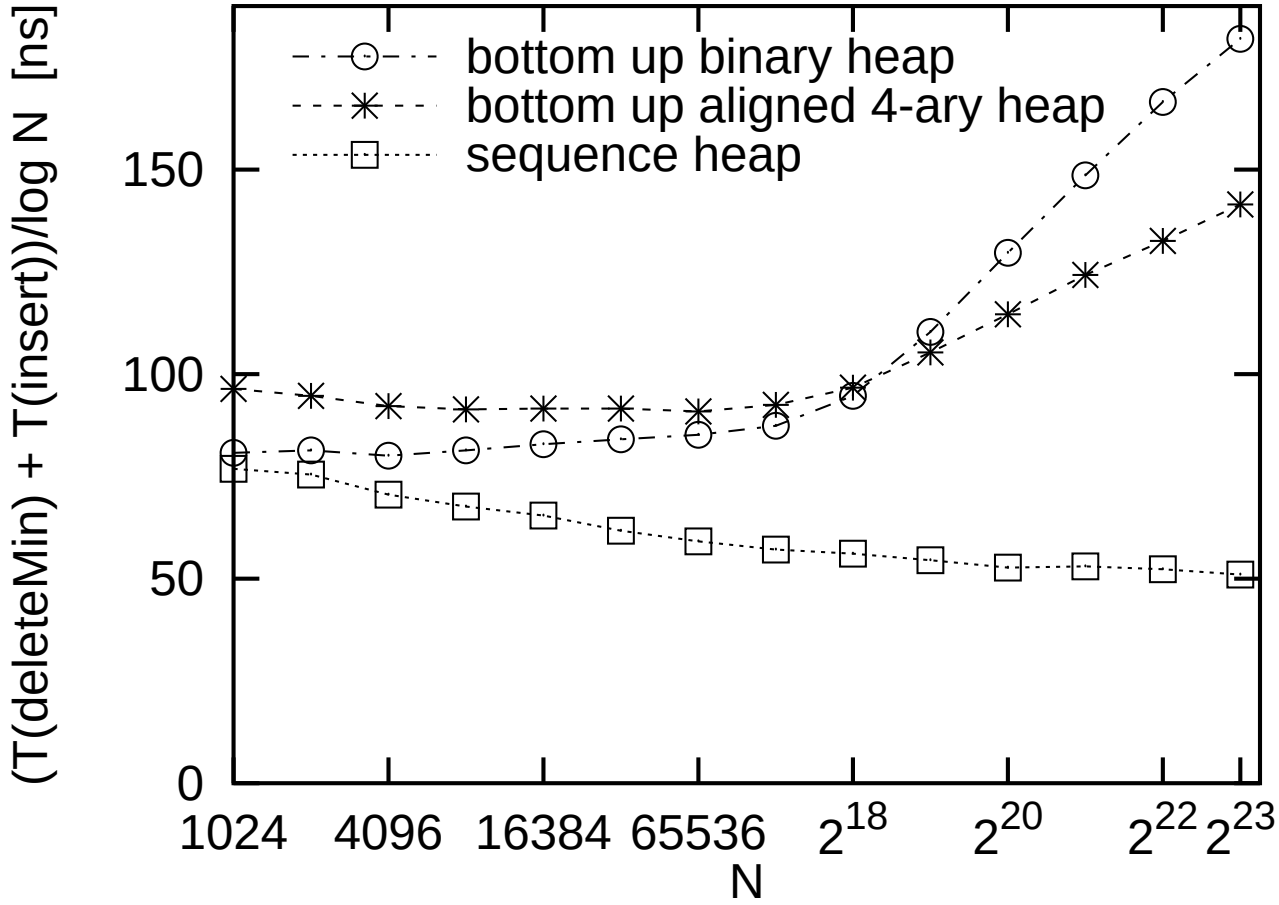
x Axis

An exotic scale: arrival rate $1 - \varepsilon$ of saturation point



y Axis

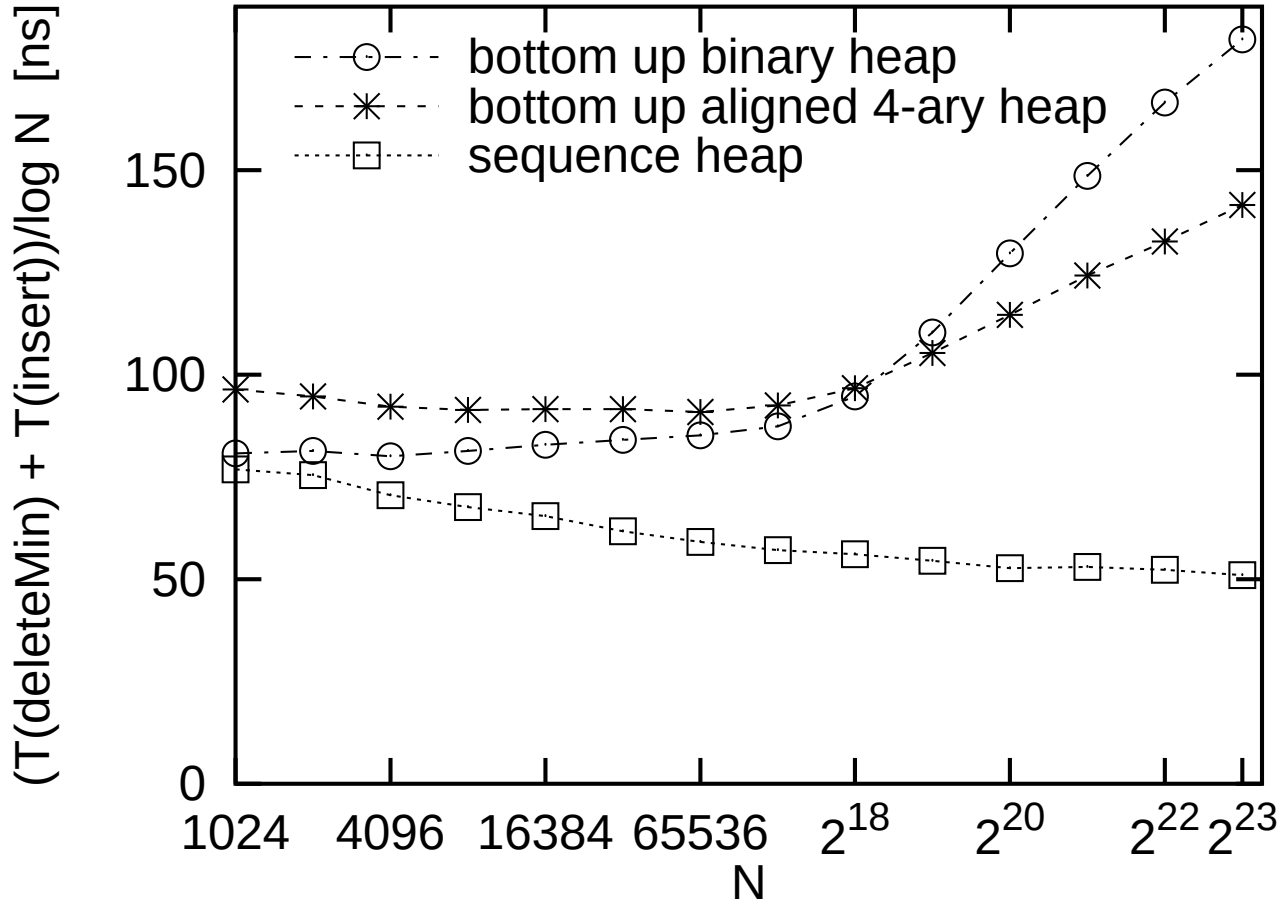
Avoid log scale ! scale such that theory gives ≈ horizontal lines



but give easy interpretation of the scaling function

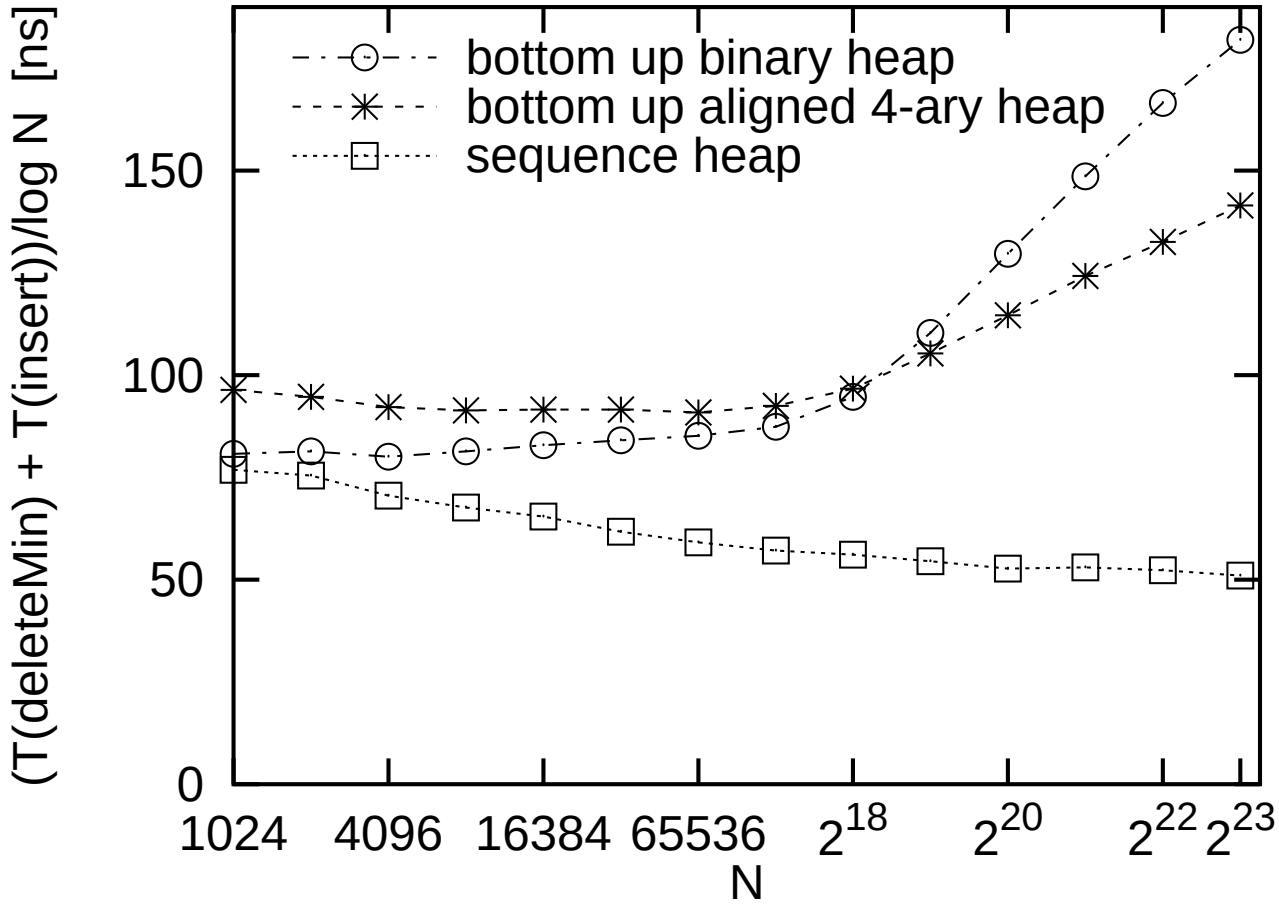
y Axis

give units



y Axis

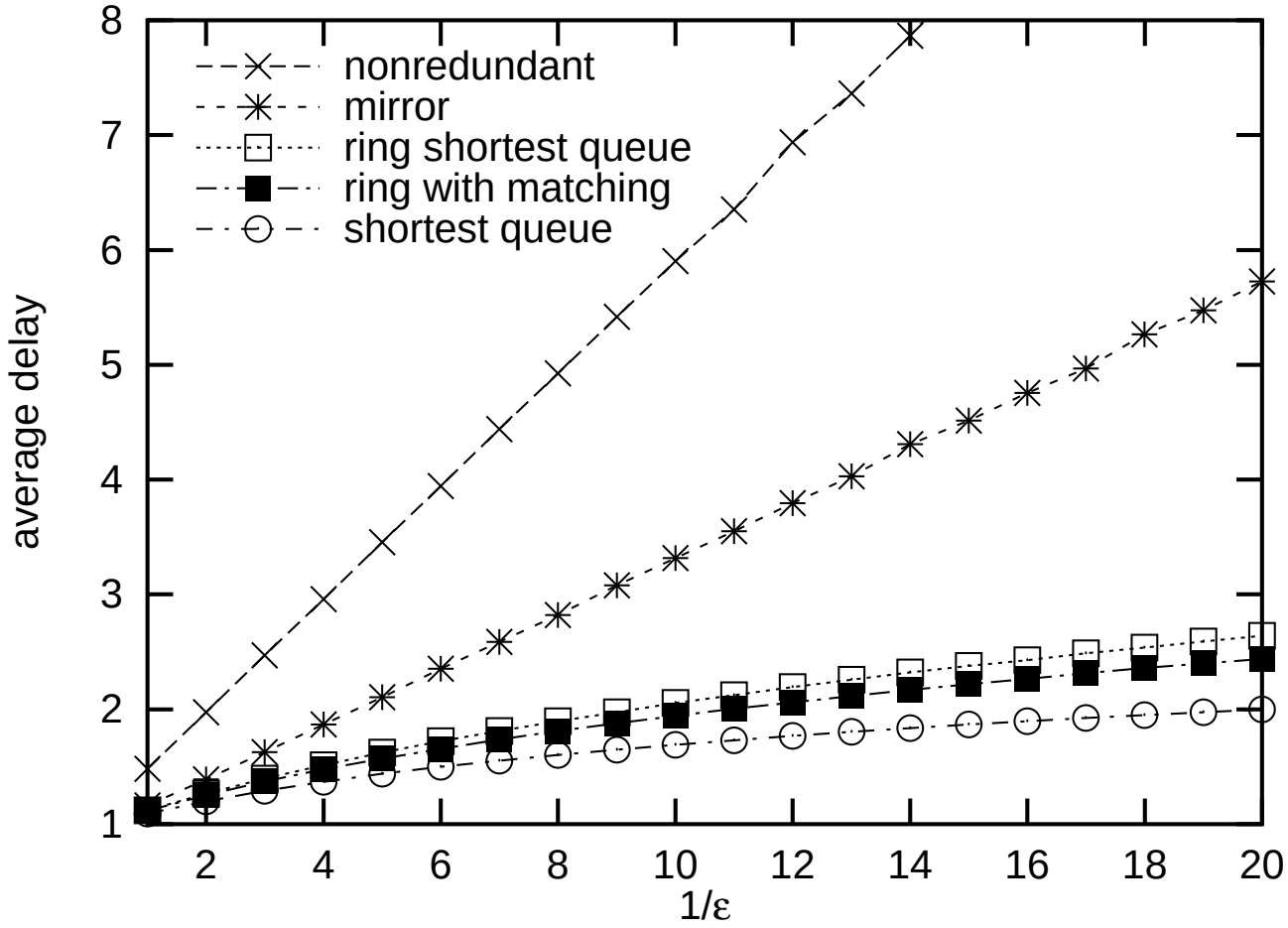
start from 0 if this does not waste too much space



you may assume readers to be out of Kindergarten

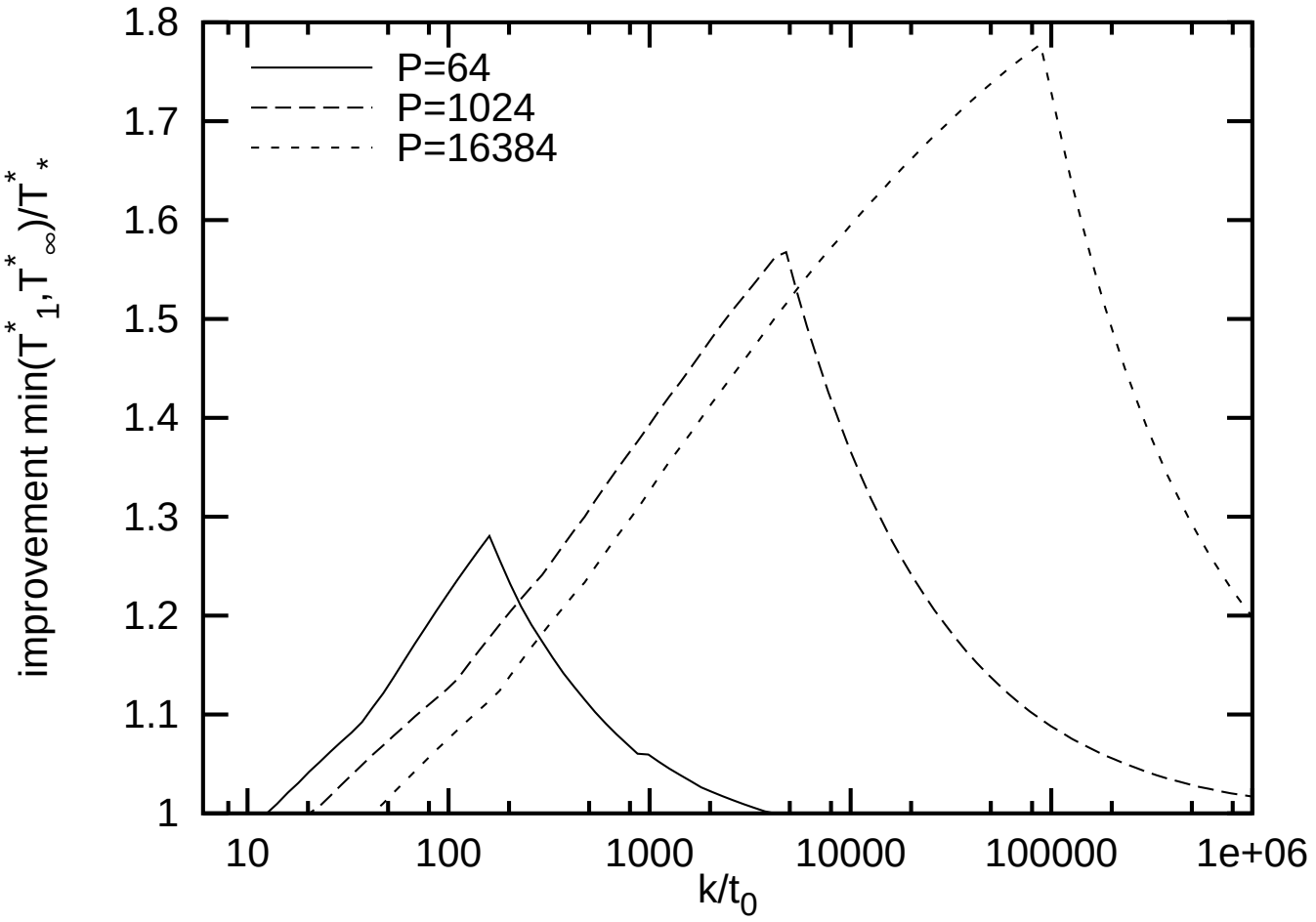
y Axis

clip outclassed algorithms



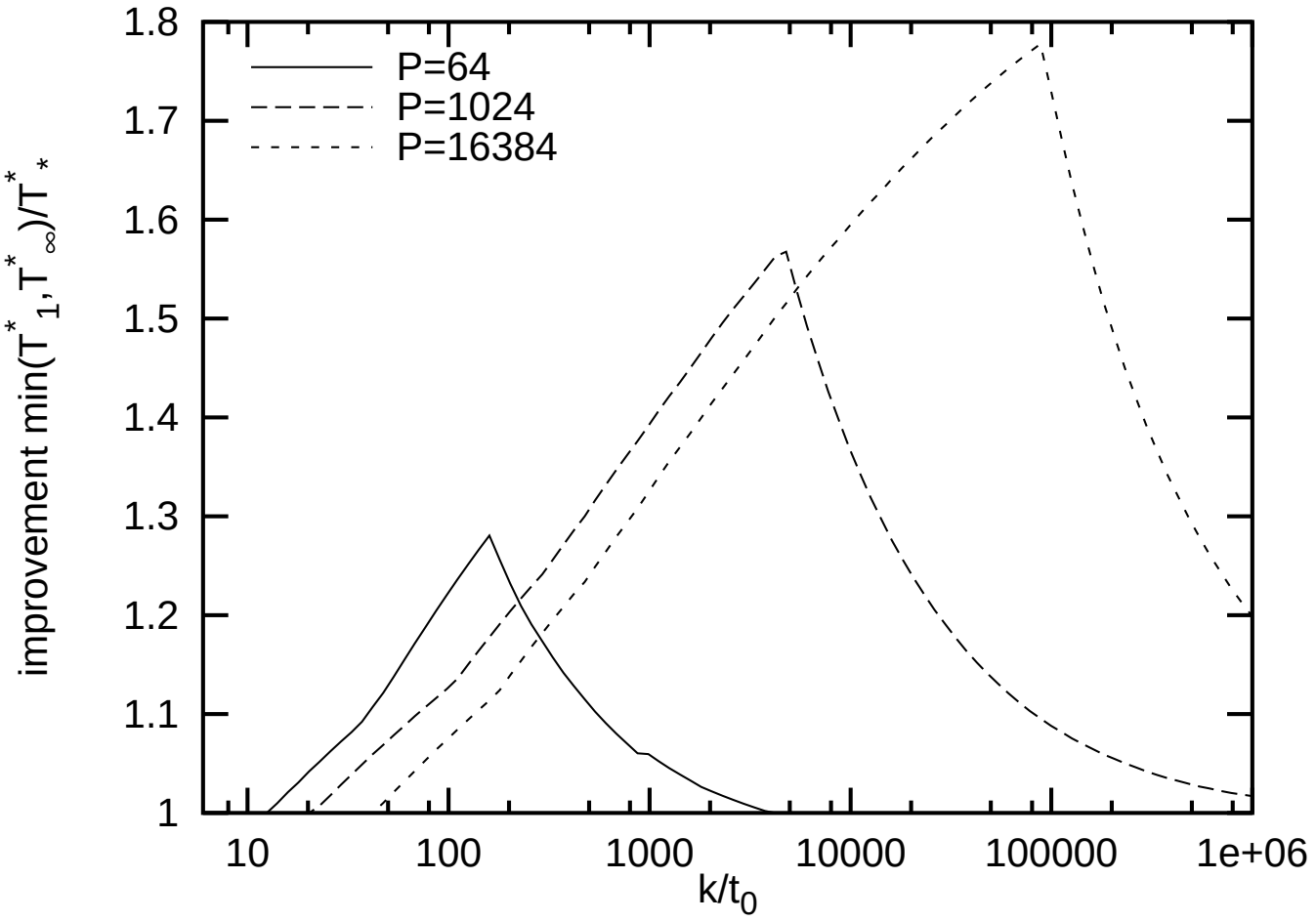
y Axis

vertical size: weighted average of the slants of the line segments in the figure should be about 45° [Cleveland 94]



y Axis

graph a bit wider than high, e.g., golden ratio [Tufte 83]



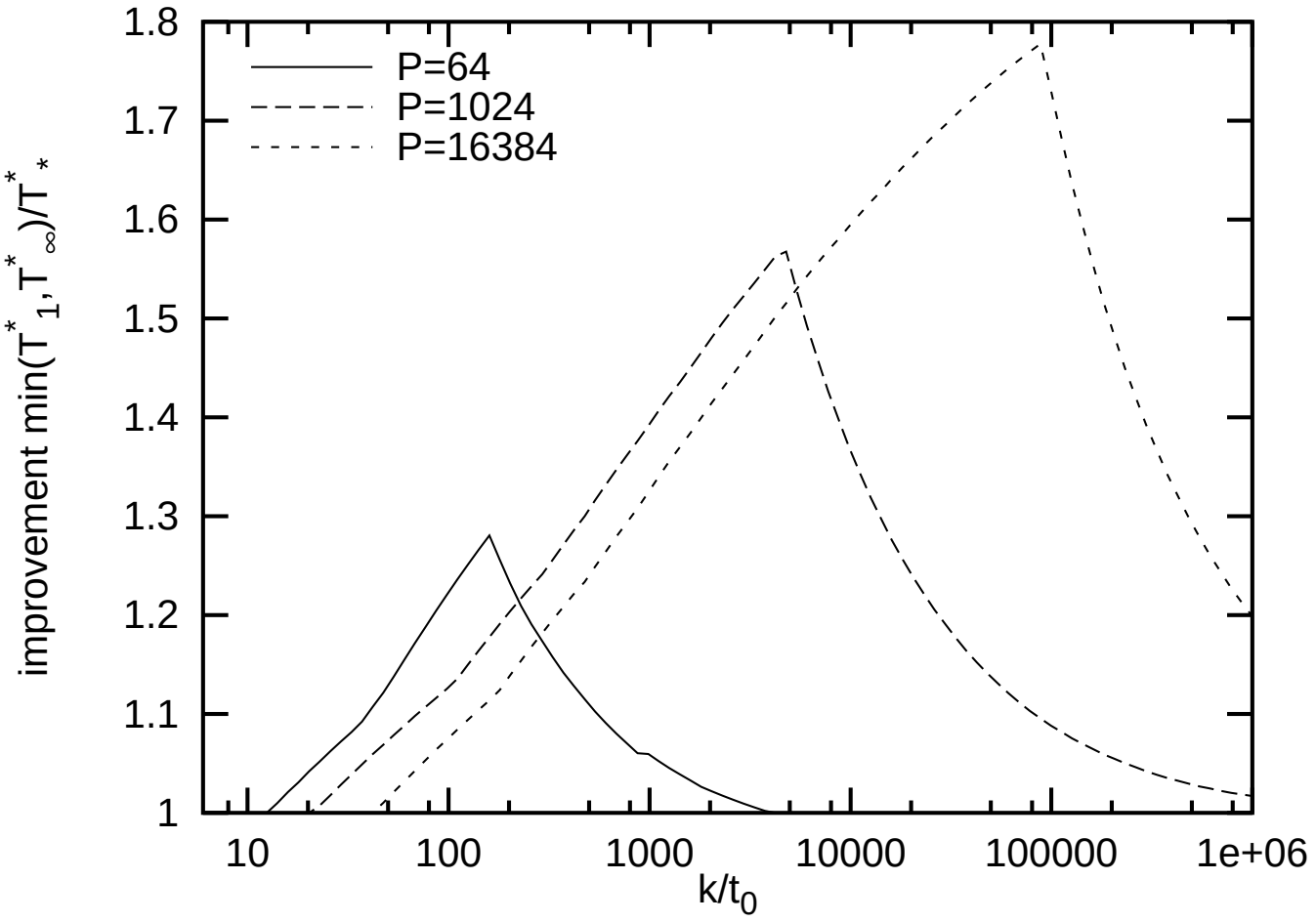
Multiple Curves

- $+$ high information density
- $+$ better than 3D (reading off values)
- $-$ Easily overdone

≤ 7 smooth curves

Reducing the Number of Curves

use ratios



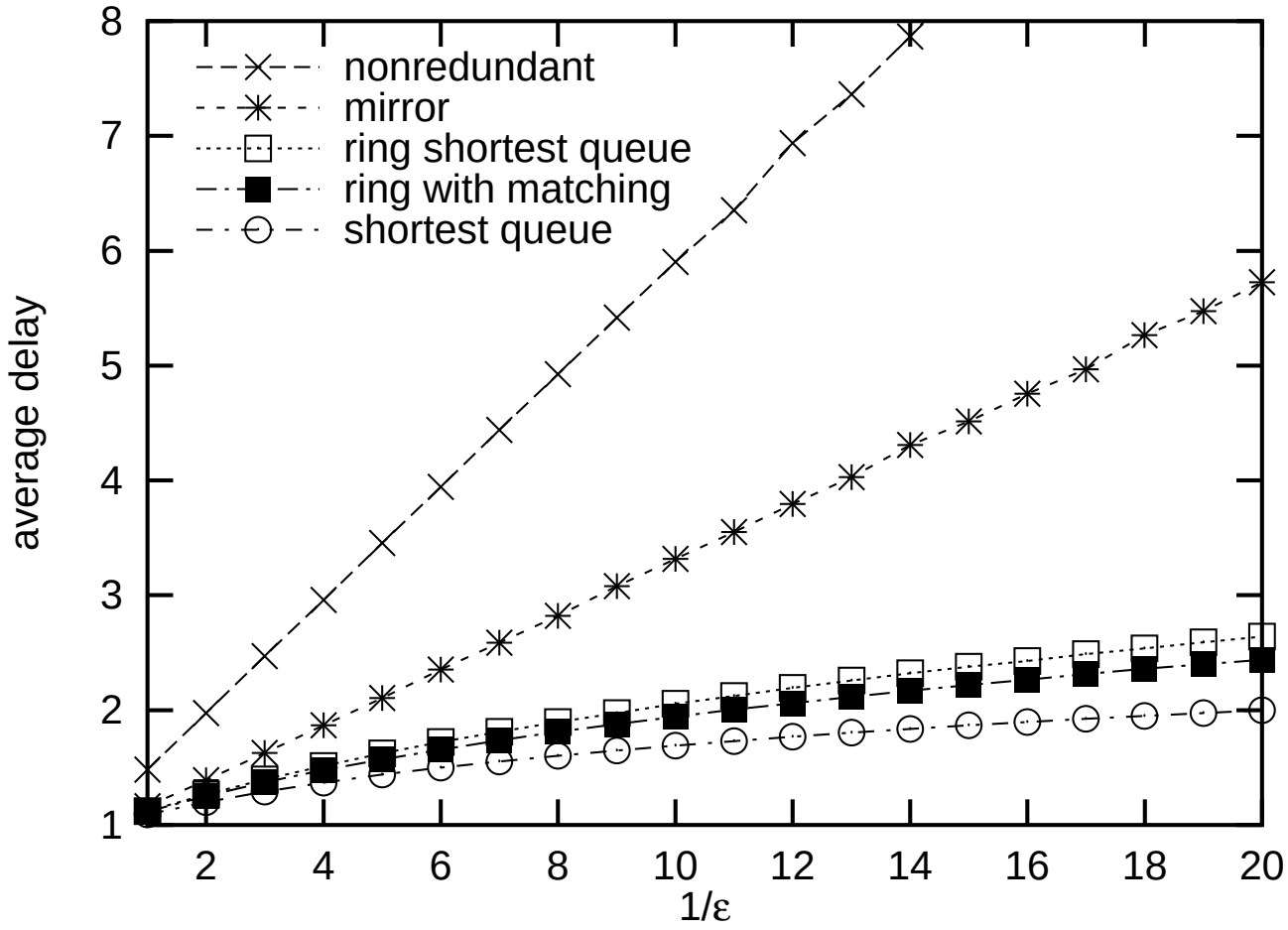
Reducing the Number of Curves

omit curves

- outclassed algorithms (for case shown)
- equivalent algorithms (for case shown)

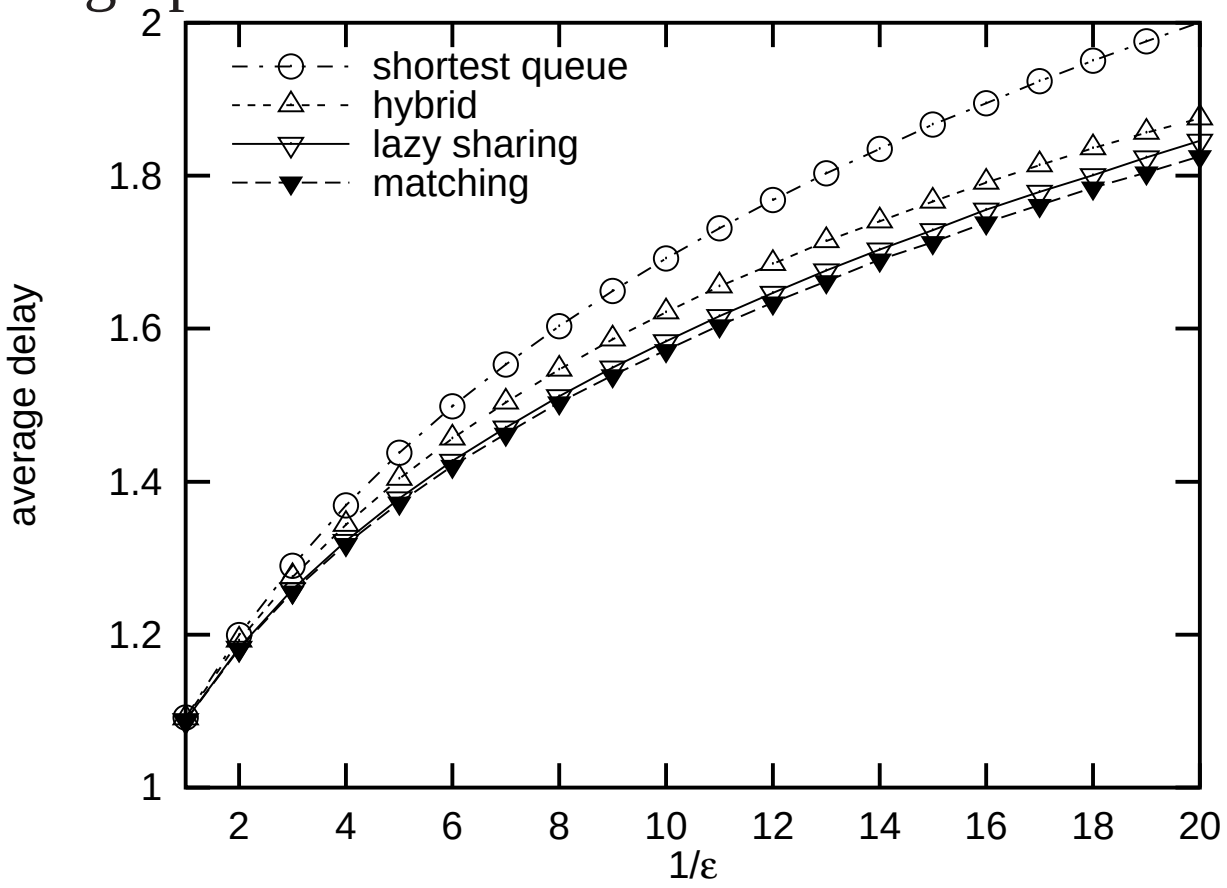
Reducing the Number of Curves

split into two graphs

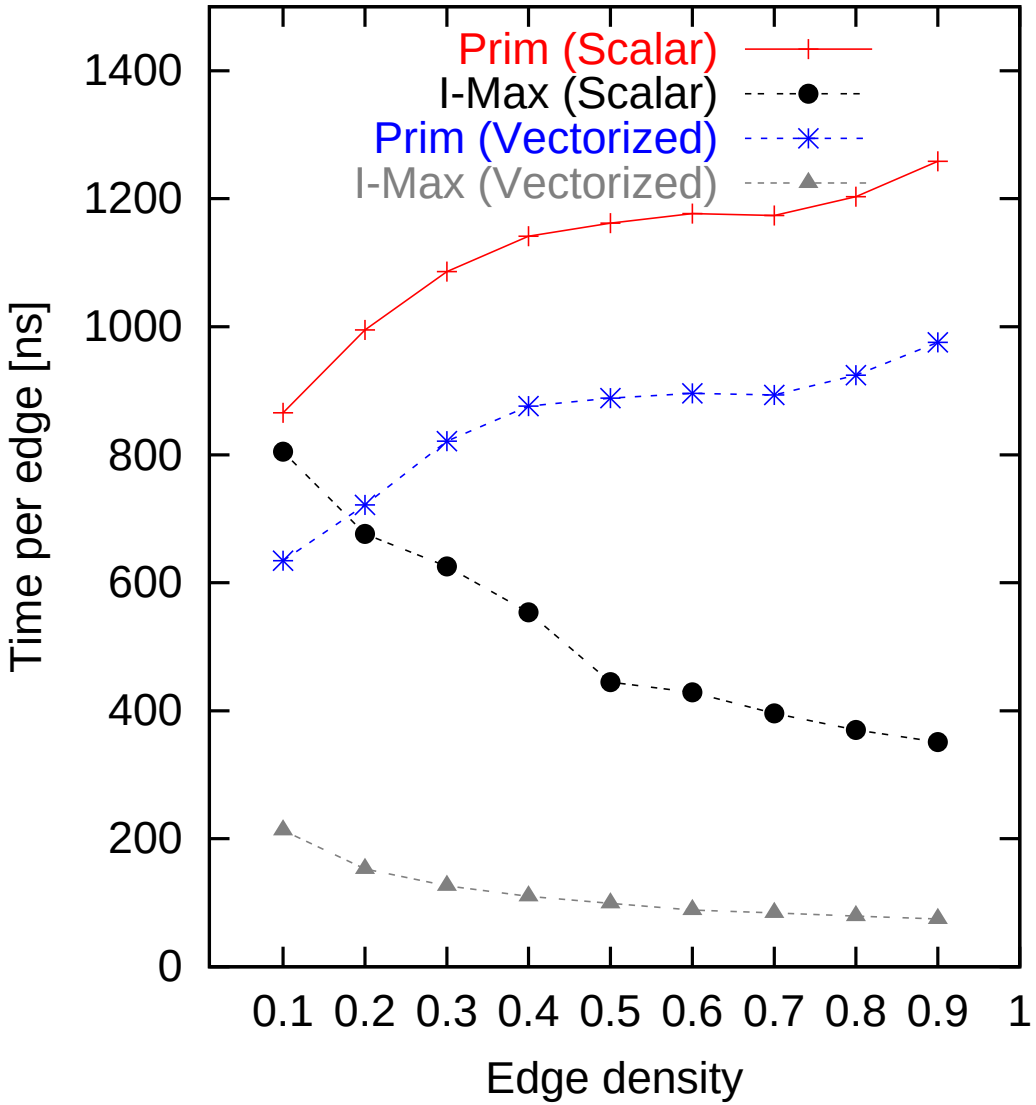


Reducing the Number of Curves

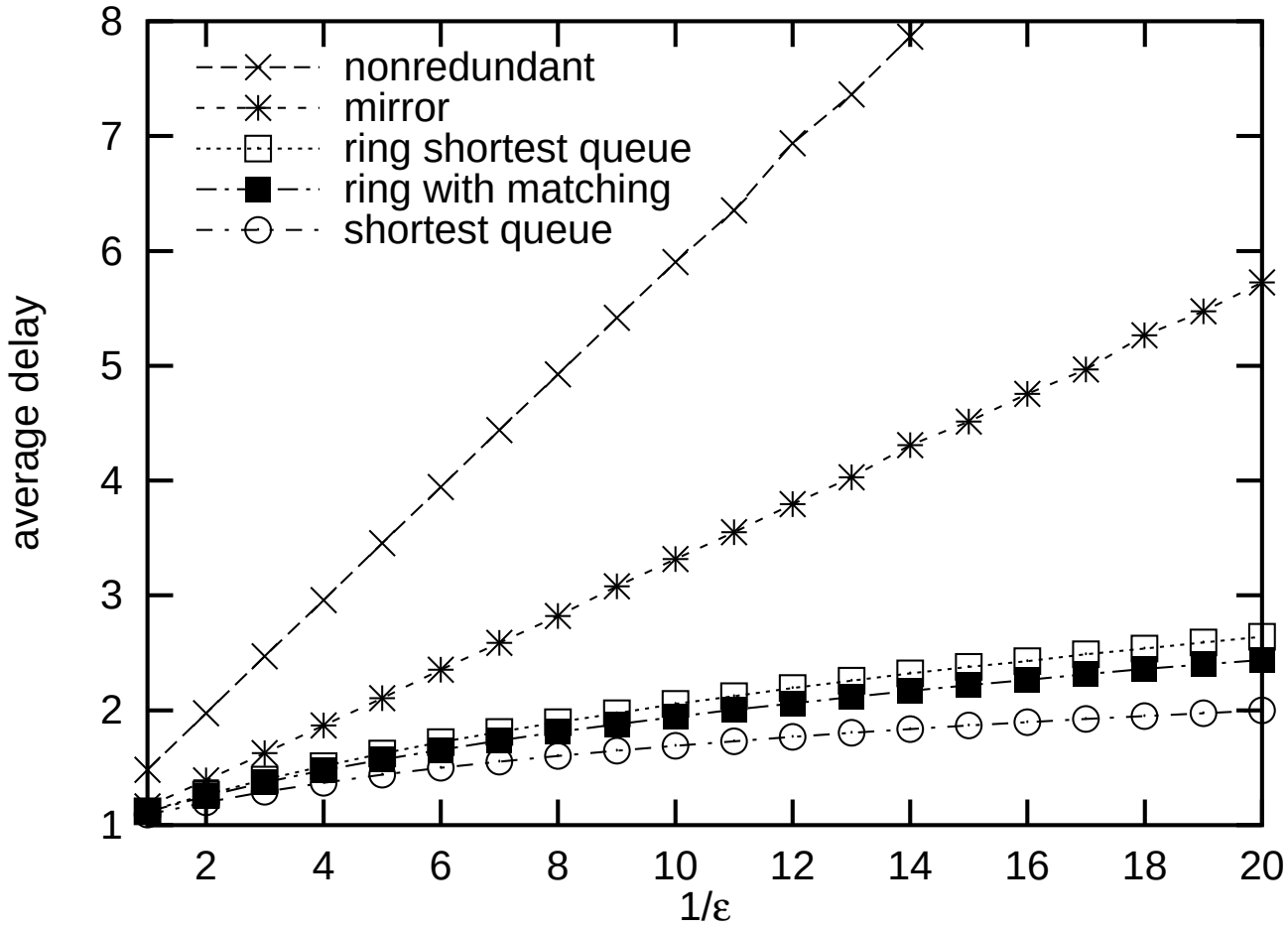
split into two graphs



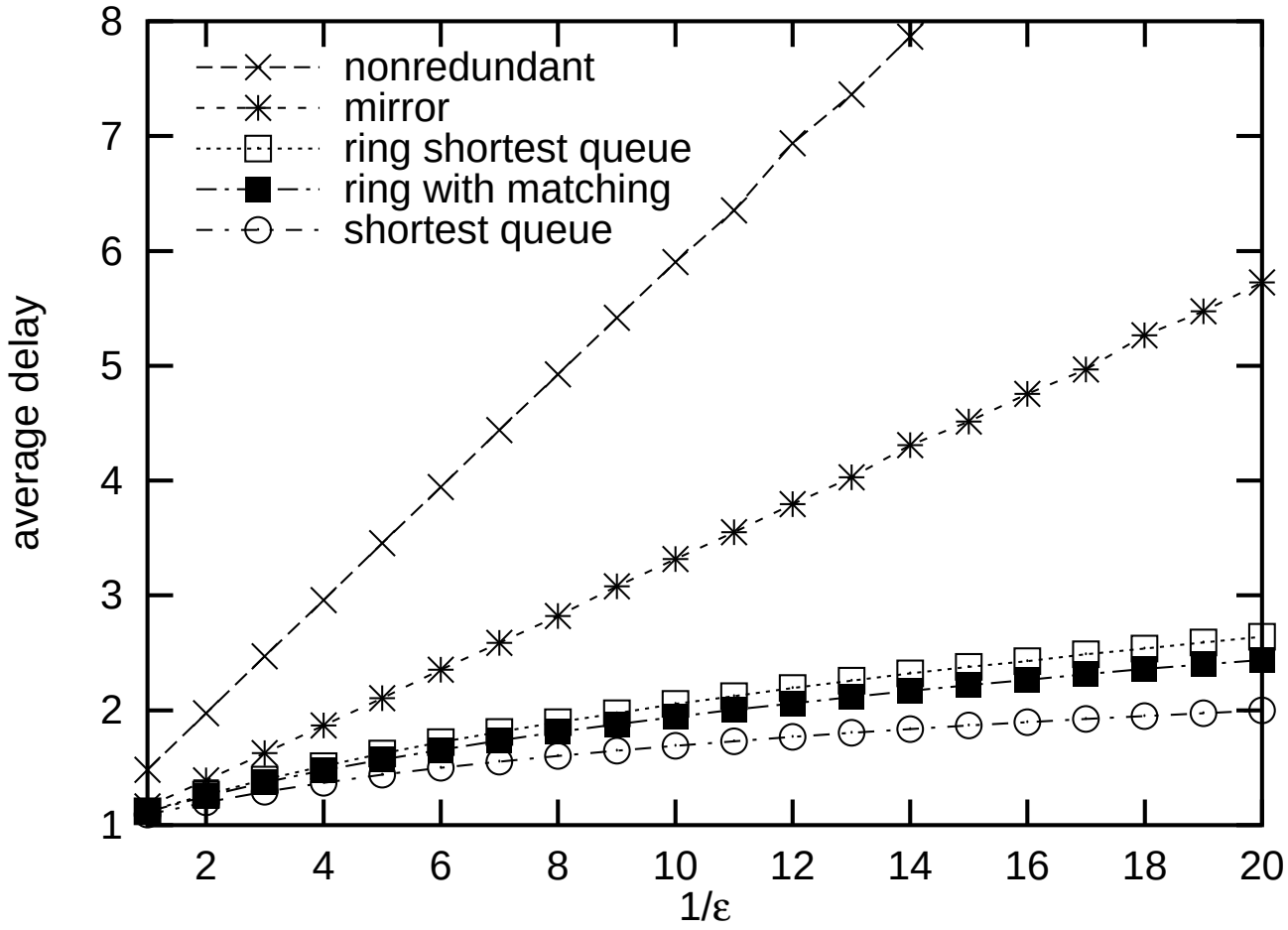
Keeping Curves apart: log y scale



Keeping Curves apart: smoothing



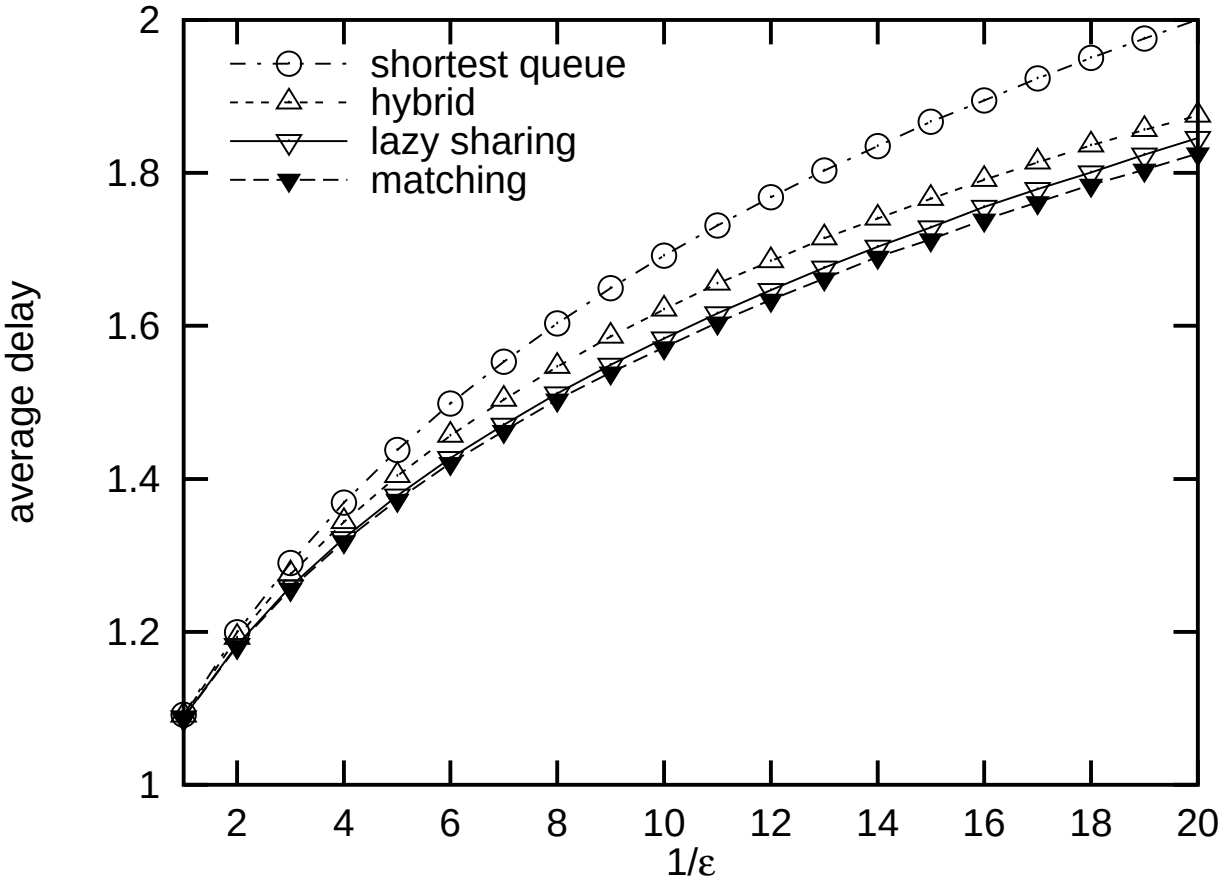
Keys



same order as curves

Keys

place in white space



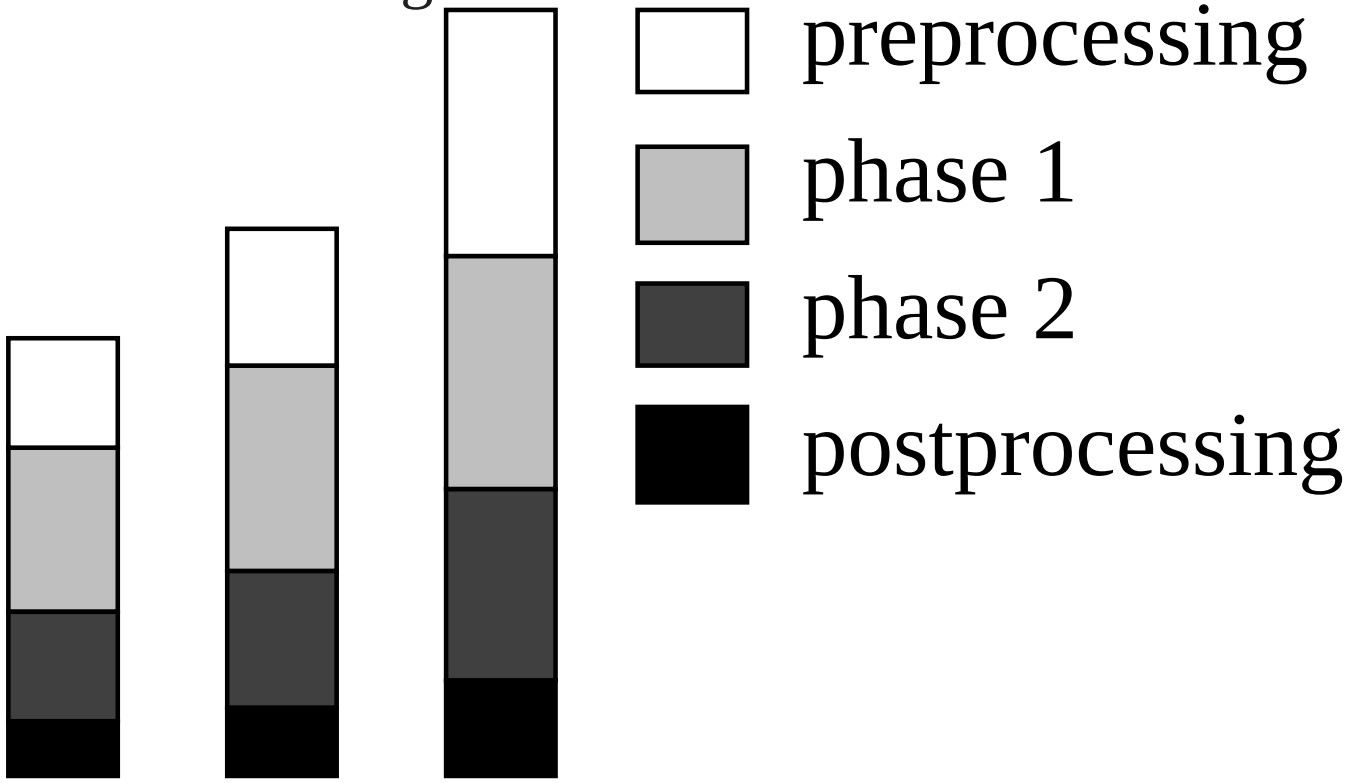
consistent in different figures

Todsünden

1. forget explaining the axes
2. connecting unrelated points by lines
3. mindless use/overinterpretation of double-log plot
4. cryptic abbreviations
5. microscopic lettering
6. excessive complexity
7. pie charts

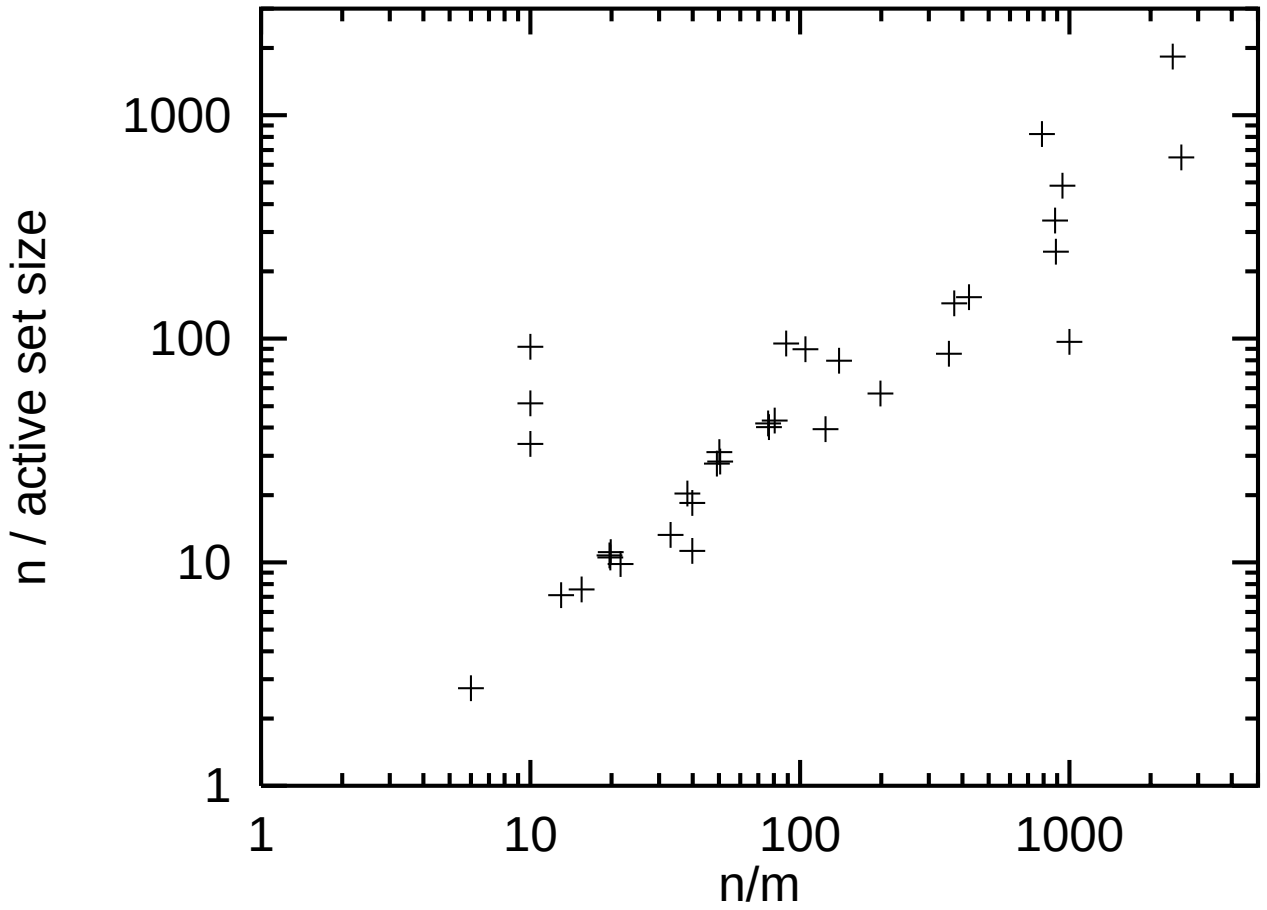
Arranging Instances

- bar charts
- stack components of execution time
- careful with shading



Arranging Instances

scatter plots



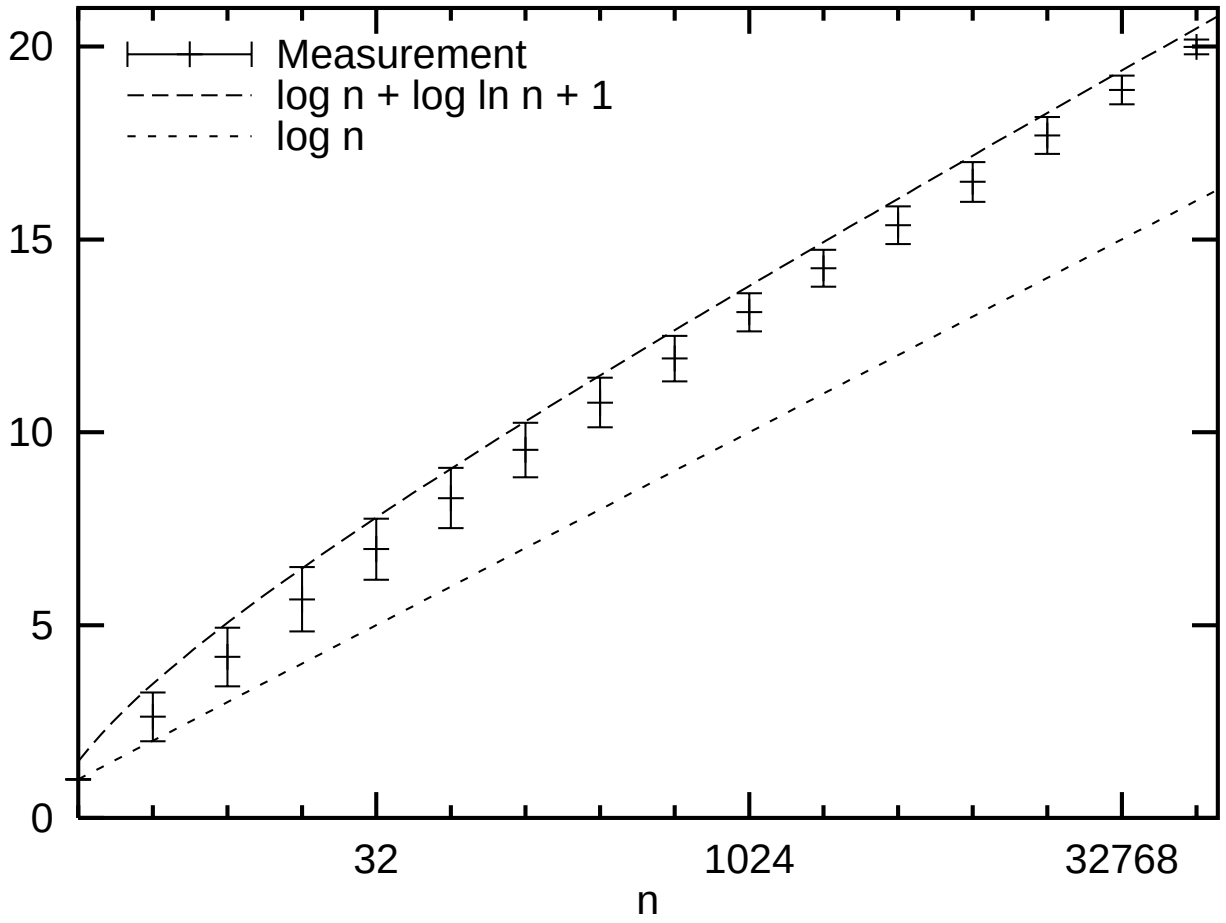
Measurements and Connections

- straight line between points do not imply claim of linear interpolation
- different with higher order curves
- no points imply an even stronger claim. Good for very dense smooth measurements.

Grids and Ticks

- Avoid grids or make it light gray
- usually round numbers for tic marks!
- sometimes plot important values on the axis

usually avoidable for randomized algorithms. median $\neq$ average,. . .



errors may not be of statistical nature!

3D

- you cannot read off absolute values
- interesting parts may be hidden
- only one surface
+ good impression of shape

Caption

what is displayed

how has the date been obtained

surrounding text has more.

Check List

- Should the experimental setup from the exploratory phase be redesigned to increase conciseness or accuracy?
- What parameters should be varied? What variables should be measured? How are parameters chosen that cannot be varied?
- Can tables be converted into curves, bar charts, scatter plots or any other useful graphics?
- Should tables be added in an appendix or on a web page?
- Should a 3D-plot be replaced by collections of 2D-curves?
- Can we reduce the number of curves to be displayed?
- How many figures are needed?

- Scale the x-axis to make y-values independent of some parameters?
- Should the x-axis have a logarithmic scale? If so, do the x-values used for measuring have the same basis as the tick marks?
- Should the x-axis be transformed to magnify interesting subranges?
- Is the range of x-values adequate?
- Do we have measurements for the right x-values, i.e., nowhere too dense or too sparse?
- Should the y-axis be transformed to make the interesting part of the data more visible?
- Should the y-axis have a logarithmic scale?

- Is it be misleading to start the y-range at the smallest measured value?
- Clip the range of y-values to exclude useless parts of curves?
- Can we use banking to 45°?
- Are all curves sufficiently well separated?
- Can noise be reduced using more accurate measurements?
- Are error bars needed? If so, what should they indicate? Remember that measurement errors are usually *not* random variables.
- Use points to indicate for which x-values actual data is available.
- Connect points belonging to the same curve.

- Only use splines for connecting points if interpolation is sensible.
- Do not connect points belonging to unrelated problem instances.
- Use different point and line styles for different curves.
- Use the same styles for corresponding curves in different graphs.
- Place labels defining point and line styles in the right order and without concealing the curves.
- Captions should make figures self contained.
- Give enough information to make experiments reproducible.