

The Weak-Heap Family of Priority Queues in Theory and Praxis

Stefan Edelkamp¹⁾ and Amr Elmasry²⁾

Jyrki Katajainen²⁾

1) University of Bremen

2) University of Copenhagen

These slides are available at <http://www.cphst1.dk>

Our research program

Provide the best bounds on the comparison complexity of priority-queue operations (n the current size of the data structure):

find-min: no element comparisons

delete, delete-min: $\sim \beta \lg n$ element comparisons

Other operations: $\sim \kappa$ element comparisons

where β and κ are constants.

worst case per operation

Full operation repertoire: *insert, borrow, decrease, meld*

New game

Application = Request sequence
(Think graph applications)

What is the best bound when handling a request sequence consisting of n *insert*, n *delete-min*, and m *decrease* operations?

worst case per sequence

Our result

Rank-relaxed weak heaps are **better** than Fibonacci heaps!

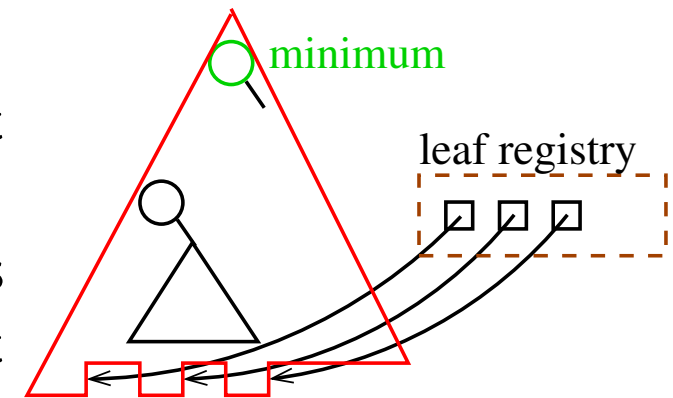
Data structure	# element comparisons
Fibonacci heap	$2m + 2.89n \lg n$
Rank-relaxed weak heap	$2m + 1.5n \lg n$

But they are not **simpler**!

Data structure	Lines of code
Binary heap	205
Fibonacci heap	296
Rank-relaxed weak heap	883

Weak heap

- A binary tree
- The root only has a right child if any
- Elements obey the weak-heap ordering: for element x , every element in the right subtree is $\geq x$
- With the exception of the root, the nodes that have at most one child are at the last two levels only



Our representation pointer-based!

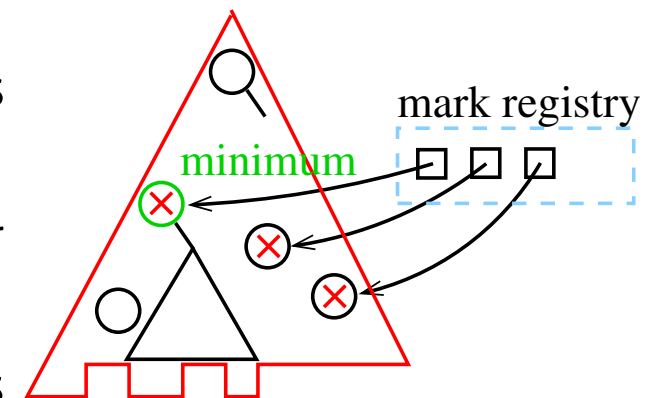
Rank-relaxed weak heap

- $\lambda \leq \lfloor \lg n \rfloor - 1$ nodes marked; they may violate the weak-heap ordering

insert: Insert a leaf, mark it, apply λ -reducing transformations as long as possible.

decrease: Decrease the value in the given node, mark it, apply λ -reducing transformations as long as possible.

delete-min: Find the minimum (at the root or one of the marked nodes), borrow a leaf, fix the structure of the subtree that lost its root, mark the root of the fixed subtree, apply λ -reducing transformations as long as possible.



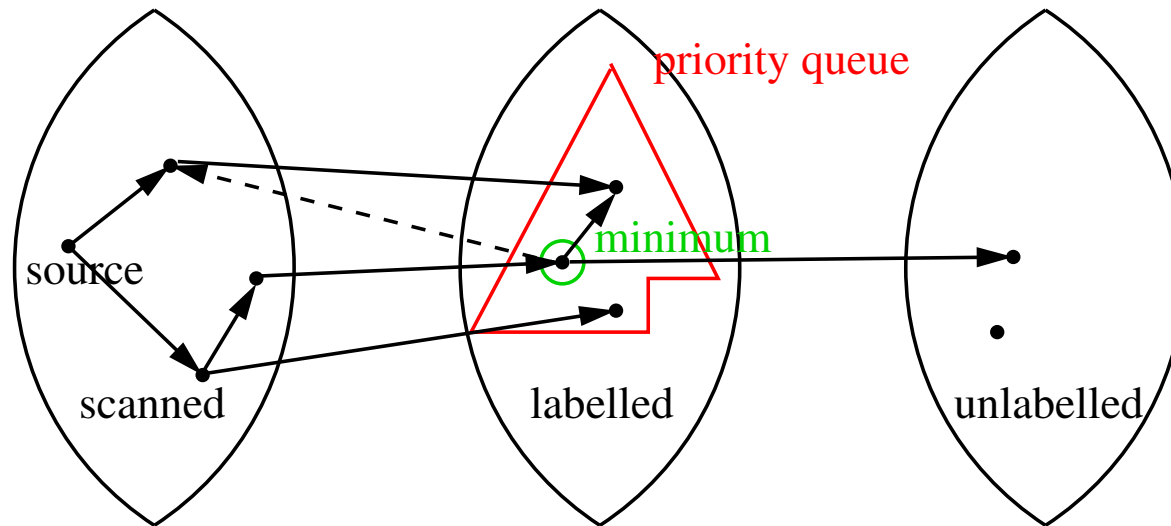
Improvement in *delete-min*: If the mark registry is more than half full before the minimum finding, empty it.

Normal referee comments

- Element comparisons are not relevant in practice!
- Pairing heaps are much faster in practice!
- No one would ever implement Fibonacci heaps since they are so complicated.

This is you guys! Thank you! You pushed us to do some practical experiments!

Our play with Dijkstra's algorithm



a factor of two speed-up

With your search engine, you will find many experimental studies on Dijkstra's algorithm. Be critical when you read the results.

- Which algorithm
- Which graph representation
- Which priority queue
- Which tuning level

Graph representation

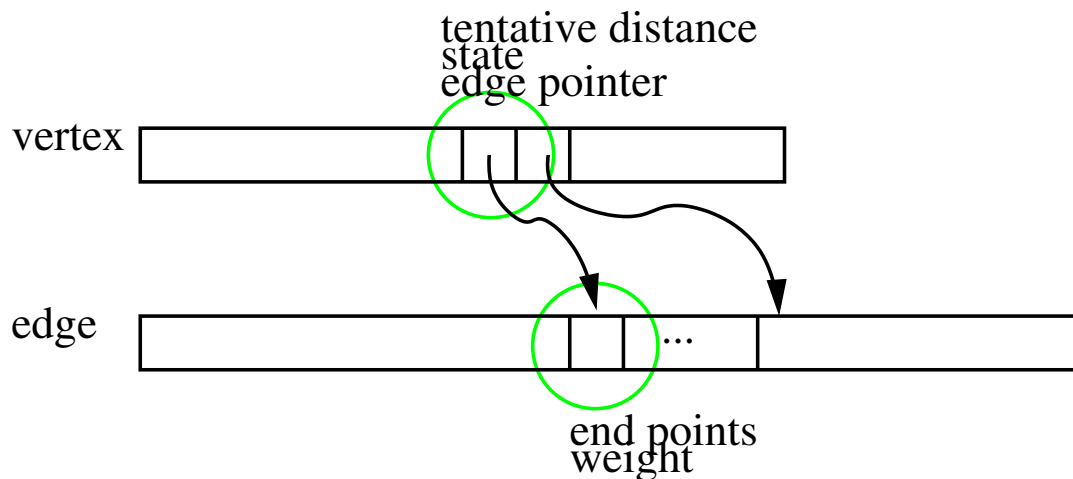
CPH STL

- adjacency arrays
- simple
- static
- $16m + 16n + O(1)$ bytes for a graph with m edges and n vertices

LEDA

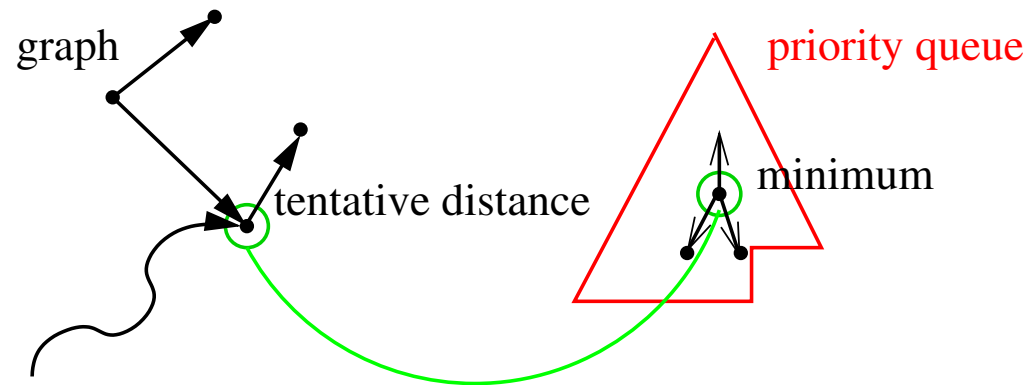
- adjacency lists
- nice interface
- fully dynamic
- parameterized
- $52m + 60n + O(1)$ bytes [LEDA Book, § 6.14]

weights doubles



a factor of two speed-up

Interaction



Combine the graph vertex and the priority-queue node [Knuth 1994]

→ improves cache behaviour

a factor of two speed-up

Tuning

Running time per n [μ s]

Structure	CPH STL	LEDA 6.2
Operation	Fibonacci heap	Fibonacci heap
<i>insert</i>		
$n: 10\ 000$	0.10	0.18
$n: 100\ 000$	0.09	0.15
$n: 1\ 000\ 000$	0.09	0.15
<i>decrease</i>		
$n: 10\ 000$	0.03	0.06
$n: 100\ 000$	0.05	0.22
$n: 1\ 000\ 000$	0.06	0.31
<i>delete-min</i>		
$n: 10\ 000$	0.7	1.2
$n: 100\ 000$	1.4	2.7
$n: 1\ 000\ 000$	2.8	4.5

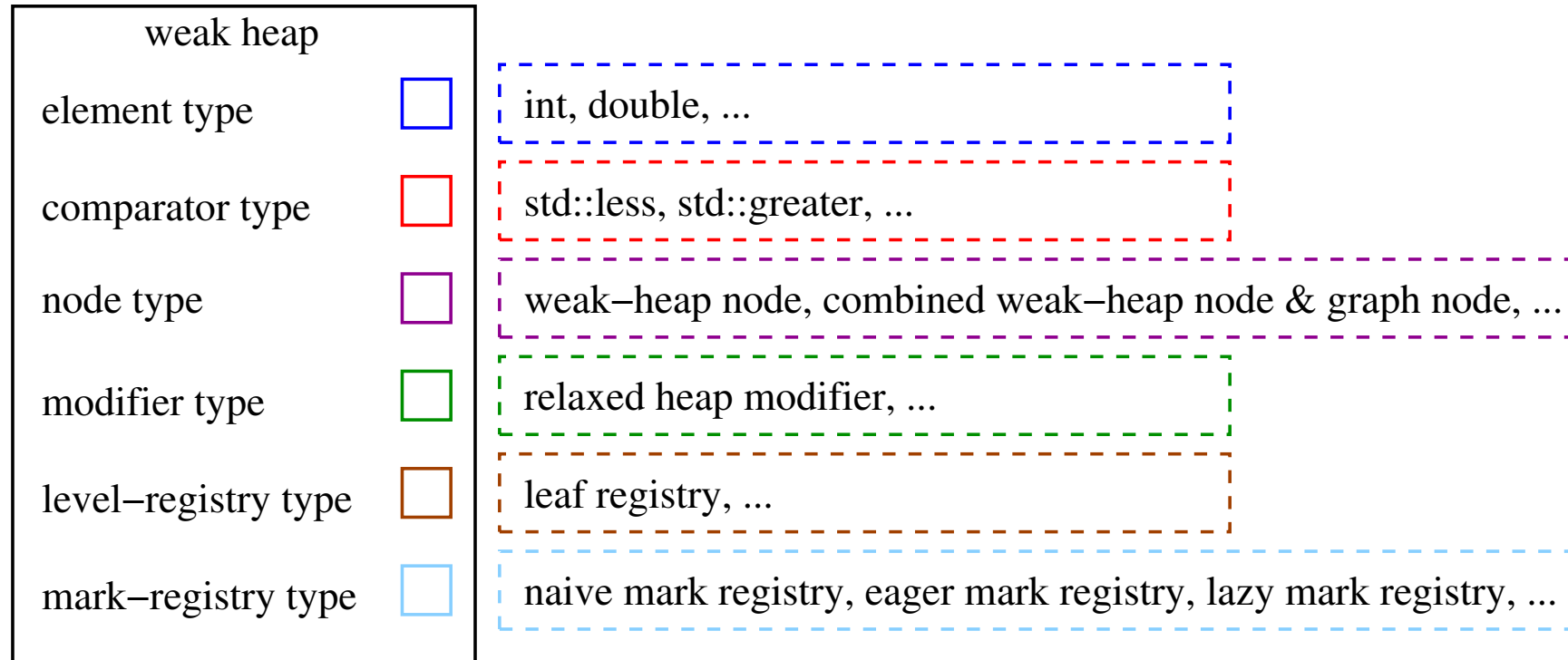
Element comparisons per n

Structure	CPH STL	LEDA 6.2
Operation	Fibonacci heap	Fibonacci heap
<i>insert</i>		
$n: 10\ 000$	0	1
$n: 100\ 000$	0	1
$n: 1\ 000\ 000$	0	1
<i>decrease</i>		
$n: 10\ 000$	0	2
$n: 100\ 000$	0	2
$n: 1\ 000\ 000$	0	2
<i>delete-min</i>		
$n: 10\ 000$	16.2	29.9
$n: 100\ 000$	21.2	38.3
$n: 1\ 000\ 000$	26.2	46.5

a factor of two speed-up

On my computer (Ubuntu, gcc, with -O3)

Parameterized design



- comparators shared
- nodes shared
- transformations shared
- level registries shared
- mark registries shared

a factor of two less code

What is the best?

Our reference sequence

Theory: rank-relaxed weak heap

Dijkstra—time: binary heap

[Williams 1964]

Dijkstra—comps: weak heap

[Dutton 1993]

Worst case per operation

insert—**time:** Fibonacci heap

[Fredman & Tarjan 1987]

insert—**comps:** Fibonacci heap

decrease—**time:** Fibonacci heap

decrease—**comps:** Fibonacci heap

delete-min—**time:** weak queue

[Vuillemin 1978]

delete-min—**comps:** weak heap

Conclusions

Earlier conjectures—see my SWAT-2010 slides

Katajainen's third conjecture: Our reference sequence can be handled with $2m + n \lg n + o(n)$ element comparisons in $O(m + n \lg n)$ time.

Cache effects: We have neglected the cache behaviour of priority queues for too long.

Source code: Our programs are available via the CPH STL website.