Graph Algorithms
Spanning Trees and Ranking

Martin Mareš
mares@kam.mff.cuni.cz

Department of Applied Mathematics
MFF UK Praha

2008

1. Minimum Spanning Tree Problem:

- Given a weighted undirected graph,
  what is its lightest spanning tree?
- In fact, a linear order on edges is sufficient.
- Efficient solutions are very old [Borůvka 1926]
- A long progression of faster and faster algorithms.
- Currently very close to linear time, but still not there.

2. Ranking of Combinatorial Structures:

- We are given a set C of objects with a linear order $\prec$.
- **Ranking function** $R_\prec(x)$**:** how many objects precede x?
- **Unranking function** $R_\prec^{-1}(i)$**:** what is the i-th object?

The Ranking Problems

2. Ranking of Combinatorial Structures:

- We are given a set C of objects with a linear order $\prec$.
- **Ranking function** $R_{\prec}(x)$**:** how many objects precede x?
- **Unranking function** $R_{\prec}^{-1}(i)$**:** what is the i-th object?

Example (toy)

$C = \{0, 1\}^n$ with lexicographic order

The Ranking Problems

2. Ranking of Combinatorial Structures:

- We are given a set C of objects with a linear order $\prec$.
- **Ranking function** $R_\prec(x)$**:** how many objects precede x?
- **Unranking function** $R_\prec^{-1}(i)$**:** what is the i-th object?

Example (toy)

$C = \{0, 1\}^n$ with lexicographic order
R = conversion from binary
R^{-1} = conversion to binary

The Ranking Problems

2. Ranking of Combinatorial Structures:

- We are given a set C of objects with a linear order $\prec$.
- **Ranking function** $R_{\prec}(x)$**:** how many objects precede x?
- **Unranking function** $R_{\prec}^{-1}(i)$**:** what is the i-th object?

Example (toy)

$C = \{0,1\}^n$ with lexicographic order
R = conversion from binary
R^{-1} = conversion to binary

Example (a real one)

$C =$ set of all permutations on $\{1, \ldots, n\}$

2. Ranking of Combinatorial Structures:

- We are given a set C of objects with a linear order $\prec$.
- **Ranking function** $R_\prec(x)$**:** how many objects precede x?
- **Unranking function** $R_\prec^{-1}(i)$**:** what is the i-th object?

Example (toy)

$C = \{0,1\}^n$ with lexicographic order
R = conversion from binary
R^{-1} = conversion to binary

Example (a real one)

$C =$ set of all permutations on $\{1,\ldots,n\}$

How to compute the (un)ranking function efficiently?
For permutations, an $\mathcal{O}(n \log n)$ algorithm was known [folklore].
We will show how to do that in $\mathcal{O}(n)$.

Models of computation: RAM

As we approach linear time, we must specify the model.

1. The Random Access Machine (RAM):

- Works with integers
- Memory: an array of integers indexed by integers

Models of computation: RAM

As we approach linear time, we must specify the model.

1. The Random Access Machine (RAM):

- Works with integers
- Memory: an array of integers indexed by integers

Many variants exist, we will use the **Word-RAM:**

- Machine words of W bits
- The “C operations”: arithmetics, bitwise logical op’s
- Unit cost
- We know that $W \geq \log_2 |\text{input}|$

Models of computation: PM

2. The Pointer Machine (PM):

- Memory cells accessed via pointers
- Each cell contains $\mathcal{O}(1)$ pointers and $\mathcal{O}(1)$ symbols
- Operates only on pointers and symbols

Models of computation: PM

2. The Pointer Machine (PM):

- Memory cells accessed via pointers
- Each cell contains $\mathcal{O}(1)$ pointers and $\mathcal{O}(1)$ symbols
- Operates only on pointers and symbols

Key differences

- PM has no arrays, we can emulate them in $\mathcal{O}(\log n)$ time.
- PM has no arithmetics.

We can emulate PM on RAM with constant slowdown.
Emulation of RAM on PM is more expensive.

PM Techniques

Bucket Sorting does not need arrays.

Interesting consequences:

- Flattening of multigraphs in $\mathcal{O}(m+n)$
- Unification of sequences in $\mathcal{O}(n + \sum_i \ell_i + |\Sigma|)$
- (Sub)tree isomorphism in $\mathcal{O}(n)$ simplified [M. 2008]
- Batched graph computations [Buchsbaum et al. 1998]

We can use RAM as a vector machine:

Example (parallel search)

We can encode the vector $(1, 5, 3, 0)$ with 3-bit fields as:

$$0001\ 0101\ 0011\ 0000$$

And then search for 3 by:

	1001 1101 1011 1000	$(1, 5, 3, 0)$
XOR	0011 0011 0011 0011	$(3, 3, 3, 3)$
	1010 1110 1000 1011	
$-$	0001 0001 0001 0001	$(1, 1, 1, 1)$
	1001 1101 0111 1010	
AND	1000 1000 1000 1000	
	1000 1000 0000 1000	

RAM Data Structures

We can translate vector operations to $\mathcal{O}(1)$ RAM instructions

... as long as the vector fits in $\mathcal{O}(1)$ words.

We can build "small" data structures operating in $\mathcal{O}(1)$ time:

- Sets
- Ordered sets with ranking
- "Small" heaps of "large" integers [Fredman & Willard 1990]

Algorithms for Minimum Spanning Trees:

- Classical algorithms [Borůvka, Jarník-Prim, Kruskal]
- Contractive: $\mathcal{O}(m \log n)$ using flattening on the PM (lower bound [M.])
- Iterated: $\mathcal{O}(m\,\beta(m,n))$ [Fredman & Tarjan 1987] where $\beta(m,n) = \min\{k : \log_2^{(k)} n \le m/n\}$
- Even better: $\mathcal{O}(m\,\alpha(m,n))$ using *soft heaps* [Chazelle 1998, Pettie 1999]
- MST verification: $\mathcal{O}(m)$ on RAM [King 1997, M. 2008]
- Randomized: $\mathcal{O}(m)$ expected on RAM [Karger et al. 1995]

MST – Special cases

Cases for which we have an $\mathcal{O}(m)$ algorithm:

Special graph structure:

- Planar graphs [Tarjan 1976, Matsui 1995, M. 2004] (PM)
- Minor-closed classes [Tarjan 1983, M. 2004] (PM)
- Dense graphs (by many of the general PM algorithms)

MST – Special cases

Cases for which we have an $\mathcal{O}(m)$ algorithm:

Special graph structure:

- Planar graphs [Tarjan 1976, Matsui 1995, M. 2004] (PM)
- Minor-closed classes [Tarjan 1983, M. 2004] (PM)
- Dense graphs (by many of the general PM algorithms)

Or we can assume more about weights:

- $\mathcal{O}(1)$ different weights [folklore] (PM)
- Integer weights [Fredman & Willard 1990] (RAM)
- Sorted weights (RAM)

MST – Optimality

There is a provably optimal comparison-based algorithm [Pettie & Ramachandran 2002]

However, there is a catch . . .

MST – Optimality

There is a provably optimal comparison-based algorithm [Pettie & Ramachandran 2002]

However, there is a catch: Nobody knows its complexity.

We know that it is $\mathcal{O}(\mathcal{T}(m,n))$ where $\mathcal{T}(m,n)$ is the depth of the optimum MST decision tree. Any other algorithm provides an upper bound.

MST – Optimality

There is a provably optimal comparison-based algorithm [Pettie & Ramachandran 2002]

However, there is a catch: Nobody knows its complexity.

We know that it is $\mathcal{O}(\mathcal{T}(m,n))$ where $\mathcal{T}(m,n)$ is the depth of the optimum MST decision tree. Any other algorithm provides an upper bound.

Corollary

It runs on the PM, so we know that if there is a linear-time algorithm, it does not need any special RAM data structures. (They can however help us to find it.)

MST – Dynamic algorithms

Sometimes, we need to find the MST of a changing graph. We insert/delete edges, the structure responds with $\mathcal{O}(1)$ modifications of the MST.

- Unweighted cases, similar to dynamic connectivity:
  - Incremental: $\mathcal{O}(\alpha(n))$ [Tarjan 1975]
  - Fully dynamic: $\mathcal{O}(\log^2 n)$ [Holm et al. 2001]

MST – Dynamic algorithms

Sometimes, we need to find the MST of a changing graph. We insert/delete edges, the structure responds with $\mathcal{O}(1)$ modifications of the MST.

- Unweighted cases, similar to dynamic connectivity:
  - Incremental: $\mathcal{O}(\alpha(n))$ [Tarjan 1975]
  - Fully dynamic: $\mathcal{O}(\log^2 n)$ [Holm et al. 2001]
- Weighted cases are harder:
  - Decremental: $\mathcal{O}(\log^2 n)$ [Holm et al. 2001]
  - Fully dynamic: $\mathcal{O}(\log^4 n)$ [Holm et al. 2001]
  - Only C weights: $\mathcal{O}(C \log^2 n)$ [M. 2008]

MST – Dynamic algorithms

Sometimes, we need to find the MST of a changing graph. We insert/delete edges, the structure responds with $\mathcal{O}(1)$ modifications of the MST.

- Unweighted cases, similar to dynamic connectivity:
  - Incremental: $\mathcal{O}(\alpha(n))$ [Tarjan 1975]
  - Fully dynamic: $\mathcal{O}(\log^2 n)$ [Holm et al. 2001]
- Weighted cases are harder:
  - Decremental: $\mathcal{O}(\log^2 n)$ [Holm et al. 2001]
  - Fully dynamic: $\mathcal{O}(\log^4 n)$ [Holm et al. 2001]
  - Only C weights: $\mathcal{O}(C \log^2 n)$ [M. 2008]
- K smallest spanning trees:
  - Simple: $\mathcal{O}(T_{MST} + Km)$ [Katoh et al. 1981, M. 2008]
  - Small K: $\mathcal{O}(T_{MST} + \min(K^2, Km + K \log K))$ [Eppst. 1992]
  - Faster: $\mathcal{O}(T_{MST} + \min(K^{3/2}, Km^{1/2}))$ [Frederickson 1997]

Back to Ranking

Ranking of permutations on the RAM: [M. & Straka 2007]

- We need a DS for the subsets of $\{1, \ldots, n\}$ with ranking
- The result can be $n!$ $\Rightarrow$ word size is $\Omega(n \log n)$ bits
- We can represent the subsets as RAM vectors
- This gives us an $\mathcal{O}(n)$ time algorithm for (un)ranking

Easily extendable to k-permutations, also in $\mathcal{O}(n)$

Restricted permutations

For restricted permutations (e.g., derangements): [M. 2008]

- Describe restrictions by a bipartite graph
- Existence of permutation reduces to network flows
- The ranking function can be used to calculate permanents, so it is $\#\mathrm{P}$-complete
- However, this is the only obstacle. Calculating $\mathcal{O}(n)$ sub-permanents is sufficient.
- For derangements, we have achieved $\mathcal{O}(n)$ time after $\mathcal{O}(n^2)$ time preprocessing.

Summary

Summary:

- Low-level algorithmic techniques on RAM and PM
- Generalized pointer-based sorting and RAM vectors
- Applied to a variety of problems:
    - A short linear-time tree isomorphism algorithm
    - A linear-time algorithm for MST on minor-closed classes
    - Corrected and simplified MST verification
    - Dynamic MST with small weights
    - *Ranking and unranking of permutations*
- Also:
    - A lower bound for the Contractive Borůvka's algorithm
    - Simplified soft-heaps

THE END