2D Range Maximum Queries

Rajeev Raman

University of Leicester

December 30, 2011

RMQ problems

We will consider two problems:

- Matrix RMQ.
  - Joint work with Golin, Iacono, Krizanc and Satti.
- Geometric RMQ.
  - Joint work with Farzan and Munro.

Basically two talks in one.

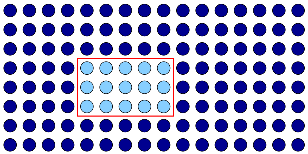
Problem Definition

Matrix RMQ

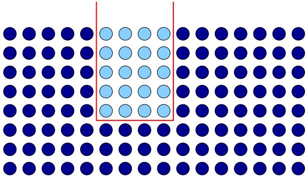
Given static $m \times n$ matrix A ($m \leq n$) pre-process A to answer 2D-RMQ queries:

- return $\mathrm{argmax}_{i_1 \leq i \leq i_2, j_1 \leq j \leq j_2} A[i,j]$ for any indices $1 \leq i_1 \leq i_2 \leq m$, $1 \leq j_1 \leq j_2 \leq n$.

Introduced by Amir et al. (2007), generalises classic 1D RMQ problem.
Also: 3-sided, 2-sided, 1-sided queries.



4-sided query



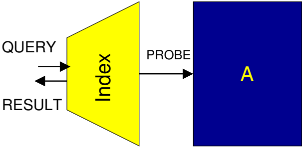
3-sided query

Matrix RMQ: Encoding vs. Indexing

Brodal et al. (2010) studied this in two models:

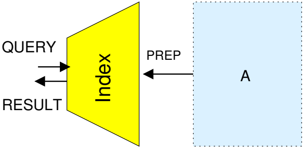
Indexing model

- Preprocess A to get *index*.
- Queries can access index, and *probe* A.
- Parameters: (a) size of index (b) number of probes (c) running time of query.



Encoding model

- Preprocess A to get *index*, delete A.
- Queries can access *only* index, and not A.
- Parameters: (a) size of index (b) running time of query.



We focus on *encoding* model. Amir et al. only looked at indexing model.

Effective Entropy

A key concept is *effective entropy*.

- Extensive literature on *succinct and compressed data structures*.
- Space usage is related to "information content of data"
  - To represent an object from a set S requires at least $\lceil \lg |S| \rceil$ bits[1]
- We consider the "information content of the data *structure*:"
  - Given a set of objects S,
  - a set of queries Q,
  - let $\mathcal{C}$ be the equivalence class on S induced by Q ($x, y \in S$ are equivalent if they cannot be distinguished by queries in Q).
  - We want to store x in $\lceil \lg |\mathcal{C}| \rceil$ bits.
- Want index size to equal the effective entropy (exact constant if possible).
- Can define *expected* effective entropy as well.

[1] $\lg = \log_2$.

Effective Entropy

Term "effective entropy" is new, but the concept is not. For $m = 1$:

- Input is $A[1..n]$.
  - Information content of $A = \lg n! \leq n \lg n$ bits.
- Use "Cartesian tree" of A [Vuillemin, '80].
  - Cartesian tree can be represented in $2n - O(\lg n)$ bits.
  - Cartesian tree completely characterizes 1D case $\rightarrow$ effective entropy of 1D RMQ is $2n - O(\lg n)$ bits.
- The low effective entropy of 1D RMQ is used in many space-efficient data structures.

For other values of m (recall $m \leq n$):

$m = n$	$\Omega(n^2 \lg n)$	[Demaine et al., 09]
General	$O(nm \lg n)$	[Obvious]
	$\Omega(nm \lg m)$	[Brodal et al., 10]
	$O(nm^2)$	*ibid.*

Open question by Brodal et al.: close the gap between upper and lower bounds.

Our contributions

1. Asymptotically tight upper and lower bounds on expected effective entropy for 1-, 2-, 3- and 4-sided queries when A is a random permutation, as follows (values in bits):

1-sided	2-sided	3-sided	4-sided
$\Theta((\log n)^2)$	$\Theta((\log n)^2 \log m)$	$\Theta(n(\log m)^2)$	$\Theta(nm)$

   - Random model interesting "in practice" ("adaptive" DS?) also may be used for approximate range queries (cf. [Kaplan et al. 2010]).
   - For $m = 1$ we show expected upper bound of $\leq 1.92n$, less than known $2n - O(\lg n)$ for worst case.

2. Bounds on effective entropy for small m for arbitrary A:

$m = 2$	$5n - O(\lg n)$	$\leq 7n - O(\lg n)$ [Brodal et al. 2010]
$m = 3$	$\leq 8.32n$	$\leq 14.32n - O(\lg n)$ [*ibid.*]

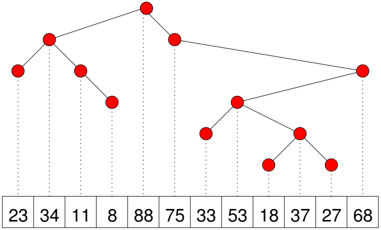
3. Data structures (RAM model):
   - For $m = 2$, and any A, can answer 2D RMQ queries in $(5 + \epsilon)n$ bits and $O(1/\epsilon)$ query time, for any $\epsilon > 0$.
     - **NB:** Cartesian tree (1980) $\rightarrow 2n + o(n)$-bit, $O(1)$-time DS [Fischer and Heun, 2010].
   - For random A, space $O(N)$ bits with high probability and $O(1)$ worst-case query time.

Encoding 1-D RMQ

Cartesian tree [Vuillemin, '80]

CT of $A[1..n]$ is a binary tree:

- If $j = \text{argmax}_i A[i]$, place j at root.
- Left (right) child: recurse on $A[1..j]$, $A[j+1..n]$.

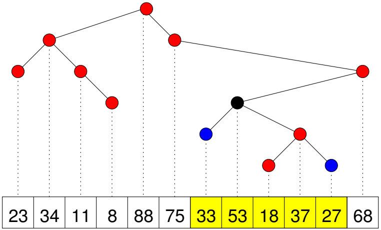


Encoding 1-D RMQ

Cartesian tree [Vuillemin, '80]

CT of $A[1..n]$ is a binary tree:

- If $j = \mathrm{argmax}_i A[i]$, place j at root.
- Left (right) child: recurse on $A[1..j]$, $A[j+1..n]$.



- RMQ(i,j) is the LCA of i and j.
- Binary tree can be encoded using $2n - O(\log n)$ bits.
    - Effective entropy of 1D RMQ $= 2n - O(\log n)$ bits.

Effective Entropy for 1-D RMQ: Random Input

- Every binary tree is the CT of some array A.
- Binary trees need exactly $2n - O(\log n)$ bits: done?

Choosing A uniformly does not induce a uniform distribution on CT(A).

- Arity of each node in CT can be 0, L, R or LR.
- Can reconstruct CT from arities written in DFS.
- Arity is *locally* determined. Arity of $A[i]$ is:

0	iff $\min\{A[i-1], A[i+1]\} > A[i]$	$\Pr = 1/3$
L	iff $A[i-1] < A[i] < A[i+1]$	$\Pr = 1/6$ (R symmetric)
LR	iff $\max\{A[i-1], A[i+1]\} < A[i]$	$\Pr = 1/3$.

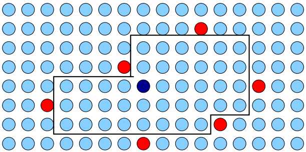
Entropy of distribution < 1.92 bits/symbol. We believe 1.74 bits/symbol is the right answer.

Effective Entropy of 2D RMQ for Random Matrices

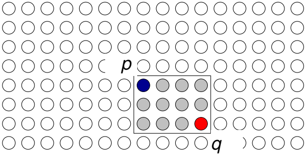
Key idea used: *areas of influence (AOI)*.

- For each (i, j), its AOI is the region such that any RMQ query contained entirely in the AOI has (i, j) as the answer.
- AOI delimited by *boundary points*.
- Encoding: for each (i, j), write coordinates of boundary points of its AOI.
- $q = (i + x, j + y)$ is a boundary point of $p = (i, j)$ iff $A[q] > A[p]$ and all other points in rectangle defined by p and q are smaller.
- $\Pr \approx 1/(x^2 y^2)$, cost $= O(\lg x + \lg y)$, get $O(nm)$ expected effective entropy by linearity of expectation.

This is tight: can encode $(nm)/2$ random bits by comparing disjoint pairs.



Boundary points in red.



Small m, arbitrary A

Objective

To identify expected entropy (to within constant factors) of 2D RMQ for fixed values of m and arbitrary (non-random) A.

$m = 2$, Brodal et al.'s approach

Call the two rows T and B.

1. Store CTs for T, B and TB (columnwise maxima). Total $\sim 6n$ bits.
2. Answer queries on T or B alone using respective CTs.
3. To answer queries on two rows, use TB — this only gives the column number, so store n bits giving location of columnwise max bigger ($\sim 7n$ bits total).

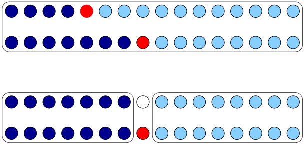
$m = 3$, Brodal et al.'s approach

Same calculation gives $6 \cdot (2n - O(\log n)) + n \log_2 5 \sim 14.32$ bits.

$m = 2$

Upper bound[2]

Based on *merging* CTs. Store T and B as before. For TB:



- Use T, B to get row maxima i and j.
- Compare $T[i]$ and $B[j]$ — store 1 bit. Say $B[j]$ is bigger.
- Recurse on $[1..j-1]$ and $[j+1..n]$.

Uses 2 CTs + n bits $\sim 5n - O(\lg n)$ bits.

Lower bound

- We show that for *any* given CTs for T and B, all possible n merging bits are "valid".
- Effective entropy for $m = 2$ is *exactly* n bits plus the effective entropy for T and B.
- Encoding for $m = 2$ is *exactly* optimal.

[2] Also discovered by Brodal.

$m = 3$

1. Store CTs for T, M, B ($\sim 6n$ bits).
2. Create CT for TMB by "merging" T, M, B.
   2.1 "merge string" is of ternary alphabet $\{t, m, b\}$.
   2.2 Encode using arithmetic coding, with $\Pr[t] = \Pr[b] = 2/5$, $\Pr[m] = 1/5$.
   2.3 Uses $(n - z)\log_2(5/2) + z\log_2 5$ bits, if $z = \#m$'s.
3. When creating TB and MB, do not store "merge bit" if comparison result can be found from TMB.
   3.1 Each max in TMB from M saves 2 bits, and from T/B saves 1 bit.
   3.2 Merge strings for T and B use $(2n - (n - z) - 2z) = n - z$ bits.
4. Total (2.3) + (3.2): $n\log_2 5$ bits, or $\sim (6 + \log_2 5)n$ bits.

- **OPEN:** Tight? (Believe: yes)
- **OPEN:** Pro(v/b)ably tight bound for $m = 4$, $m = 5$?

Data Structures

We give RAM data structures for some of these encodings.

Random Input

- Encoding based on AOIs gives a nice characterization of RMQ answer as LCA in an "AOI DAG".
  - DAG LCA can't be done sufficiently fast.
- Our data structure uses the following idea (roughly):
  - Store relative order explicitly for top $nm/(\lg n)$ values.
  - Areas of size $(\lg n)^{O(1)}$ must contain at least one "large" value whp.
  - Recurse on smaller areas (details...).

$m = 2$, arbitrary A

- Based on "CT merging" encoding.
  - Store the CTs for T and B via [Fischer and Heun, '10], using $4n + o(n)$ bits.
  - CT for TB is not explicitly stored. We partition it into micro-trees using the method of [Farzan and Munro, '08] and reconstruct micro-tree on the fly using the "merging" bits.
  - Space used is $(5 + \epsilon)n$ bits, running time $O(1/\epsilon)$, for any $\epsilon > 0$.
- **O**PEN: Optimal space, $O(1)$ time?

Conclusion

We considered the problem of encoding 2D RMQs. We didn't solve Brodal et al.'s open problem but:

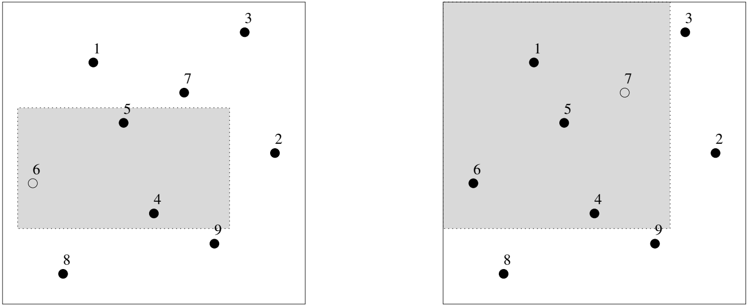
- Gave solutions for the random case;
- Took steps towards solving the problem by consdering the case of small fixed m.
- Gave some data structures matching the encoding sizes.

Many open questions and directions to follow up.

Geometric RMQ

Problem

Given n points in $\Re^2$, each point with a priority, return the point with the maximum priority in a query rectangle.



4-sided query 2-sided query

- Classic "decomposable" range search problem.

Simplifying Assumptions

Input set of N points (N is a power of 2).

- Wlog problem reduced to *rank space*.
- x coordinates are $\{1, \ldots, N\}$.
- y coordinates are permutation of $\{1, \ldots, N\}$.
- priorities are permutation of $\{1, \ldots, N\}$.

Model of computation: word RAM with word size $O(\log N)$ bits.

- Classic "decomposable" range search problem.

Existing Results

Citation	Size (in words)	Query time
Chazelle'88	$O\left(N \log^{\epsilon} N\right)$	$O\left(\log N\right)$
Chan et al.'10	$O\left(N \log^{\epsilon} N\right)$	$O\left(\log \log N\right)$
Karpinski et al.'09	$O\left(N(\log \log N)^{O(1)}\right)$	$O\left((\log \log N)^{2}\right)$
Chazelle'88	$O\left(N \log \log N\right)$	$O\left(\log N \log \log N\right)$
Chazelle'88	$O\left(\frac{1}{\epsilon} N\right)$	$O\left(\log^{1+\epsilon} N\right)$

Existing Results

Citation	Size (in words)	Query time
Chazelle'88	$O\left(N \log^{\epsilon} N\right)$	$O\left(\log N\right)$
Chan et al.'10	$O\left(N \log^{\epsilon} N\right)$	$O\left(\log\log N\right)$
Karpinski et al.'09	$O\left(N(\log\log N)^{O(1)}\right)$	$O\left((\log\log N)^{2}\right)$
Chazelle'88	$O\left(N \log\log N\right)$	$O\left(\log N \log\log N\right)$
Chazelle'88	$O\left(\frac{1}{\epsilon} N\right)$	$O\left(\log^{1+\epsilon} N\right)$
NEW	$O\left(N\right)$	$O\left(\log N \left(\log\log N\right)^{O(1)}\right)$

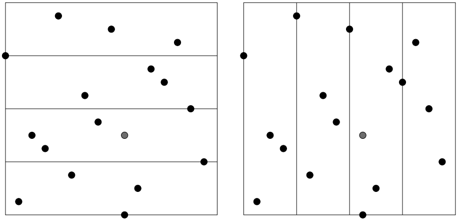
Key steps:

- Effective entropy + reduce 2S queries to point location.
- Decomposition into recursive subproblems.
- Implicit representation of subproblems.

Decomposition into recursive problems

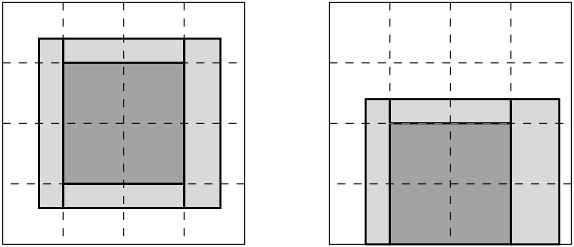
Current problem size: m (initially $m = N$).

- Divide given $m \times m$ grid into m/k horizontal slabs of size $k \times m$ and also into m/k vertical slabs of size $m \times k$ ($k = \Theta(\sqrt{m})$).



- Each slab is "essentially" a $k \times k$ problem.
- After r levels of recursion, there are $2^r N^{1-1/2^r}$ sub-problems.
- Stop when $2^r = \log N / \log\log N$:
  - Depth of recursion is $\Theta(\log\log N)$.
  - Final recursive problem size is $(\log N)^{O(1)}$.
  - Total number of terminal recursive problems $= O(N/\log\log N)$.
- Terminal problems take $O(m \log m)$ bits, and non-terminal problems take $O(m\sqrt{\log m})$ bits $\Rightarrow O(N)$ space.

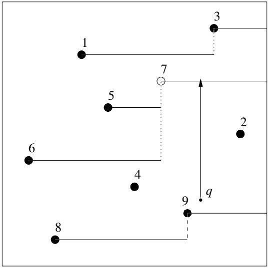
Reductions



A given 4-sided query is decomposed as above:

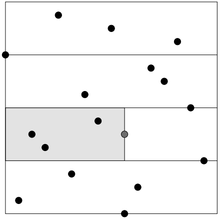
- "block-aligned" queries are easy [Atallah and Yuan '09, Brodal et al. '10].
- Once – decompose into four three-sided queries.
- Each three-sided query gives one recursive 3-sided query plus two two-sided queries.
- Each two-sided query is solved directly and gives rise to a *candidate* – $O(\log \log N)$ candidates. We scan all candidates and find the maximum.

2-sided Queries



- Each point is associated with a line *Inf*(p).
- 2-sided query at q: shoot ray upward until you hit *Inf*(q) for some q – this is the answer.
- Given point set, $O(m)$ bits suffice to encode all priority information (low effective entropy).
  - Plane sweep, note how many currently existing *Inf* lines are killed off by new point.

Slab-rank and Slab-select



- Slab-rank in $O(\log N / \log\log N)$ time by range counting.
- Slab-select can be done by range selection (we use simplified solution).

Point location structures

- *Very* limited space: $O(m\sqrt{\log m})$ bits!
- Create complete "rectangular" decomposition of *Inf* lines (too expensive).
- Select $O(m/\sqrt{\log m})$ lines that partition the decomposition into rectangular cells that contain $O(\sqrt{\log m})$ points and/or pieces of *Inf* lines.
- Build planar point location data structure on these lines.
- Locate query point q in this structure, but we can't afford to keep the points in the cell that we find – retrieve the points using range queries.
  - However, the cell can have *Inf* lines crossing it originating at points in other cells.

Conclusions

- Presented a linear-space data structure for geometric RMQ.
- Running time is $O(\log N(\log\log N)^{O(1)})$, would like $O(\log N)$ time.
- Key ingredient: ideas from *succinct indexes* and *effective entropy*.