Untangled Monotonic Chains and Adaptive Range Search

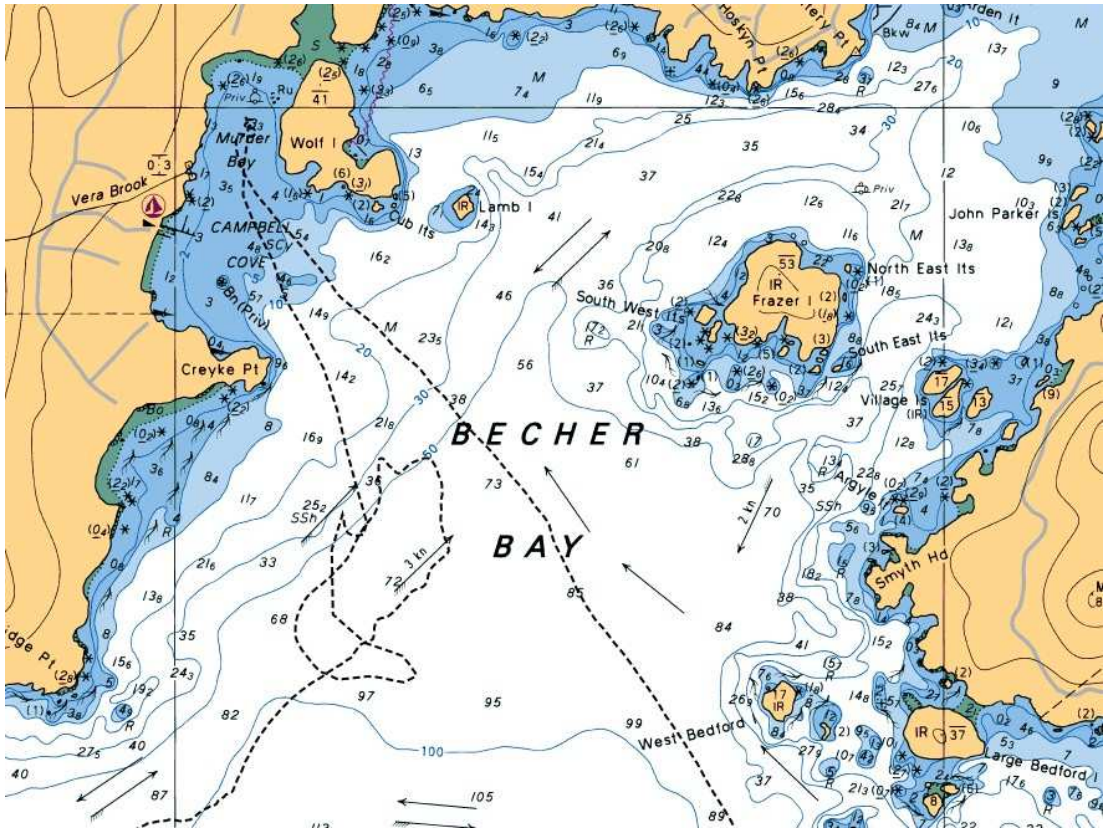
Diego Arroyuelo, Francisco Claude, Reza Dorrigiv, Stephane Durocher, Meng He, Alejandro López-Ortiz, J. Ian Munro, Patrick K. Nicholson, Alejandro Salinger, Matthew Skala*

December 16, 2009

Outline

- Orthogonal range search
- Previous work
- Adapting to good data
- Untangling
- Conclusions

Orthogonal range search



We want to preprocess the map and store it in some compact form so that we can quickly retrieve the points in any orthogonal rectangle.

Previous work

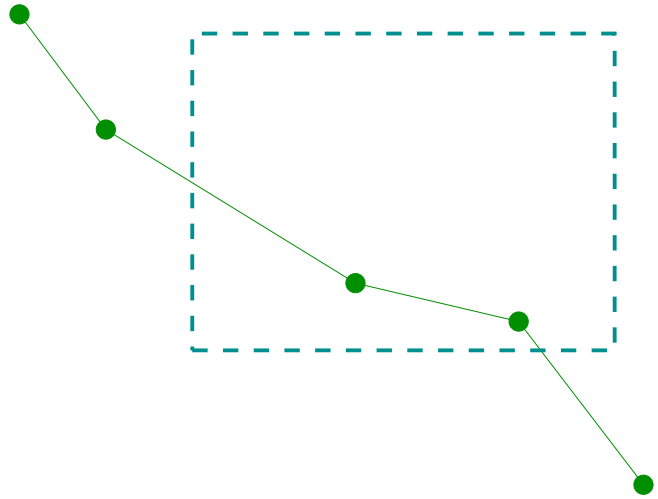
Data structure	Worst-case query	Space
kd-trees [Bentley, 1975]	$O(\sqrt{n}+m)$	implicit [Munro, 1979]
Range trees [Lueker, 1978]	$O(\log n+m)$	$O(n\log n)$
R-trees [Guttman, 1984]	$O(n)$	$O(n)$
PR-trees [Arge et al., 2008]	$O(\sqrt{n}+m)$	$O(n)$
[Nekrich, 2009]	$O(\log n+m\log^{\epsilon} n)$	$O(n)$
This paper	$O(\log n+k+m)$	$O(n)$
This paper	$O(k\log n+m)$	implicit

In this table n is the size of the database, m is the size of the query result, $k=O(\sqrt{n})$ is a hardness measure, and ϵ is an adjustable small constant.

Optimal structures require either $O(\sqrt{n}+m)$ query or superlinear space [Kanth and Singh, 1999]. Can we beat that with lucky data?

Adapting to good data

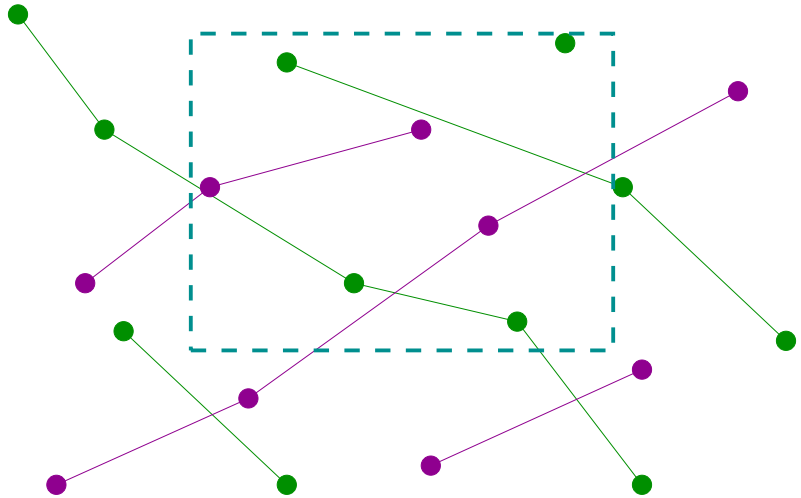
Q: On what kind of data would it be easy to do orthogonal range queries?



A: A diagonal chain, where the ordering along one dimension was the same as along the other. Then the query result would always be a contiguous subrange and we could finds its ends by binary search, with $O(\log n + m)$ time to return the result.

A first attempt (the pitch)

Split the data into as few of those chains as possible and search each one separately.



Gives $O(k \log n + m)$ query time; k is worst-case $O(\sqrt{n})$; and space not just linear but implicit. Sounds good, right?

Actually building it (the catch)

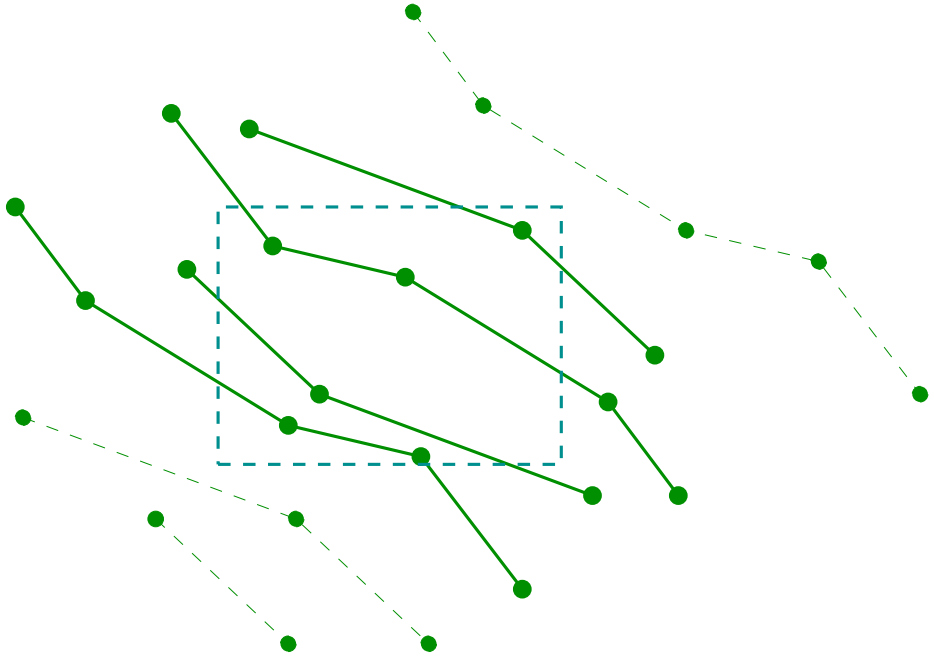
"Given a set of points in the Euclidean plane, partition it into the minimum number of chains such that the points in each chain are ordered with one dimension monotonically increasing or decreasing while the other dimension increases (directions of chains independent of each other)."

That is an $\mathcal{NP}$-hard problem, which must be solved in order to build our data structure. It's preprocessing so we can pretend it doesn't count, but $\mathcal{NP}$-hard preprocessing is less than cool.

Fortunately: Fomin, Kratsch, and Novelli [2002] constant factor approximate it in $O(n^3)$ time. Yang et al. [2007] achieve $O(\sqrt{n})$ chains, and optimal chains on worst-case data, in $O(n^{3/2})$ time—and they think they come close to the FKN approximation in practice. A constant factor is good enough for us.

Choosing the chains faster

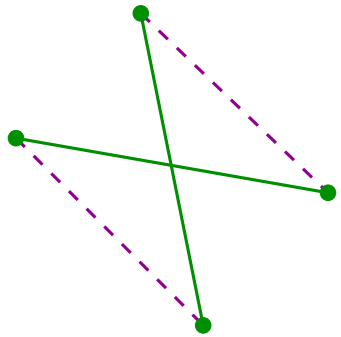
We'd like to be able to do another binary search across the chains in the other direction, to avoid looking at them all.



However, that requires the chains to be *untangled*; they cannot intersect each other.

Untangling

We can always remove any one tangle.



Note this doesn't change the number of chains, so it might seem like we could start with an optimal possibly-tangled chain decomposition and just remove the tangles.

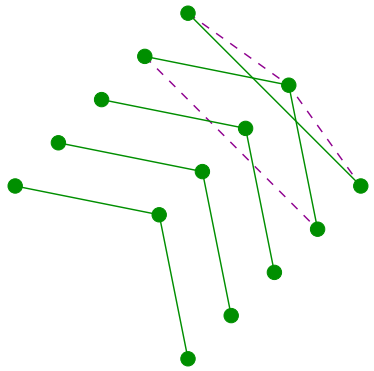
But each untangling step could create many new tangles! Will it ever end? Will it end quickly?

More untangling

- The untangling process must end eventually, because every step strictly decreases the total Euclidean length and that can only take finitely many different values.
- Side effect: the minimal number of untangled chains is the same as the minimal number of possibly-tangled chains.
- It must end after $O(n^3)$ untangling steps, using a tricky potential function [van Leeuwen and Schoone, 1981].
- From that reference: we can find each tangle by brute force in $O(n^2)$ time for overall untangling time $O(n^5)$. (Good enough for vL&S because their application was postprocessing TSP solutions.)
- This paper: we do it in $O(n \log n + kn)$ time.

A taste of the proof

An arbitrary set of chains could contain badly-behaved cases requiring many passes of untangling:



We prove that if the chains came from Supowit's [1985] chain decomposition algorithm, then the chains have special properties limiting the number of passes.

After doing the untangling, we can apply fractional cascading to get the benefit of the faster chain-selection, bringing the query time to $O(\log n + k + m)$, though losing the implicit storage space.

Conclusions

- Putting together the building blocks: constant factor approximation for chain decomposition, followed by examining every chain, gives $O(k \log n + m)$ query time, with implicit space, improving on optimal worst-case when k is smaller than $O(\sqrt{n}/\log n)$.
- Preprocessing is $O(n^3)$, or $O(n^{3/2})$ if we're willing to fake it with the result of Yang et al.
- First known adaptive data structure for this problem.
- Do untangling (in $O(n^{3/2})$ time worst case) and fractional cascading, and get $O(\log n + k + m)$ query time with linear space. (Doesn't contradict optimality because k could be $\Theta(\sqrt{n})$.)
- Untangling seems interesting for its own sake.