Convex Hull for Dynamic Data
Convex Hull and Parallel Tree Contraction

Jorge L. Vittes

June 12, 2003
(joint work with Umut Acar, and Guy Blelloch)
Motivation

- Application data is dynamic
  - word processors: slowly changing text
  - graphics: render similar images
  - mobile phone networks: continuously moving hosts

- Important to handle dynamic data efficiently
Dynamic Algorithms: Changing Data

Convex Hull for Dynamic Data
Kinetic Algorithms: Moving Data

Time = 0

Time = 1

Time = 1 + ε ... ∞
How to invent Dynamic/Kinetic Algorithms

- Just like any other algorithm. Think, ponder, divide, conquer...
- Or, use adaptivity...
Adaptivity

- Makes a standard algorithm dynamic or kinetic
- Requires little change to the standard algorithm
- Can be done semi-automatically
- Not all algorithms yield efficient adaptive algorithms
  - Will talk about this more
How does Adaptivity Work?

- Represent a computation with a dynamic dependency graph
  - nodes = data, edges = dependencies
  - Sources = input, sinks = output,

- The user can
  - change the input,
  - update the output

- Update:
  - Take a changed node,
  - Update all its children (the children are now changed)
  - Repeat until no more changed nodes
fun f (a,b,c) = 
  let
    u = a+b
  in
    if (u > 0) then
      g(u)
    else
      g(b+c)
  end

if u>0 then
  g(u)
else
  g(b+c)
fun f (a, b, c) = 
  let
    u = a + b
  in
    if (u > 0) then
      g(u)
    else
      g(b + c)
  end

if u > 0 then
  g(u)
else
  g(b + c)
fun f (a, b, c) = 
let
  u = a+b
in
  if (u > 0) then
    g(u)
  else
    g(b+c)
end

if u>0 then
  g(u)
else
  g(b+c)
fun f (a,b,c) = 
  let 
    u = a+b 
  in 
    if (u > 0) then 
      g(u) 
    else 
      g(b+c) 
  end
Adaptivity and Stability

- Adaptivity updates the result by rerunning the parts of the computation affected by the input change.
- It is efficient when the computation is “stable”, i.e., computations on “similar” inputs are “similar”.

- We apply the adaptivity technique to convex hulls.
1-D Convex Hull: Max and Min

- Just consider upper hull: Finding the maximum

- Consider two algorithms:
  - The March: March through the list
  - The Tournament: pair up the elements and take the max of each pair
Kinetic Maximum

Numbers increase/decrease continuously in time $n_i(t) = n_i + c_i t$

$n_1(t) = 2+t$

$n_2(t) = 5$

$n_3(t) = 6-t$

$t_{\text{red}} = 0$

$t_{\text{green}} = 1$

$t_{\text{blue}} = 3$
Sampling

\[ n_1(t) = 2 + t \]
\[ n_2(t) = 5 \]
\[ n_3(t) = 6 - t \]
Internal and External Events

- External event: Final result changes
- Internal event: Final result does not change but a test fails
Proof Simulation via Certificates

- A set of comparisons that prove the current maximum
  - Associate a **certificate** with each comparison
    - certificate = comparison result + failure time
  - Consider the times that a certificate fails

- **We need an algorithm that updates the result as well as the certificates**

- **Use adaptivity to obtain to do the update efficiently**
Kinetic March

Time=0

4+t  5-t  1  7+1.5t  6  3  8  4

Time=0.5+\epsilon

4+t  5-t  1  7+1.5t  6  3  8  4

Time=0.66+\epsilon

4+t  5-t  1  7+1.5t  6  3  8  4

07/15/03 Convex Hull for Dynamic Data
Kinetic March Performance

- $O(n)$: Because an item in the beginning of the list could become the maximum and it will be compared to the rest of the list
- Not acceptable because computing from scratch takes $O(n)$
The Kinetic Tournament

Time = 0

\[ f = 2 \]
The Kinetic Tournament

Time = 2 + ε

8

7

7 4 1+t 5-t

1+t

f = 6

6

6 3

8 4
The Kinetic Tournament

Time = 6 + \varepsilon

\[ f = 7 \]

Diagram of the Kinetic Tournament with nodes 7, 4, 1+t, 5-t, 6, 3, 8, and 4.
The Kinetic Tournament

Time = 7 + \varepsilon
Performance of Kinetic Tournament

- Worst case log n time per event

- This kinetic algorithm is an adaptive version of the standard tournament algorithm for finding maximum
2-D Convex Hull

- Many algorithms: Quick Hull, Graham Scan, Incremental, Merge Hull, Ultimate, Improved Ultimate...
- We will focus on the Quick Hull algorithm
- Input: A list of points $P$
- Output: The boundary points on the hull of $P$
- Example: Input = [a,b,c,d]  Output = [a,b,d]
Quick Hull Example

[A B C D E F G H I J K L M N O P]
Quick Hull Example - Filter

[A B D F G H J K M O P]

Convex Hull for Dynamic Data
Quick Hull Example - Maximum

[A B D F G H J K M O P]

07/15/03 Convex Hull for Dynamic Data 28
Quick Hull Example - Filter

07/15/03

Convex Hull for Dynamic Data

[ [A B F J ] [J O P ] ]
Quick Hull Example - Maximum

[[A B F J] [J O P]]
Quick Hull Example - Base Case

\[
\text{[ [A B] [B J] [J O P] ]}
\]
Quick Hull Example - Done

[A B] [B J] [J O] [O P]
Kinetic Quick Hull

- Two kinds of tests: Line-side and distance comparisons
- Filtering $\Rightarrow$ Line Side
- Finding the furthest point $\Rightarrow$ Distance comparisons

- Have certificates for these two events that is all
Line Side Test Fails

[ A B D F G H J K M O P]
"I" is inserted in the middle of the list

[A B D F G I H J K M O P]
Recompute Maximum

\[ [A \ B \ D \ F \ G \ I \ H \ J \ K \ M \ O \ P] \]
Dynamic Tournament - Random Trees
Dynamic Tournament - Random Trees
Dynamic Tournament - Random Trees
Distance Comparison Fails - Case 1

[A B D F G H J K M O P]
Distance Comparison Fails - Case 2

[A B D F G H J K M O P]
“B” is the new maximum

[A B D F G H J K M O P]
New recursive calls

[[A B] [J M O]]
Experiments

![Graph showing time stamps per event against input size with two curves: Quickhull and 30*log(x).]
Summary of ConvexHull Work

- Kinetic Algorithms for convex hulls using adaptivity
  - Timothy Chan’s $O(h \log n)$ algorithm: Improved “Ultimate Convex Hull”: Have a working version
  - QuickHull

- Bounce events: Can maintain convex hull of points in a box - the points bounce off of the walls of the box

- Streamlined library for kinetic convex hulls in the SML language
  - A standard algorithm can be made kinetic in a few hours of work
Parallel Tree Contraction

- Fundamental technique [Miller & Reif ‘85]
- Contraction proceeds in rounds
  - Each round shrinks the tree by a constant factor
  - Expected $O(\log n)$ rounds

Innovative Idea: Shrink the tree by local operations
Parallel Tree Contraction

- Start with a tree
- In each round:
  - Each node flips a coin
  - If leaf node then rake
  - If degree=2 and flip = H, and neighbors = T then contract
- Expected $O(\log n)$ rounds.
Example
Contracting and Raking

07/15/03 Convex Hull for Dynamic Data 49
Contracting and Raking
Contracting and Raking (cont.)
Dynamic Trees Problem

- Given a forest of weighted trees
- Operations
  1. Link: edge insertion
  2. Cut: edge deletion
  3. Queries
     - Heaviest edge in a subtree?
     - Heaviest edge on a path?
Data Structures for Dynamic Trees

- Sleator Tarjan ‘85
  - Amortized $O(\log n)$ and worst-case $O(\log n)$
- Topology Trees [Frederickson ‘93]
  - Ternary (degree-tree) trees
  - Worst case $O(\log n)$
- Top Trees [AlstrupHoLiTh ‘97]
  - Generalize Topology Trees for arbitrary degree

Idea: Trees as paths
Dynamic Parallel Tree Contraction

- Keep a copy of each round of the initial run.
- Each round affects next round.
- The nodes that “live” to the next round copy their neighbors scars, and pointers to them.
- Dependencies are based on what the node reads to do its work.
Dynamic Parallel Tree Contraction
Propagation

- If any data changes nodes whose action depend on that data are woken up.
- Wake-up only those nodes that get affected by a change.
- Run same code as in original run.
- Expected constant amount of nodes woken up per round.
Change an edge
Three nodes woken up

![Diagram showing the state transitions of a system with three nodes, labeled H, T, and X, in different states and connections.](image-url)
Nodes rerun code, more nodes woken up
Propagation continued
Experimental Results

Number Queued when remarking Nodes

\[ y = 17.566 \ln(x) - 25.282 \]
\[ R^2 = 0.9987 \]
Analyzing Power of the data structure (what it can and cannot do)

Different Applications

Analyzing Running times for:
  - Different changes.
  - Unbalanced Trees.
Conclusion and Future Work

- **ConvexHull**
  - Used adaptivity to solve the kinetic convex hull problem.
  - Encouraging results.
  - Adaptivity makes writing dynamic/kinetic algorithms a simple edition on the standard algorithm.
  - The quickhull algorithm updates based on events efficiently in the expected case.

- **Parallel Tree Contraction**
  - Efficient times $O(\log(n))$ expected time for an update.
  - Future Work:
    - More Applications