Geometria Computacional
Fecho Convexo Planar

Alexis S. L. Carvalho

10 de julho de 2001

1 Introdução

O problema do fecho convexo planar consiste de, dado um conjunto finito de pontos no plano, encontrar o polígono convexo com menor área que inclui todos esses pontos.

No projeto descrito nesse relatório, foram implementados alguns algoritmos para esse problema. Como exemplo de aplicação, também foi implementado um algoritmo que determina o diâmetro de um conjunto de pontos.

2 Implementação

Como o objetivo primordial desse projeto é permitir a visualização de algoritmos e permitir a comparação entre eles, as implementações realizadas não são ótimas: diversas otimizações que dificultariam o entendimento ou a visualização de um algoritmo não foram realizadas.

A implementação foi feita na linguagem Python, e existem três *front-ends*:

- `tkgeocomp.py`: usa Tk para a interface gráfica;
- `ggeocomp.py`: usa Gtk+/GNOME para a interface gráfica;
- `cligeocomp.py`: modo texto, para poder ser usado em scripts.

O programa foi testado em Python 1.5.2 e 2.1, usando gnome-python 1.4.1 .

Foram implementados os seguintes algoritmos:

- Graham
- Embrulho para Presente
- Quickhull
- Incremental
- Incremental Probabilístico

- Mergehull
- Algoritmo ótimo proposto por Chan em [chan96]
- Algoritmo ótimo proposto por Bhattacharya e Sen em [bs97]

Os dois últimos algoritmos possuem complexidade de tempo $O(n \log h)$ (pior caso e complexidade esperada, respecivamente), onde n é o número de pontos na entrada, e h é o número de vértices no fecho convexo. Como Kirkpatrick e Seidel demonstraram em [ks86] uma cota inferior de $O(n \log h)$ para esse problema quando tanto a entrada quanto a saída são levadas em conta, esses dois algoritmos são ótimos.

Os algoritmos de Graham e Mergehull são ótimos quando se considera apenas a entrada — complexidade de tempo $O(n \log n)$. Quickhull possui complexidade de tempo média $O(n \log n)$ e $O(n^2)$ no pior caso. Incremental Probabilístico possui complexidade esperada $O(n \log n)$. Embrulho para Presente executa em tempo $O(nh)$ e o algoritmo Incremental em tempo $O(n^2)$.

3 Utilitários

No diretório `utils` existem diversos *scripts* que geram pontos com coordenadas aleatórias, distribuídos uniformemente:

- em uma caixa (ou seja, um quadrado)
- em uma circunferência
- em um círculo
- em um anel
- em uma espiral
- de tal forma que o fecho convexo possua h vértices, para algum h fornecido pelo usuário

Esses *scripts* foram usados para gerar os arquivos de entrada usados para testar o programa.

4 Comparações

Nos gráficos abaixo, o eixo das abcissas pode representar ou o número de pontos na entrada, ou o número de vértices no fecho convexo. O eixo das ordenadas representa sempre o número de operações primitivas (cálculos de área) realizados pelos algoritmos.

"Gift" é o algoritmo Embrulho para Presente, "bhatta_sen", o algoritmo proposto por Bhattacharya e Sen, "chan", o proposto por Chan e "incr_prob" é o algoritmo Incremental Probabilístico.

4.1 Uniforme em uma circunferência

O número de vértices no fecho convexo é máximo.

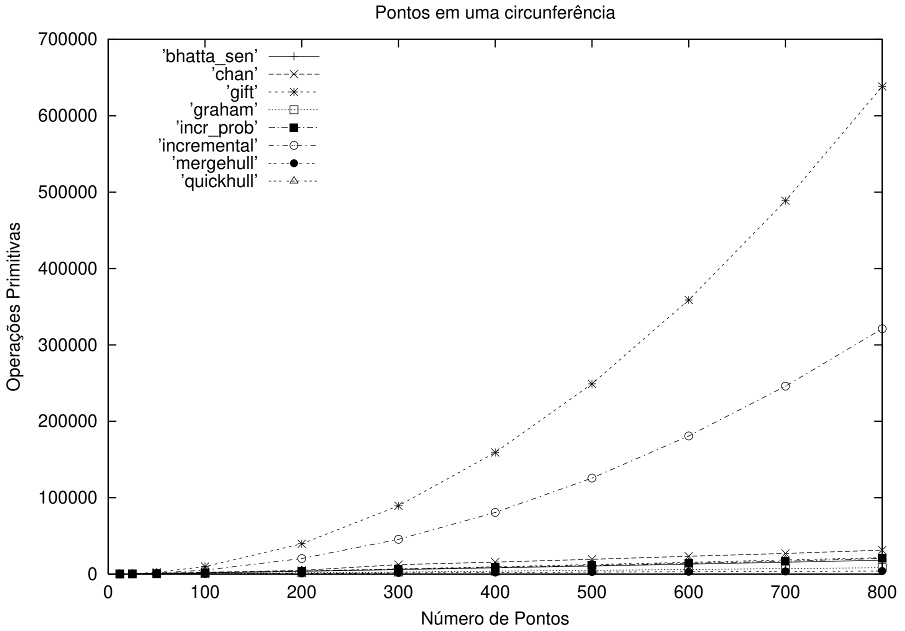

Figura 1: Pontos em uma circunferência

Esse é o pior caso de alguns algoritmos *output-sensitive*. A figura 1 permite visualizar o comportamento quadrático dos algoritmos Embrulho para Presente e Incremental.

A figura 2 mostra que os algoritmos de Graham e Mergehull, que não são *output-sensitive*, realizam menos operações nesse caso.

4.2 Uniforme em uma caixa

O número de vértices cresce lentamente em comparação com o número de pontos.

A figura 3 mostra que os algoritmos "bhatta_sen" e Quickhull são os mais eficientes nesse caso, seguidos de perto pelo Mergehull.

4.3 Número de vértices no fecho convexo é constante

Novamente, Quickhull e "bhatta_sen" são os mais eficientes. Por causa do número reduzido de vértices no fecho, Embrulho para Presente e "chan" são comparáveis ao Mergehull. Graham e Incremental Probabilístico são comparáveis entre si, mas bem menos eficientes que os outros. Incremental é o pior.

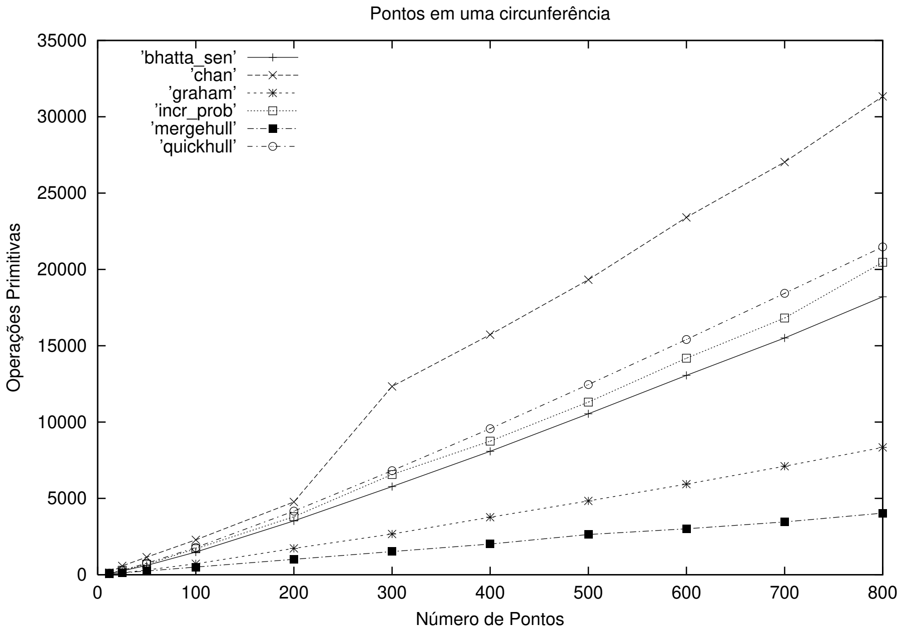

Figura 2: Pontos em uma circunferência

4.4 Número de pontos constante; número de vértices na saída aumenta

As figuras 5 e 6 permitem ver claramente a sensibilidade à saída de alguns algoritmos:

Embrulho para Presente e Incremental crescem rapidamente à medida que o número de vértices no fecho aumenta.

"Chan" dá saltos quando o número de vértices chega a 16, 64 e 256.

Incremental Probabilístico fica aproximadamente constante, com um pequeno aumento no final, que talvez possa ser explicada pela randomização do algoritmo.

Quickhull e "bhatta_sen" crescem *lentamente* com o número de vértices.

Graham e Mergehull permanecem constante — na verdade até diminuem um pouco.

5 Observações

Se levarmos em conta apenas as figuras apresentadas, podemos concluir que "bhatta_sen" é mais eficiente que Quickhull e que Mergehull é bem mais eficiente que Graham.

Mas as figuras apresentadas *não* apresentam o tempo (em segundos) que

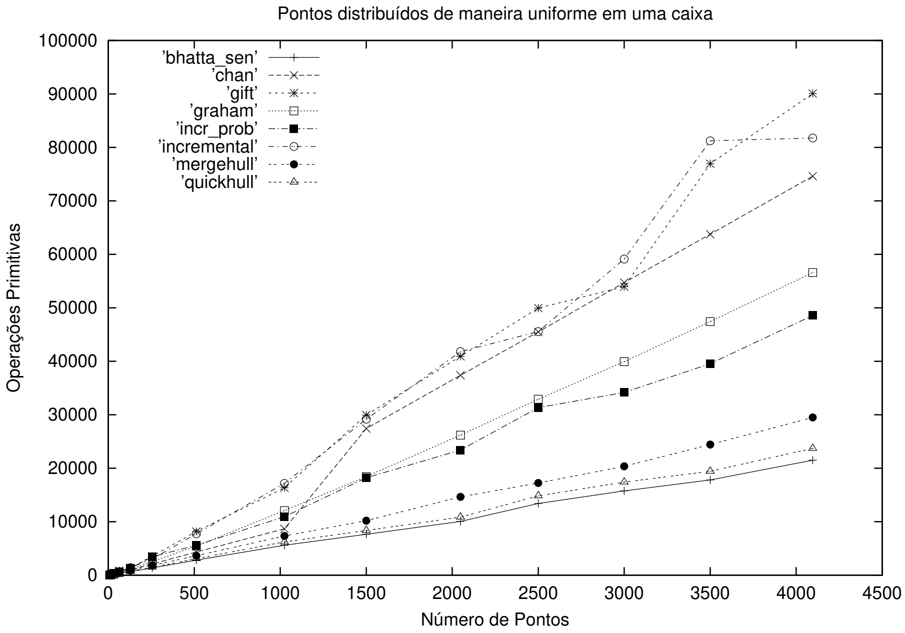

Figura 3: Pontos em uma caixa

cada algoritmo levou. Se levarmos em conta apenas esse tempo, vemos que Quickhull é um pouco mais rápido que "bhatta_sen" nos casos apresentados, e que Graham, em geral, é mais rápido do que Mergehull.

Nos algoritmos randomizados ("bhatta_sen" e Incremental Probabilístico), as medidas apresentadas são as médias obtidas depois de rodar cada teste 4 vezes.

Referências

[chan96] T. M. Chan. *Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions.* Discrete & Computational Geometry, 16:361-368, 1996. `http://http://citeseer.nj.nec.com/256494.html`

[bs97] B. K. Bhattacharya e S. Sen. *On a Simple, Practical, Optimal, Output-Sensitive Randomized Planar Convex Hull Algorithm.* J. Algorithms, 25:177–193, 1997. `http://citeseer.nj.nec.com/206645.html`

[ks86] D. G. Kirkpatrick e R. Seidel. *The ultimate planar convex hull algorithm?* SIAM Journal of Computing, 15:287–299, 1986.

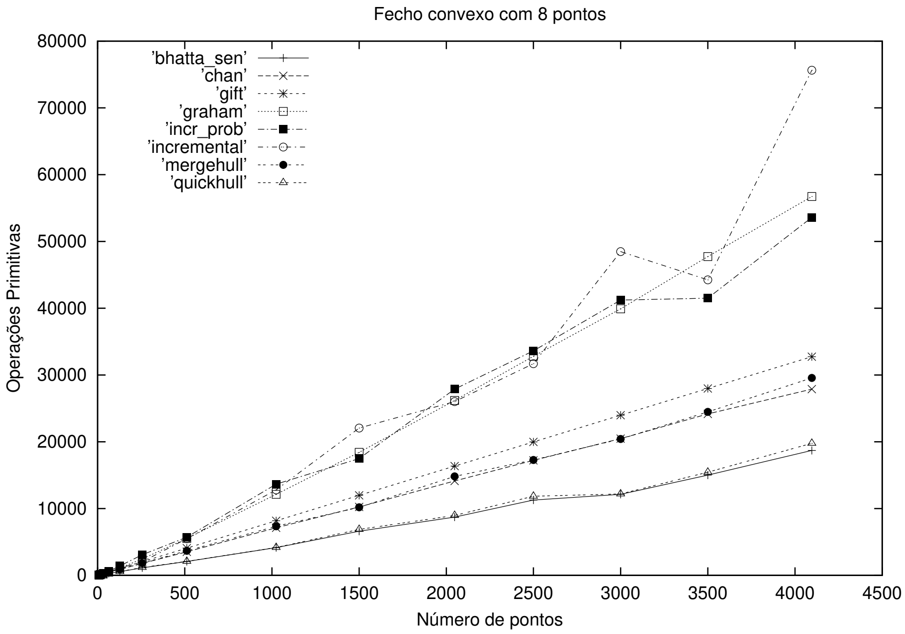

Figura 4: Número de vértices na saída é constante

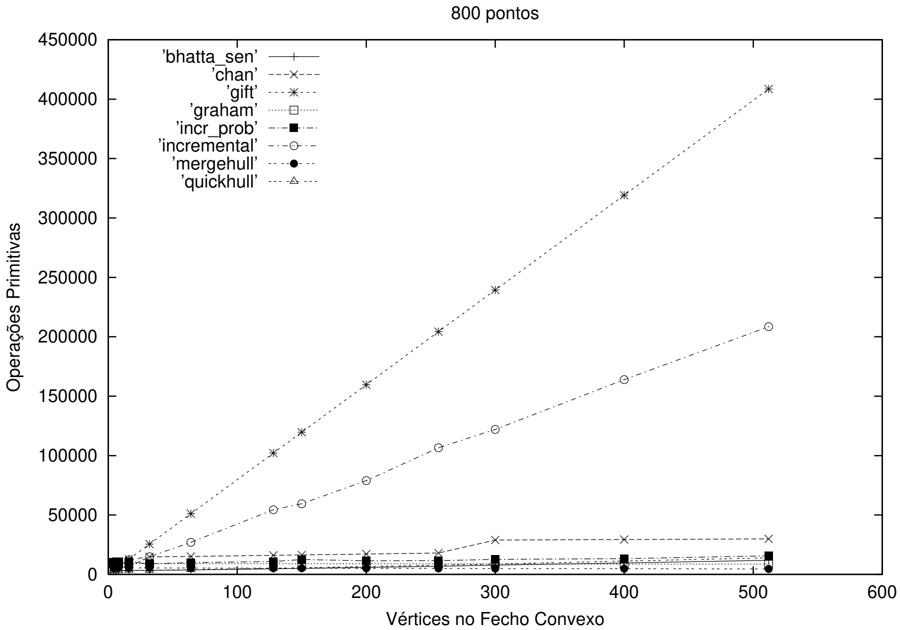

Figura 5: Número de pontos constante

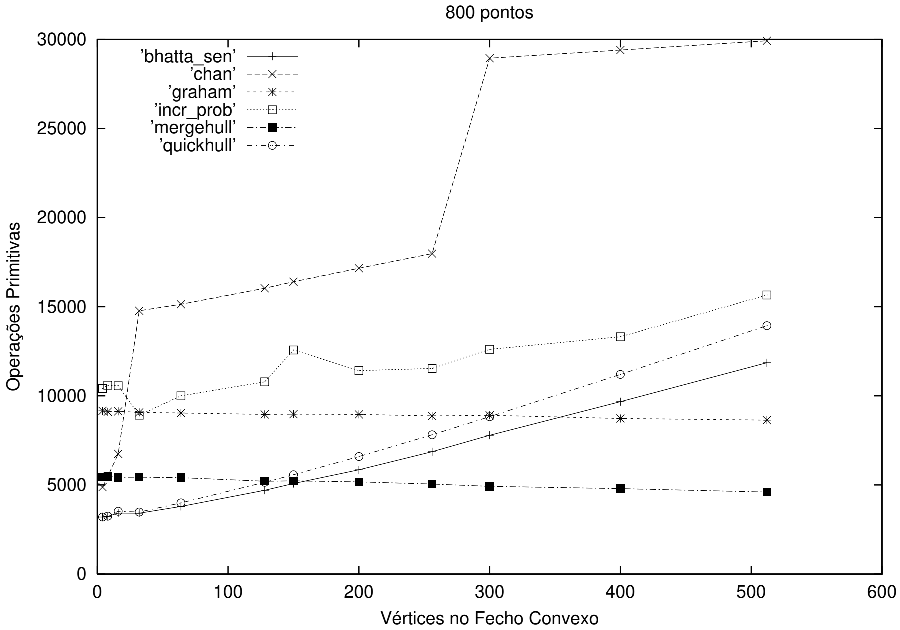

Figura 6: Número de pontos constante