Cubiertas convexas III

Dr. Eduardo A. Rodríguez Tello

Cinvestav-Tamaulipas

29 de enero del 2013

1. Cubiertas convexas IV
   - Algoritmos sensibles a la salida
   - Algoritmo de Chan
   - Tarea

Algoritmos sensibles a la salida

- El material de hoy está basado en el siguiente artículo
- T. Chan, "Optimal output-sensitive convex hull algorithms in two and three dimensions", Discrete and Computational Geometry, 16, 1996, 361–368.
- El cual pueden encontrar en la siguiente URL:
- `http://www.tamps.cinvestav.mx/~ertello/gc/Chan1996-OptimalOutput-sensitiveCH.tgz`

Algoritmos sensibles a la salida

- Los algoritmos que hemos estudiado hasta hoy en el peor caso no pueden calcular una cubierta convexa más rápido que $\Omega(n \log n)$
- Una forma de ver esto intuitivamente es observar que la cubierta convexa está ordenada ella misma a lo largo de su frontera, por lo tanto si cada punto cae en la cubierta el calcular ésta requiere alguna forma de ordenamiento
- Timothy M. Chan probó que determinar cuáles puntos están sobre la cubierta (sin ordenarlos a lo largo de la frontera) todavía requiere un tiempo $\Omega(n \log n)$

Algoritmos sensibles a la salida

- Sin embargo, ambos resultados se basan en el hecho de que todos los puntos (o al menos una fracción constante) caen en la cubierta convexa
- En la práctica ésto no es comúnmente cierto
- El algoritmo de Caminata de Jarvis que vimos la clase pasada plantea la pregunta de qué tan rápido puede ser calculada una cubierta convexa asumiendo que el tiempo de ejecución pueda ser descrito en términos del tamaño de la entrada n y de la salida h

Algoritmos sensibles a la salida

- Muchos algoritmos geométricos tienen la propiedad de que el tamaño de la salida puede ser una función del tamaño de la entrada
- Por eso el tamaño de la salida en el peor caso puede no ser un buen indicador de lo que sucede típicamente con estos algoritmos
- Un algoritmo que intenta ser más eficiente para tamaños pequeños de salida es llamado *sensible a la salida*, y su tiempo de ejecución está descrito por una función asintótica tanto del tamaño de la entrada como de la salida

1. Cubiertas convexas IV
   - Algoritmos sensibles a la salida
   - Algoritmo de Chan
   - Tarea

Algoritmo de Chan

- Dado que cualquier algoritmo para calcular la cubierta convexa toma al menos un tiempo $O(n)$, y dado que el factor $\log n$ surge del hecho de que se requiere ordenar al menos n puntos en la cubierta
- Si se supiera que solamente hay h puntos en la cubierta, entonces un tiempo razonable de ejecución es $O(n \log h)$ (Veremos adelante que éste es el óptimo)
- En 1986 Kirkpatrick y Seidel descubrieron un algoritmo complejo que corre en tiempo $O(n \log h)$, basado en un método ingenioso de poda

Algoritmo de Chan

- Este algoritmo fue considerado el mejor durante cerca de 10 años hasta que Timothy M. Chan propuso un algoritmo mucho más simple con el mismo tiempo de ejecución
- Uno de los aspectos interesantes del algoritmo de Chan es que combina inteligentemente dos algoritmos más lentos
- El algorimo de Graham y el de Caminata de Jarvis
- De está combinación Chan logró plantear un nuevo algoritmo que es más rápido que cualquiera de los dos anteriores

Algoritmo de Chan

- El problema con el algoritmo de Graham es que ordena todos los puntos
- Por lo tanto está condicionado a tener un tiempo de ejecución $\Omega(n \log n)$ sin importar el tamaño de la cubierta (número de puntos)
- Por otra parte, el algoritmo de Caminata de Jarvis puede tener un mejor desempeño si se tienen pocos vértices en la cubierta
- Pero consume un tiempo $\Omega(n)$ para cada uno de ellos

Algoritmo de Chan

- La primera observación es que si esperamos lograr un tiempo de ejecución $O(n \log h)$, sólo se puede ofrecer un factor log que depende de h
- Por lo tanto, si ejecutamos el algoritmo de Graham, estamos limitados a considerar conjuntos de tamaño h
- En realidad, cualquier polinomio en h funcionará adecuadamente, ya que para cualquier constante c tenemos que $\log h^c = c \log h = O(\log h)$

Algoritmo de Chan

- Normalmente no conocemos el valor de h por adelantado, pero vamos a suponer que por ahora tenemos un estimado para h
- Lo llamaremos m, y su valor es al menos tan grande como h, pero no demasiado grande, digamos $h \leq m \leq h^2$
- Chan ideó particionar los puntos en grupos, cada uno de tamaño m
- El número de grupos es entonces $r = \lceil n/m \rceil$

Algoritmo de Chan

- Para cada grupo se calcula su cubierta utilizando el algoritmo de Graham, lo cual toma un tiempo $O(m \log m)$ para cada uno
- El tiempo total que esto consume es $O(rm \log m) = O(n \log m)$
- Dado que $m \leq h^2$, esto resulta en $O(n \log h)$, y de esta forma se está dentro del límite de tiempo de ejecución deseado

Algoritmo de Chan

- Pero aún debemos resolver la cuestión de cómo combinar estas r cubiertas individuales para formar una sola
- Chan ideó en ejecutar el algoritmo de Caminata de Jarvis, pero tratando cada grupo como si fuera un solo y "gran punto"
- Aquí se toma ventaja del hecho de que se pueden calcular las tangentes entre un punto y un m-gono (polígono de m lados) en tiempo $O(\log m)$

Algoritmo de Chan

- Observemos que encontrar el punto de tangencia puede reducirse a una especie de búsqueda binaria
- Ya que cada "gran punto" es en sí mismo un polígono convexo, es muy posible que el próximo vértice de la cubierta venga del mismo grupo que el vértice actual

Algoritmo de Chan

- Analicemos ahora el tiempo de ejecución
- Sabemos que hay h puntos en la cubierta convexa final, y por lo tanto como máximo h de los r grupos pueden contribuir con ésta
- Eso es, hay como máximo h pasos en el algoritmo de Caminata de Jarvis
- Cada uno de estos pasos implica calcular r tangentes
- Cada tangente puede ser calculada en un tiempo $O(\log m)$

Algoritmo de Chan

- Retomando todo lo anterior, podemos deducir que el tiempo promedio que toma la fase de combinación del algoritmo es

$$O(hr \log m) = O\left(h\frac{n}{m} \log m\right) \tag{1}$$

- Combinando estas dos partes, obtenemos un tiempo total de

$$O\left(\left(n + h\frac{n}{m}\right) \log m\right) \tag{2}$$

- Mediante el supuesto que $h \leq m \leq h^2$ el tiempo total es $O(n \log h)$, como se requiere

Algoritmo de Chan

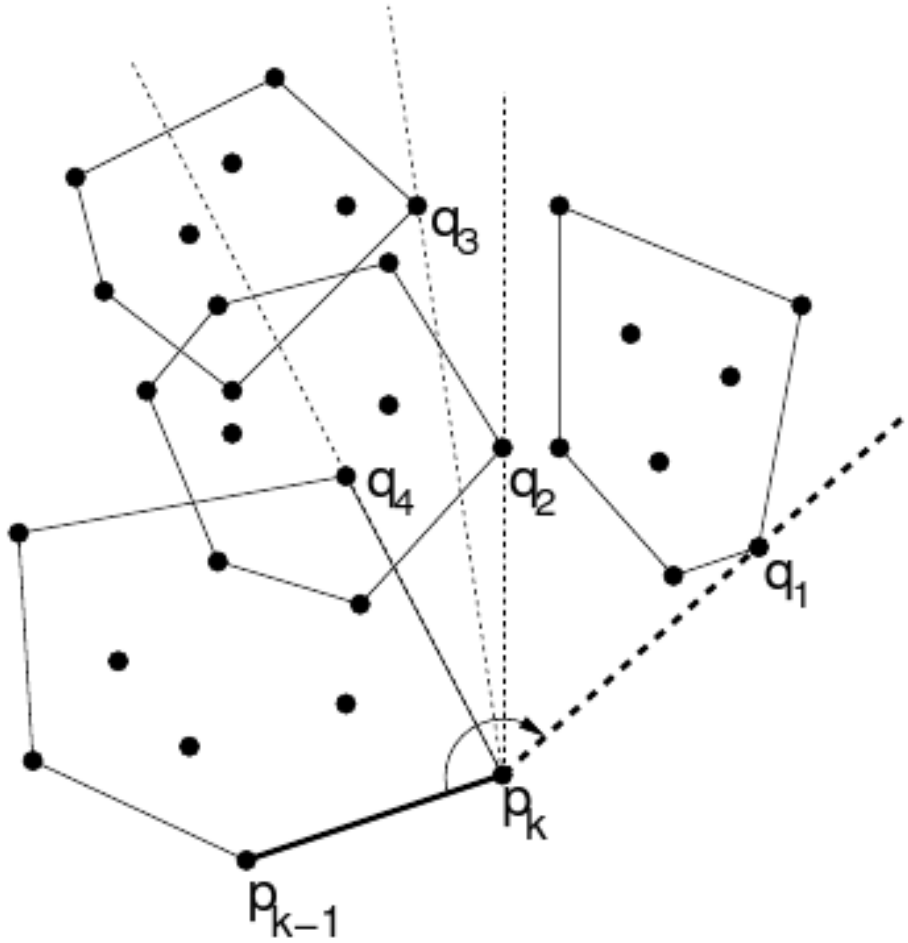
- Hay todavía un problema. No sabemos por adelantado cuánto vale h, por lo tanto no conocemos que valor de m se dará cuando el algoritmo se ejecute
- Más adelante veremos como remediar ésto
- Por ahora supongamos que conocemos ya el valor de m
- El siguiente algoritmo trabaja correctamente mientras $m \geq h$ de lo contrario regresa un mensaje de error especial

Algoritmo de Chan

PartialHull(P, m)

1. $r \leftarrow \lceil n/m \rceil$ Particionar P en subconjuntos disjuntos $P_1, P_2, \ldots, P_r$, cada uno de tamaño a lo más m
2. **for** $i = 1$ **to** r **do**
   a. Calcular conv(P_i) usando el algoritmo de Graham y almacenar los vértices en un arreglo ordenado
3. $p_0 \leftarrow (-\infty, 0)$ y $p_1 \leftarrow$ el punto de P más abajo
4. **for** $k = 1$ **to** m **do**
   a. **for** $i = 1$ **to** r **do**
      - Calcula el punto $q_i \in P_i$ que minimiza el ángulo $\angle p_{k-1} p_k q_i$
   b. $p_{k+1} \leftarrow$ el punto $q \in \{q_1, \ldots, q_r\}$ que maximiza el ángulo $\angle p_{k-1} p_k q$
   c. **if** $p_{k+1} = p_1$ **then return** $\langle p_1, \ldots, p_k \rangle$
5. **return** "Error: m es demasiado pequeño, intente de nuevo"

Algoritmo de Chan



Algoritmo de Chan

- Asumimos que podemos almacenar las cubiertas convexas del paso $(2a)$ en un arreglo ordenado así que el paso dentro del ciclo **for** del paso $(4a)$ puede ser resuelto en tiempo $O(\log m)$ usando búsqueda binaria
- De otra forma, el resto del análisis se deriva directamente de lo que hemos comentado anteriormente
- El único detalle pendiente es cómo saber qué valor asignarle a m
- Podríamos intentar con $m = 1, 2, 3, \ldots,$ hasta tener suerte y tener $m \geq h$, pero esto tomaría mucho tiempo

Algoritmo de Chan

- En realidad ésto tomaría un tiempo total de $O(nh \log h)$, lo cual es todavía peor que el algoritmo de Caminata de Jarvis
- Usar búsqueda binaria para encontrar m es una opción más eficiente que la búsqueda lineal
- Pero si pensamos en un valor grande de m (por ejemplo $m = n/2$) entonces inmediatamente quedamos atrapados en un tiempo $O(n \log n)$, el cual es demasiado lento

Algoritmo de Chan

- Para evitar ésto un procedimiento ingenioso consiste en comenzar con un pequeño valor de m (por ejemplo, $m = 1$) e incrementarlo repetidamente al doble hasta lograr que el algoritmo se ejecute exitosamente
- Corremos el algoritmo con $m = 1, 2, 4, 8, \ldots, 2^t$ hasta tener éxito
- Esto es conocido como *búsqueda por duplicación*

Algoritmo de Chan

- Desafortunadamente, es todavía demasiado lento, puesto que su tiempo de ejecución es $O(m \log^2 h)$
- Este tiempo es mejor que el de la búsqueda lineal ($O(nh \log h)$) pero todavía es posible mejorarlo
- La dependencia del tiempo de ejecución con m es solamente en el término log
- Como vimos anteriormente, mientras el valor de m esté dentro de un polinomio de h, es decir, $m \leq h^c$ para una constante c, entonces el tiempo de ejecución sera aún $O(n \log h)$

Algoritmo de Chan

- El enfoque de Chan es intentar sucesivamente valores más grandes de m, cada vez elevando al cuadrado el valor anterior, hasta que el algoritmo regrese un resultado exitoso
- Observemos que la secuencia de elecciones es por lo tanto

$$\begin{aligned} m &= 2, 4, 16, \ldots \\ m &= 2^1, 2^2, 2^4, \ldots, 2^{2^t} \end{aligned}$$

- Esto sugiere el siguiente algoritmo

Algoritmo de Chan

Hull(P)

1. **for** $t = 1, 2, \ldots$ **do**
   a. $m \leftarrow \min\left(2^{2^t}, n\right)$
   b. Invocar a PartialHull(P, m), regresando el resultado en L
   c. **if** $L \neq$ "Error" **then return** L

Algoritmo de Chan

- Notemos que 2^{2^t} tiene el efecto de elevar al cuadrado el valor anterior de m
- ¿Cuánto tiempo toma esto?
- La iteración t toma un tiempo $O(n \log 2^{2^t}) = O(n2^t)$
- Sabemos que el algoritmo se detendrá tan pronto como $2^{2^t} \geq h$, eso es si $t = \lceil \lg \lg h \rceil$ (usaremos lg para denotar el logaritmo base 2)

Algoritmo de Chan

- Usando el hecho de que $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$, el tiempo de ejecución (aproximando factores constantes) es

$$\begin{aligned}
\sum_{t=1}^{\lg\lg h} n2^t &= n \sum_{t=1}^{\lg\lg h} 2^t \leq n2^{1+\lg\lg h} \\
&= 2n2^{\lg\lg h} \\
&= 2n \lg h \\
&= O(n \log h)
\end{aligned}$$

- Lo cual es justamente lo que necesitamos

Tarea 4

- Fecha de entrega: 12 de febrero del 2013 antes de las 12h00
- Implementar el algoritmo de Chan para calcular la cubierta convexa de un conjunto de puntos
- Realizar un reporte donde se efectúe un análisis de los algoritmos de Chan, Graham y Caminata de Jarvis en cuanto a su desempeño con respecto al escalamiento del tamaño de las instancias de prueba generadas aleatoriamente
- Grafique sus resultados y comente en el reporte las conclusiones a las que llega con este experimento