Pauta Lectura 2 CC60Q: Geometría Computacional. *Towards a discipline of experimental algorithmics* Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano

Profesora: Nancy Hitschfeld Kahler. Ayudante: Diego Díaz Espinoza

October 4, 2009

Abstract

bsection

1. **Pregunta uno: ¿Qué estudia la disciplina de algoritmos experimentales? ¿Cuándo es indispensable hacer experimentación de algoritmos?**

   Algoritmos experimentales, estudia algoritmos y estructuras de datos uniendo la experimentación con el análisis teórico de los algoritmos. Es indispensable para evaluar heurísticas para problemas duros (*hard problems*), diseño en casos de test, caracterización de comportamiento asintótico de algoritmos complejos, comparación de diseños para problemas *tractables*, formulación de nuevas conjeturas y en la evaluación de criterios de optimización en actividades relacionas con el humano. Además es la clave para la transferencia de resultados de investigaciones de la publicación al código de producción proveyendo una base de algoritmos bastante comprobados.

2. **Pregunta dos: ¿Qué semejanzas y diferencias se pueden apreciar en los experimentos realizados en ciencias naturales y los usados para validar algoritmos?** Primero: se menciona la diferencia en lo apropiado de la experimentación en las ciencias naturales, donde la experimentación aparece de manera intuitiva en el método científico en donde la naturaleza es el árbitro final; en cambio, en las ciencias de la computación, el ambiente es más "artificial", donde en principio los comportamientos están totalmente determinados desde un comienzo.

   Segundo: en las ciencias naturales, los científicos pueden no sólo comprobar hipótesis basados en la experimentación, si no que también, pueden extraer otras nuevas a partir de los resultados. En ciencias de la computación, no se obtiene nada nuevo -en principio- de la experimentación, pues todo es predicible desde el comienzo.

Tercero: en ciencias de la naturaleza que utilizan herramientas computacionales, se comparan predicciones mediante modelos usando datos obtenidos de fuentes naturales. En ciencias de la computación, se mide el algoritmo en sí mismo, no sus predicciones y los resultados no pueden ser contrastados con algún estándar, pero sí comparados con otros experimentos del mismo tipo o simplemente informados a la comunidad.

3. **Pregunta tres: Enumere brevemente seis categorías en las que se pueden clasificar los aspectos a validar/chequear de un algoritmo.**

   Se enumeran todas en este caso:

   (a) Verificación de correctitud y precisión en casos extremos.

   (b) Medición del tiempo de ejecución de algoritmos exactos para problemas *NP-Hard* en instancias del mundo real.

   (c) Evaluación de la calidad de heurísticas para obtención de soluciones aproximadas en problemas *NP-Hard*.

   (d) Comparación del desempeño real de algoritmos competitivos para problemas *tractable*.

   (e) Descubrimiento del *speed-up* alcanzado al paralelizar algoritmos en la práctica.

   (f) Investigación y refinamiento en los criterios de optimización dirigidos por uso humano.

   (g) Comprobar la calidad y robustez en simulaciones de estrategias de optimización para sistemas complejos.

4. **Pregunta cuatro: ¿A qué se refiere el artículo con problemas *tractables* y problemas *intractables*? ¿Qué diferencias hay en verificar la correctitud de ambos tipos de algoritmos?**

   Problemas *tractables* son aquellos algoritmos para problemas *well-solved*, es decir, problemas con soluciones conocidas y exactas. Los problemas *intractables* son generalmente algoritmos heurísticos para problemas *NP-Hard*.

   En el caso de los *intractables* generalmente se verifica correctitud limitando las calidades de las aproximaciones, comprender cuándo una heurística se ejecuta de forma rápida y crear límites entre instancias que corren rápido y aquellas que exhiben comportamiento exponencial en el peor caso.

   En el caso de los *tractables* se verifica de forma exacta qué propiedades teóricas se cumplen en las distintas implementaciones, pueden aparecer nuevas ideas o enfoques para refinar o simplificar el algoritmo, se pueden determinar además las constantes reales de los análisis de tiempos de ejecución, por ejemplo, mediante análisis de regresión.

5. **Pregunta cinco: ¿Cuál es el procedimiento recomendado para experimentar con algoritmos? Enumere cinco aspectos que pueden distorsionar los resultados.**

   (a)

(b) Conjunto claro de objetivos: ¿qué se está preguntando? ¿qué se va a comprobar?

(c) Una vez que el diseño de experimentos está completo se deben obtener los datos. En este paso, ningún cambio debe hacerse a las configuraciones si no hasta que se obtenga todos los datos, a modo de evitar sesgos o desviaciones.

(d) Analizar los datos para responder a los objetivos planteados. Luego, se puede considerar una segunda ronda de experimentos para mejorar la comprensión de los mismos.

Se mencionan:

(a) Elección del computador o máquina (*caching*, *adressing*, *data movement*), del lenguaje (manipulación de registros, tipos nativos) o del compilador (calidad de las optimizaciones y generación de código).

(b) Calidad del código (consistencia y sofisticación de los programadores).

(c) Selección o generación de nuevas instancias (usar suficientes y variados datos para asegurar significancia).

(d) Metodología de análisis (para minimizar el impacto de la elección de la máquina).

(e) Trabajos sin relevancia: comparación de distintos lenguajes o plataformas y en particular aquellos de poco uso. Comparación de algoritmos de distinta índole: cuadráticos frente a lineales.

(f) Configuraciones erróneas: comprobación de tiempos o espacio de ejecución sin verificar previamente la aparición o existencia del comportamiento asintótico, uso de pocas instancias o no representativas, uso de código sin documentación, ignorar *test suites*, ignorar librerías existentes y usar solamente código propio, entre otros.

(g) Análisis erróneo o mal presentado: eliminación de datos que no se ajustan sin explicación o aviso alguno, presentar datos sin analizarlos y usar comparaciones de estándares no definidos.

6. **Pregunta seis: Siguiendo las recomendaciones de la lectura, diseñe un experimento para testear qué algoritmo de cerradura convexa (vistos en clases o diseñado por ustedes) es más conveniente usar de acuerdo a las características del conjunto de puntos a procesar. Un ejemplo de conjunto de puntos a procesar es cuando se sabe que pocos de ellos formarán parte de la cerradura convexa (use en su experimento por lo menos 4 tipos de conjuntos de puntos a procesar).**

   No tiene respuesta única, sin embargo una posible solución sería:

   (a) Objetivos: Verificar qué algoritmo de cerradura convexa, es más conveniente usar sobre un conjunto de puntos dado. Por conveniente se entiende que: es eficiente (en tiempo y espacio), es robusto (frente a pequeños cambios en el conjunto de entrada) y correcto (obtiene realmente la cerradura convexa). [1]

[1] Notar que es necesario definir qué se entiende por "conveniente", puesto que es lo que se está analizando, luego el objeto de estudio debe estar plenamente identificado

(b) Se fijan las condiciones ambientales del experimento: lenguaje, máquinas (memoria, cpu, *cache*, etc.), implementaciones (conjunto de posibles algoritmos como opciones), conjunto de puntos sobre los cuales se ejecutarán los experimentos. En particular, los conjuntos de puntos se pueden dividir en: conjunto vacío, conjunto unitario (un elemento), conjunto de varios elementos (más de un punto). Para el caso de los conjuntos de varios elementos, uno puede definir: conjuntos en que todos los puntos están en la cerradura convexa, conjuntos en que el 10% de los puntos está en la cerradura convexa, conjuntos en que el 20% está en la cerradura convexa, conjuntos en que el 40% está en la cerradura convexa...(aumentando cada vez por ejemplo al doble de la cantidad de puntos dentro de la cerradura convexa). Además se deben considerar dentro de estos conjuntos, aquellos en que existen casos degenerados de cada uno de los algoritmos que se tienen como opciones.

(c) Obtención de datos: correr todos los algoritmos sobre todos los conjuntos de puntos.

(d) Análisis: ordenar los algoritmos por tipo de conjunto y "conveniencia" de su uso. Buscar patrones para pares (algoritmo, conjunto de puntos).

(e) Concluir.

7. **Pregunta siete: Explique para qué sirven los sistemas de visualización de algoritmos y cuáles son sus características más interesantes para apoyar la ingeniería de algoritmos.**

   Sirven para apoyar el desarrollo, presentación y comprender los programas computacionales. Tienen la capacidad de cubrir una gran cantidad de información de modo compacto y de fácil lectura de quién observa, ayudan a tener una mirada profunda de un algoritmo, debilidades en las implementaciones y enfocarse en desarrollar modificaciones heurísticas para alcanzar un mejor desempeño.

   (a) Altos niveles de abstracción.

   (b) Crear visualizaciones de código de forma rápida y con poco esfuerzo.

   (c) Visualización sobre librerías.

   (d) Animación de algoritmos complejos, testeo en conjuntos grandes de datos.

   (e) Para *debugging*.

   (f) *Debugging* en programas concurrentes o en paralelo.

   (g) Uso de canales de información para apoyo a la divulgación (como www).