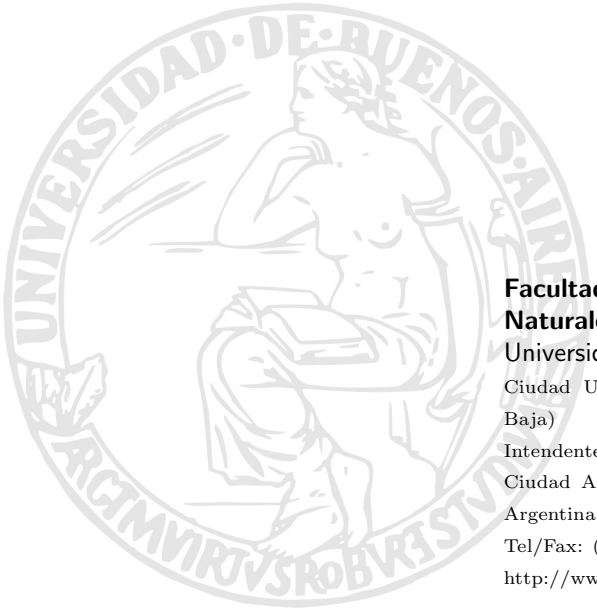
DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP1 - Backtracking - Reentrega

September 25, 2017 Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Pawlow, Dante	449/12	dante.pawlow@gmail.com



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

1 Introducción

En el presente trabajo práctico se estudia la eficiencia de la técnica algorítmica conocida como backtracking para resolver un ejercicio de optimización.

El objetivo del ejercicio es conocer el máximo número de agentes en los que se puede confiar dada una encuesta interna del grupo. En dicha encuesta cada agente puede responder si otro agente le parece confiable, desconfiable o neutral (no responde). El conjunto de agentes confiables que se debe obtener debe contar con las siguientes características: ningún agente dentro del conjunto puede decir que alguien externo es confiable ni desconfiar de alguien dentro del conjunto.

La técnica de backtracking se basa en encontrar todas las posibles soluciones válidas del problema, mediante un proceso exhaustivo. Se le puede agregar restricciones sobre los subproblemas que se deduce que no son óptimos (llamadas podas) para reducir el espectro de búsqueda de soluciones y, por lo tanto, el tiempo de cómputo.
Una poda de validez es aquella que reduce el espectro de búsqueda de posibles soluciones recortando aquellos caminos que resultan en soluciones inválidas.
Una poda de optimización, en cambio, recorta caminos que resultan en soluciones válidas, pero menos óptimas que una ya encontrada.

Notación

i: número de agentes.
a: cantidad de encuestas. C: conjunto de agentes confiables de la solución.
NC: conjunto de agentes no confiables de la solución.
C_a: conjunto de agentes confiables de un agente a dado.
NC_a: conjunto de agentes confiables de un agente a dado.

2 Algoritmos

Los algoritmos de backtracking con poda de esta sección se construyeron en base al de backtracking puro, por lo que comparten muchas características.

Para resolver el problema se modeló la estructura *Agente* que se reconoce unívocamente por su número de *ID*. Cada agente tiene dos conjuntos de enteros que representan los agentes en los que confía y en los que no.
Trabajar con conjuntos facilita muchas de las funciones involucradas en los algoritmos y permite una relación directa entre la abstracción con la que estamos trabajando y la implementación.
Se utilizó la estructura *hashset*, pero se reescribió la función de *hash* de cada Agente para que sea su *ID*, de esta manera se evita que haya colisiones (ya que esencialmente se comporta como una lista con acceso $\mathcal{O}(1)$.

2.1 Backtracking puro

Para un agente dado existen dos posibilidades: que sea confiable o desconfiable. El algoritmo de backtracking recorre recursivamente todas las formas de distribuir agentes en C y NC mediante un árbol binario de decisiones.
Este algoritmo genera una potencial solución dejando de lado todas las soluciones trivialmente inválidas, por ejemplo: que un agente esté en ambos conjuntos o que un agente no pertenezca a ninguno. Además, cada vez que se agrega un agente se verifica antes de continuar que el conjunto parcial resultante sea solución para el subconjunto de agentes tomado hasta el momento, cortando la ejecución de la rama en caso de que no lo sea.

Algorithm 1: Backtracking puro

input : C, comienza vacío; NC, comienza vacío; k: índice, comienza en 0
output: número máximo de agentes en los que se puede confiar.

```
1  if ya agregué a todos los agentes then
2  |  return el tamaño de C;
3  end
4  Aumento k en 1;
5  if Agregar el agente_k a C produce un conjunto válido then
6  |  Llamo recursivamente a esta función con el nuevo C y el viejo NC;
7  end
8  if Agregar el agente_k a NC produce un conjunto válido then
9  |  Llamo recursivamente a esta función con el viejo C y el nuevo NC;
10 end
11 Retorno el máximo de los dos caminos;
```

Si $(C \cap NC_a = \emptyset) \wedge (NC \cap C_a = \emptyset)$, entonces la solución parcial es válida.

2.2 Backtracking con poda de validez

Una poda de validez debe evitar recorrer ramas del árbol de soluciones recortando soluciones parciales que, pese a que no son inválidas, llevan a soluciones finales inválidas.

Cuando se agrega un agente al conjunto de agentes confiables se verifica que $\nexists a, b \in C / (C_a \cap NC_b \neq \emptyset) \vee (NC_a \cap C_b \neq \emptyset)$. De esta manera, se evita seguir explorando una potencial solución en la que, por ejemplo, dos agentes de C tienen opiniones distintas sobre un tercero.
Esta poda se agrega a las validaciones realizadas en el backtracking puro.

Algorithm 2: Función *esConsistente()*

```
   input  : C
   output: Respuesta booleana
 1 if C = ∅ then
 2 |  Return verdadero
 3 end
 4 for por cada agente a en C do
 5 |  for por cada agente b en C do
 6 |  |  if a confía en agentes de NC_b b o desconfía en agentes de C_b
   |  |   then
 7 |  |  |  Return falso;
 8 |  |  end
 9 |  end
10 end
11 Return verdadero;
```

2.3 Backtracking con poda de optimización

Una poda de optimización evita recorrer los caminos que producen resultados válidos pero no óptimos.
Se almacena en una variable global la cantidad máxima de agentes en los que se puede confiar encontrada en un momento dado (llamada *maxConfiables*, empezando en 0). Antes de agregar un agente al conjunto de confiables se verifica si la cantidad de agentes que queda por asignar es menor a la necesaria para sobrepasar a *maxConfiables*, se poda la rama de soluciones. Cuando se encuentra un resultado mejor que el actual, *maxConfiables* se actualiza.

Algorithm 3: Backtracking con poda de optimización

input : C, NC, comienzan vacíos; k: índice, comienza en 0;
output: número máximo de agentes en los que se puede confiar.

```
1  if ya agregué a todos los agentes then
2  |  return tamaño de C;
3  end
4  if con los agentes restantes no se llega a maxConfiables then
5  |  return 0;
6  end
7  Aumento el k en 1; if Agregar el agente_k a C produce un conjunto válido
     then
8  |  Llamo recursivamente a esta función con el nuevo C y el viejo NC;
9  end
10 if Agregar el agente_k a NC produce un conjunto válido then
11 |  Llamo recursivamente a esta función con el viejo C y el nuevo NC;
12 end
13 if el máximo retornado es mayor que maxConfiables then
14 |  Modifico maxConfiables con el nuevo valor;
15 end
16 Retorno el máximo de los dos caminos;
```

2.4 Correctitud

Estos algoritmos generan todas las soluciones posibles recorriendo la lista de agentes, asignándolos alternativamente a C y a NC y evaluando la validez de los conjuntos resultantes en cada paso. Esto garantiza que se van a encontrar todas las respuestas válidas y desechar las inválidas.
Se puede ver que el algoritmo es correcto porque de todas las respuestas válidas se evalúa cuál es la que tiene un mayor $\#C$.
La poda de validez es correcta porque devuelve el mismo conjunto de soluciones que el backtracking puro, la única diferencia es que corta las ramas de posibilidades que llevan a respuestas inválidas con mayor anticipación.
La poda de optimización devuelve un subconjunto de las soluciones que retorna el backtracking puro (potencialmente todas), pero mientras va explorando el árbol de posibilidades retiene el máximo que obtuvo hasta el momento y explora una rama sólo si puede llevar a un mayor $\#C$.

2.5 Complejidad teórica

Los algoritmos de backtracking realizan una búsqueda exhaustiva con podas sobre un árbol binario de posibilidades. En peor caso, se debe recorrer todo el árbol, produciendo una complejidad de al menos $\mathcal{O}(2^n)$.Hay que considerar, también, la complejidad que agregan las validaciones.

La complejidad de $A \cap B$ es de $\mathcal{O}(min(\#A, \#B))$, ya que se recorren todos los elementos del conjunto más chico y se verifica su pertenencia al otro conjunto con acceso $\mathcal{O}(1)$. En peor caso, la intersección se evalúa para todos los agentes cada vez que se agrega un agente, para todas las ramas del árbol. Queda, entonces, una complejidad de $\mathcal{O}(2^n \times i \sum_{k=0}^{k=i-1}(k))$.
La poda de optimización agrega solamente una validación $\mathcal{O}(1)$. La poda de validez, en cambio, compara a todos los agentes de C entre ellos, realizando una intersección entre sus conjuntos de confiables y desconfiables; la complejidad agregada es $\mathcal{O}((\#C)^2 \times i)$. Considerando todos los procesos, queda una complejidad de $\mathcal{O}(2^n \times i^2 \times \sum_{k=0}^{k=i-1}(k) \times (\#C)^2)$.
Se observa que aplicar podas puede elevar la complejidad en peor caso, pero en la siguiente sección se evaluará que puede valer la pena para reducir el tiempo de ejecución.

3 Método experimental y análisis de resultados

3.1 Complejidad empírica

Para realizar un análisis empírico de complejidad, se midieron los tiempos de ejecución de cada uno de los algoritmos para tamaños de entrada variables. Se analizó el tiempo de ejecución en función del tamaño de i.

3.1.1 Encuestas aleatorias

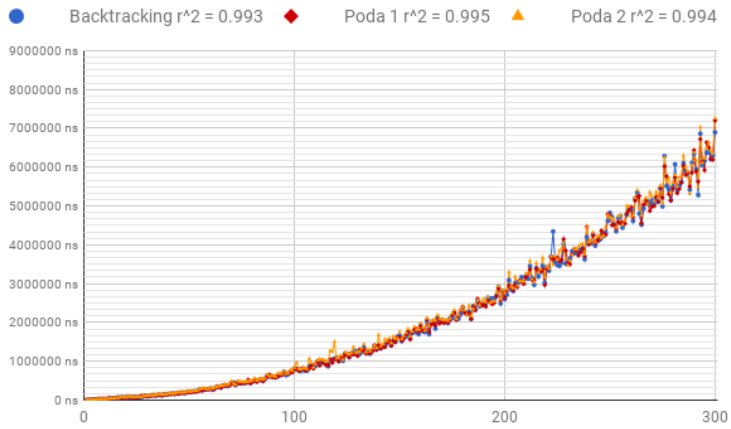


Figure 1: *Con las encuestas aleatorias no se observa una diferencia significativa entre los tiempos de ejecución de los tres algoritmos. Se observa que se ajustan a una curva polinómica con un buen r^2.*

El primer experimento que se llevó a cabo fue tomando una cantidad de agentes n de 1 a 300 y generando encuestas aleatorias para cada uno de ellos.
Para todo agente A las encuestas se generaron agregando una cantidad aleatoria (como máximo n) de agentes al conjunto de C_A. De la misma manera, se generó el conjunto NC_A, removiendo los que estuviesen presentes en C_A, para evitar inconsistencias.
El experimento se corrió 100 veces para cada n, generando nuevas encuestas para cada corrida. Para cada instancia generada se hicieron 30 corridas, y se promediaron, recortando los outliers. Se recopilaron datos de tiempo de ejecución para cada algoritmo, se realizó un promedio y se obtuvo la *figura 1*.

Calculando el *coeficiente de determinación* (r^2) se obtuvo que las curvas graficadas se aproximan con mayor confianza con una función polinomial. Esto parece contradecir la hipótesis inicial de que los algoritmos poseen una complejidad $\mathcal{O}(2^n)$. Además, se puede observar que los tres algoritmos no parecen demostrar una diferencia significativa en tiempos de ejecución.
Se plantea, la hipótesis de que nos encontramos ante un mejor caso para el algoritmo de backtracking, ocasionado por una cantidad muy grande de encuestas, lo que lleva a menos conjuntos válidos posibles y permite recortar ramas del árbol de soluciones muy rápidamente.

Teniendo esto en cuenta, se proponen los siguientes experimentos: uno con encuestas exclusivamente positivas (ningún agente desconfía de nadie) y otro con un número de encuestas acotadas para cada agente.

3.1.2 Encuestas exclusivamente positivas

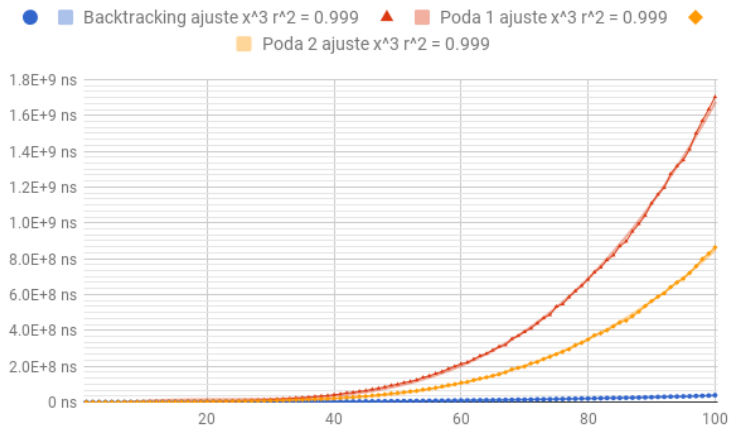


Figure 2: *Con las encuestas exclusivamente positivas nos encontramos ante un ejemplo en el que el backtracking puro tiene mejores tiempos que una poda. El ajuste polinómico presenta un r^2 muy alto.*

Para este experimento se procedió de manera muy similar al anterior, pero evitando que se generen encuestas negativas. Los resultados se graficaron en la *figura 2* y se observó que un ajuste polinómico (en particular, cúbico) presentaba el coeficiente de determinación más alto. Esto se podría atribuir a que nos encontramos frente a un caso borde, dado que no hay encuestas negativas.

Lo más interesante de este caso, es que el backtracking con poda de validez tuvo tiempos significativamente más altos que resto. Este puede ser un caso en el que la poda no recorta una cantidad significativa de posibilidades y las verificaciones que realiza aumentan considerablemente el tiempo de ejecución.

3.1.3 Cantidad de encuestas restringida a $\frac{1}{5}$ de n

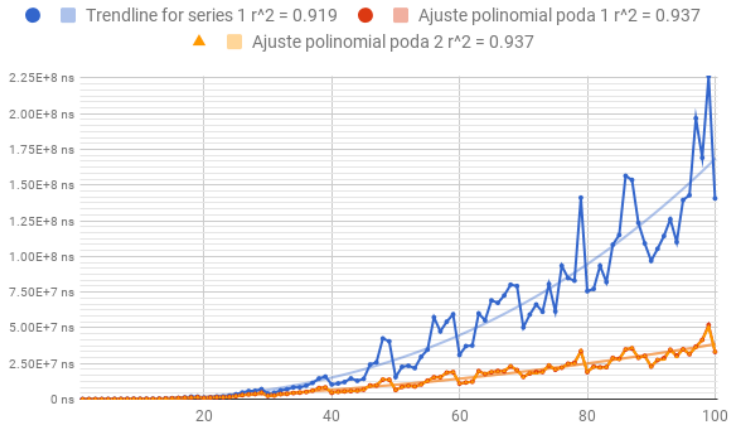


Figure 3: *Restringiendo el tamaño de* $\boldsymbol{a}$ *a* $\frac{1}{5}$ *de* $\boldsymbol{i}$ *obtenemos que se ajusta a una tendencia exponencial. Se confirma, entonces, que hay casos con orden exponencial*

En este experimento las encuestas se generaron de manera aleatoria, similarmente a como se generaron en *3.2.1.* La diferencia radica en que la cantidad máxima de individuos sobre los cuales puede opinar un agente se redujo a $\frac{1}{5}$ de la cantidad total.

Como se puede observar en la *figura 3*, los tiempos son menores que en el caso *3.2.2*, aunque con una distribución de datos mucho menos uniforme. Se observa, además, que la poda 1 y 2 presentan un tiempo promedio prácticamente idéntico.

Se sigue presentando, como en los casos anteriores, una tendencia al ajuste polinómico, aunque en este caso el coeficiente de determinación es mucho menos significativo.

3.1.4 Cantidad de encuestas restringida a $\frac{1}{25}$ de n

Por último, se restringieron las encuestas máximas de cada agente a un número aleatorio entre 1 y $\frac{1}{25}$ de la cantidad total de agentes. Esta vez se tomó un n de 1 a 40, porque los tiempos de ejecución eran significativamente mayores que en los casos anteriores. Los resultados se graficaron en la *figura 4*.

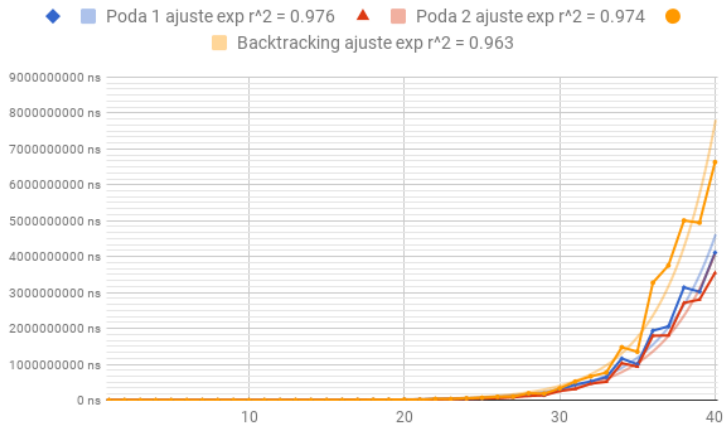


Figure 4: *Restringiendo el tamaño de* $\boldsymbol{a}$ *a* $\frac{1}{5}$ *de* $\boldsymbol{i}$ *se obtiene una tendencia exponencial más marcada (con un coeficiente de determinación más alto).*

Como se puede observar, para este caso de prueba el ajuste más apropiado es el exponencial. Para mayor certeza, se graficaron los datos en escala logarítmica en la *figura 5*, resultando en un ajuste lineal. Esto confirma las hipótesis de que los algoritmos estudiados tienen una complejidad al menos $\mathcal{O}(2^n)$.

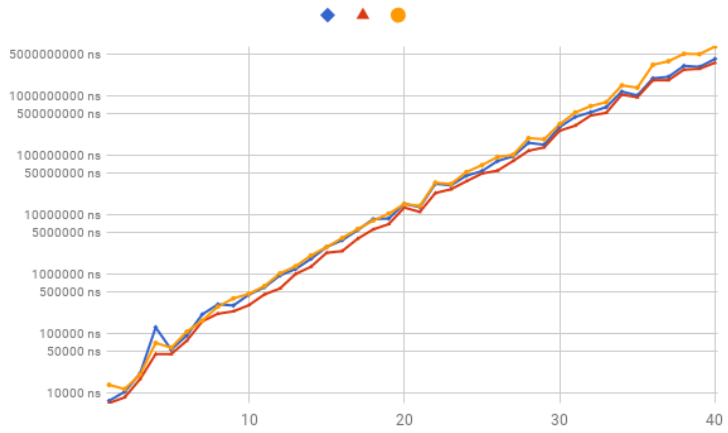


Figure 5: *Se grafican los mismos datos que para la figura 4, en escala logarítmica. Al tener un buen ajuste lineal, se confirma la tendencia exponencial de los datos.*

4 Conclusiones

A lo largo de este proceso experimental surgieron las siguientes hipótesis:

1. Los algoritmos presentados presentan complejidad exponencial.
2. Los algoritmos de backtracking podados pueden ser más eficientes que los de backtracking puro, pero también pueden aumentar la complejidad y, dependiendo del input, los tiempos de ejecución.

Por un lado se pudo comprobar que los algoritmos tienen complejidad (al menos) exponencial. Esto se pudo observar en el caso de prueba *3.2.4*.
Pese a esto, se observó que en muchos casos, los algoritmos tienen un rendimiento similar al de una complejidad polinómica. Esto puede darse porque hay casos bordes que presentan una complejidad mucho menor o porque el caso promedio del algoritmo es considerablemente mejor que el peor caso.

Por otro lado, se pudo ver en el caso *3.2.2* que aplicar una poda no necesariamente mejora el tiempo de ejecución de un algoritmo, e incluso puede pasar lo contrario.
Esto ocurre porque las podas pueden agregar procesos computacionalmente costosos y, aún así, tener que recorrer todo el árbol de soluciones. Las podas no reducen la complejidad en peor caso, peor pueden ayudar a reducir significativamente el tiempo de ejecución en el caso promedio (como se puede observar en el caso *3.2.3*.