

Algoritmos y Estructuras de Datos III

Segundo Cuatrimestre de 2017

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 1

Integrante	LU	Correo electrónico
Nicolás Ansaldi	128/14	nansaldi611@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contents

1	Backtracking	3
1.1	Descripción del problema	3
1.2	Solución Propuesta	3
1.3	Resolución	3
1.4	Pseudocódigo	4
1.5	Demostración de correctitud	4
1.6	Complejidad teórica	4
1.7	Experimentación	4
2	Backtracking (Podas)	6
2.1	Resolución	6
2.2	Pseudocódigo	6
2.3	Demostración de correctitud	6
2.4	Complejidad	7
2.5	Experimentación	7
3	Comparación	8
4	Modificaciones al informe	9
4.1	Backtracking	9
4.2	BacktrackingPodas	9
4.3	Código	9
4.4	Comparación de algoritmos	9

1 Backtracking

1.1 Descripción del problema

El problema consiste en formar el grupo más grande de agentes confiables para esto se cuenta con una tabla que representa las encuestas hechas a los agentes. La primera columna de dicha tabla representa el agente encuestado mientras que la segunda columna representa lo que dicho agente informó. Además, cada agente puede elegir informar algo como no, y si lo hace puede hacer tantas declaraciones como desee. Luego la tabla puede ser vacía como tener muchas entradas (notar que la tabla no puede ser infinita). Una vez hechas todas las encuestas se procede a decir cuál es el grupo más grande de agentes confiables. No todo conjunto de agentes es válido por lo que hay ciertas propiedades que tiene que cumplir. La primera es que si un agente dentro del conjunto confía en otro agente, ese agente también debe estar en el grupo, la segunda es que si un agente dentro del conjunto desconfía de otro agente, este no puede estar en el conjunto. Algunos ejemplos serían:

Agentes	Encuestas
1	2
2	3
3	-1
0	0

res: 2

Agentes	Encuestas
1	5
2	3
3	4
5	-1

res: 3

1.2 Solución Propuesta

La idea para resolver este problema es considerar todos los posibles casos. Para esto vimos que cada agente puede estar como no en el conjunto, o sea, se va a analizar ambos conjuntos resultantes, uno donde el agente esté en el conjunto y otro donde no esté. A la hora de agregar un agente, llamémoslo i , al conjunto primero vemos que se pueda, esto significa que va a seguir siendo un conjunto válido, para esto vamos a ver un par de cosas. La primera es que nadie en el conjunto desconfíe de i , luego veremos que i no desconfíe de nadie que está en el conjunto, también se verifica que la encuesta del agente tenga sentido lógico, o sea, que no desconfíe de sí mismo y, como puede hacer más de una declaración, que no desconfíe y confíe de un agente. Además para la rama en la cuál no se agrega al agente verificamos que tenga sentido que no este en el conjunto, esto es que nadie, que este en el conjunto, confíe en él. Una vez hecho esto, y para los otros posibles conjuntos válidos, se elegirá el conjunto con mayor cantidad de agentes.

1.3 Resolución

La técnica de backtracking consiste en pensar conceptualmente en un árbol que contenga todas las posibles soluciones. Por esto la solución que propone mi algoritmo es la siguiente:

- Partir de la raíz (ningún agente elegido).
- Comenzar por el primer agente ver si puede ser agregado al conjunto (las consideraciones descriptas en el punto anterior) y considerar que no forme parte del mismo, luego recorrer recursivamente cada una de esas 2 posibilidades para el elemento siguiente (2 ramas).
- A partir del segundo elemento, recorrer recursivamente cada una de las 2 ramas sólo si ésta es válida, en otras palabras, si puede ser agregado el agente al conjunto. Además si el agente tiene que estar en el conjunto (porque alguien en el conjunto confía en él) la rama que corresponde a no agregar al agente no se recorre ya que no sería una instancia válida del problema.
- Al llegar a una hoja que esté en el último nivel (cuando la altura del árbol es igual a la cantidad de agentes) se chequea que el conjunto resultante sea válido esto lo hacemos por que a la hora de agregar un agente al conjunto no estamos viendo que si este confía en alguien que no esta en el conjunto se pueda agregar a ese agente en el

futuro. Luego si la cantidad de agentes dentro del conjunto resultante es mayor a la que ya tenés, entonces ésta será la mejor solución. Si no, entonces me quedo con la solución que tenía antes.

1.4 Pseudocódigo

Algorithm 1 Backtracking

```

procedure BACKTRACKING(vector(pair(agente, agente)) encuestas, agentes, ConjAgentes)
  if Si no me quedan agentes para evaluar then
    if longitud(ConjAgentes) > solucion then
      solucion = longitud(ConjAgentes)
  if PuedoAgregarAgente then
    Paso Recursivo rama agregueAlAgente
  if NadieConfiaEnElAgente then
    Paso Recursivo rama NoAgregoAlAgente

```

A la hora de recorrer ambas ramas, se verifica que no genere instancias inválidas. Lo que hacen las guardas de los ifs es comprobar esa validez.

1.5 Demostración de correctitud

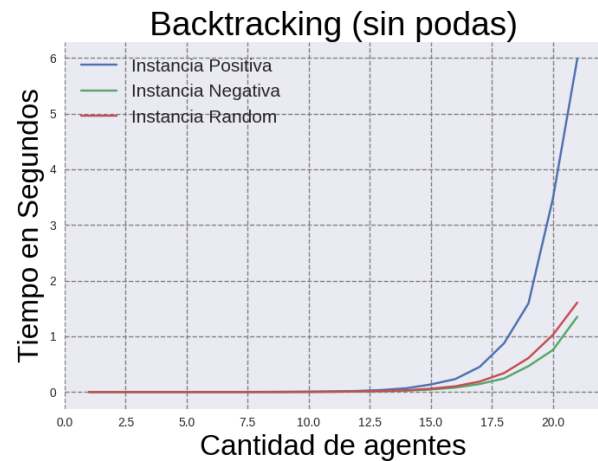
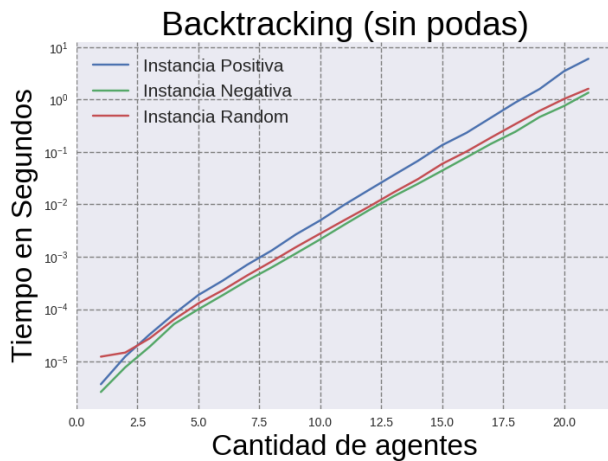
La técnica descrita anteriormente parte de pensar al problema como un árbol con todas las posibles soluciones. Cada nodo del árbol representa como se va ir modificando nuestro conjunto incluyendo o no a un determinado agente. A pesar de que nuestro algoritmo se basa en esta idea falta decir porque es correcto respecto a nuestro problema ya que como dijimos antes no recorre todos los casos. La razón es que recorre las ramas que son válidas según las condiciones del problema y no considera las inválidas. Una vez que tenemos el conjunto de soluciones posibles se queda con la mejor de ellas, en este paso (cuando ya se recorrió todo el árbol) se podrían considerar las soluciones descartadas pero serían descartas de nuevo, ya que no cumplen con lo pedido, lo que hacemos es descartalas antes de llegar al final del árbol para ahorrarnos recorrer dichas ramas. Entonces al final generamos todas las posibles soluciones y nos quedamos con la mejor.

1.6 Complejidad teórica

Como dijimos antes la idea conceptual del problema es pensarlo como un árbol de soluciones. Como este árbol contempla todas las posibles soluciones es un árbol completo, ya que por cada nodo tenemos 2 opciones, que forme parte del conjunto o que no forme (no hace falta considerar cuando alguna rama no es recorrida ya que estamos considerando complejidad en peor caso, por lo que consideramos el árbol completo). Entonces terminamos teniendo un árbol con 2^i hojas, donde "i" es la cantidad de agentes. Además para pasar de un nivel del árbol a otro, pagamos $O(a^2 * i)$ esto se debe a que vemos las encuestas del agente evaluado y las encuestas de los agentes que ya pertenecen al conjunto y chequeamos que cumplan lo descrito anteriormente. Esto lo hacemos cuando queremos ver si un agente es confiable, para la rama que no es confiable pagamos $O(a)$ para ver si efectivamente el agente evaluado puede no pertenecer al conjunto, luego nos queda la suma de las complejidades $O((a^2 * i) + (a^2 * i))$ donde nos quedamos con la más grande. Luego en el último nivel hacemos un chequeo para ver si efectivamente el conjunto es válido, esto se debe a que a la hora de agregar un agente al conjunto no estamos considerando que si éste confía en un agente y ese agente no se encuentra en el conjunto lo pueda agregar en el futuro. Para hacer esto tenemos que recorrer una vez mas el conjunto y ver que siga siendo un conjunto válido. Esto nos toma $O(i^2 a)$ pasos. Luego, una vez recorrido todo el árbol, terminamos teniendo una complejidad de $O(2^i * ((i * a^2) + (i^2 * a)))$.

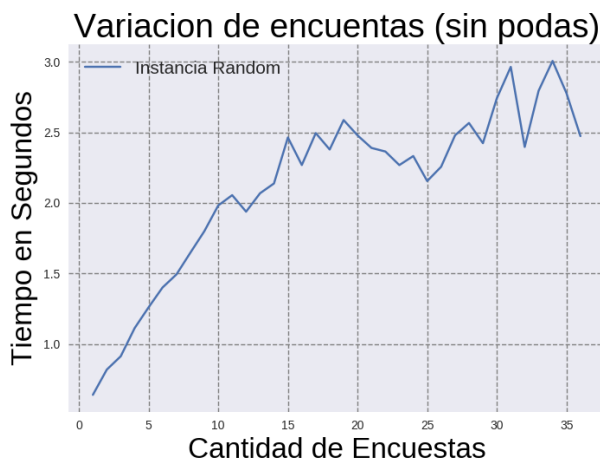
1.7 Experimentación

Para analizar la performance del algoritmo decidimos considerar diferentes entradas. Cada entrada consiste en fijar la cantidad de agentes (yendo desde 1 agente hasta 20) y tomar 100 repeticiones para la cantidad fijada. Cada repetición toma una cantidad de encuestas igual a la cantidad de agentes lo que nos queda es una relación 1 a 1, y las encuestas son elegidas de forma aleatoria. Entonces, por ejemplo, tenemos 100 repeticiones para un agente con un encuesta random, luego 100 repeticiones para 2 agentes y así sucesivamente, una vez tomadas todas las muestras se corre el algoritmo y se toma el tiempo que tarda. Una vez que se tienen los tiempos se toma un promedio de ellos. Las entradas que consideramos fueron, una donde todas las encuestas son positivas, una donde todas las encuestas son negativas y por último una donde se elige aleatoriamente si la encuesta es positiva o negativa.



En los gráficos podemos ver que la peor instancia es la positiva, esto se debe a que recorremos todas las ramas donde agregamos un agente al conjunto ya que todas las encuestas son positivas. Lo que no estaríamos recorriendo en esta instancia serían los casos donde queremos ver si podemos no agregar un agente al conjunto, ya que es posible que otro agente en el conjunto confíe en él. Pero en general estaríamos recorriendo casi todo el árbol. Para las otras instancias no se ve una diferencia muy grande.

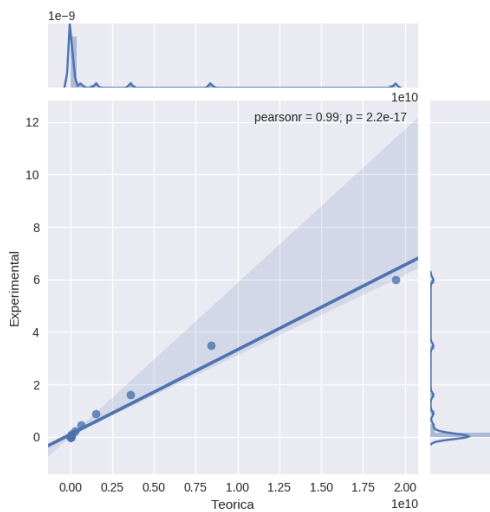
Para el segundo experimento lo que hicimos fue fijar el número de agentes a 18 e ir aumentando el número de encuestas hasta llegar a una proporción de 2 a 1. Con esto queríamos ver si a medida que aumentamos las encuestas, estas influyen en el tiempo del algoritmo. Las encuestas fueron elegidas aleatoriamente tanto el voto en sí como si es positiva o negativa. El método de medición fue el mismo que utilizamos en la experimentación anterior.



Lo que sacamos del gráfico es el aumento del tiempo en relación al aumento de las encuestas.

Por último quisimos comparar nuestra función teórica de complejidad con los resultados empíricos de nuestro algoritmo. Para esto utilizamos los resultados de la instancia positiva que, como vimos, es el peor caso para nuestro algoritmo y es lo que más se asemeja a la complejidad en peor caso. Para realizar la comparación utilizamos el coeficiente de Pearson. Este nos dice si dos variables tienen una relación o no.

En el gráfico podemos ver una relación directa casi perfecta, esto nos dice que los resultados de nuestro algoritmo y de la función teórica se comportan de manera similar, esto quiere decir que cuando uno aumenta el otro también lo hace.



2 Backtracking (Podas)

La descripción del problema como la forma de pensarlo son las mismas que el ejercicio anterior.

2.1 Resolución

Además de lo dicho en el ejercicio anterior, la idea es podar más el árbol para no recorrer ramas donde no nos interese su solución o sea un caso inválido. Para esto tenemos 2 podas, la primera consiste en no ramas que no mejoren mi mejor solución hasta el momento y la segunda consiste en hacer un análisis a futuro de la confiabilidad de los agentes. Para la primera consideremos un conjunto con "k" elementos en el, "a" representa la cantidad total de agentes e "i" representa los agentes que ya fueron evaluados, la poda consiste en que si $k + (a - i) \leq s$ (notar que "a-i" representa los agentes que faltan por evaluar), donde s es la mejor solución hasta el momento, si esto pasa no seguimos recorriendo esa rama ya que no aporta nada mejor a lo que tenemos. La otra consiste en hacer un mejor chequeo de la confiabilidad. Además de lo descrito en el ejercicio anterior, vemos que si el agente a ser agregado, llamémoslo "x", confía en otro agente "y", "y" pueda formar parte del conjunto. Para esto verificamos la confiabilidad de "y" si resulta que no podemos agregarlo al conjunto tampoco agregamos a "x" por lo que no recorremos esa posibilidad.

2.2 Pseudocódigo

Algorithm 2 BacktrackingPodas

```

procedure BACKTRACKINGPODAS(vector(pair(agente, agente)) encuestas, agentes, ConjAgentes)
  if Si no me quedan agentes para evaluar then
    if longitud(ConjAgentes) > solucion then
      solucion = longitud(ConjAgentes)
  if PuedoAgregarAgente  $\wedge$  PuedeSerMejorSolución then
    Paso Recursivo rama agregueAlAgente
  if PuedeSerMejorSolución then
    Paso Recursivo rama NoAgregoAlAgente

```

2.3 Demostración de correctitud

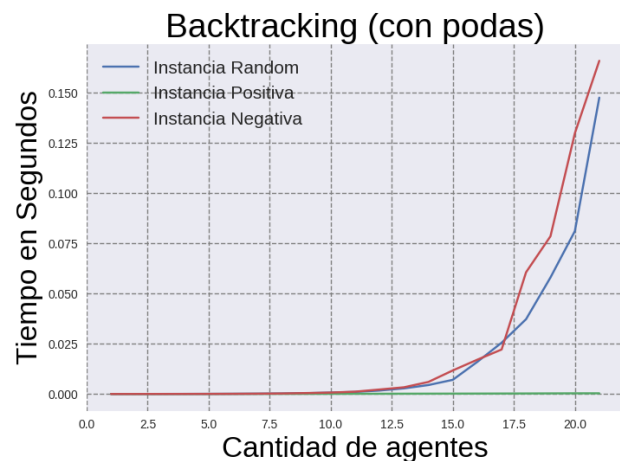
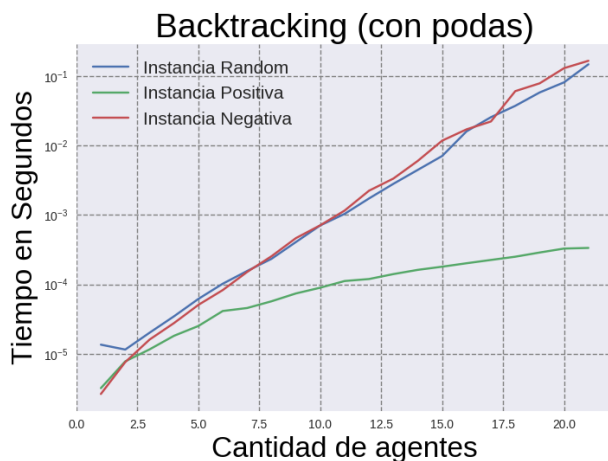
En este caso podemos decir que esta demostración es muy similar a la hecha en el algoritmo anterior. Lo que agregamos son las podas al árbol, esto hace que consideremos menos ramas. Lo que hay que ver ahora es que no estamos descartando ninguna solución válida. Pero como dijimos antes las podas sólo descartan soluciones inválidas (la diferencia con el algoritmo anterior es que la poda verifica validez a futuro y no sólo a presente) y soluciones que no mejoran la mejor solución hasta el momento. Al igual que antes descartamos soluciones en los nodos internos, en vez de descartarlas en las hojas y nos aseguramos de recorrer todas las otras soluciones.

2.4 Complejidad

Dado que las podas no agregan complejidad al algoritmo ya que se aprovechan de lo que se hacía antes y en peor caso tengo que recorrer todo el árbol, la complejidad en peor caso sigue siendo lo mismo que el ejercicio anterior. $O(2^i * ((i * a^2) + (i^2 * a)))$.

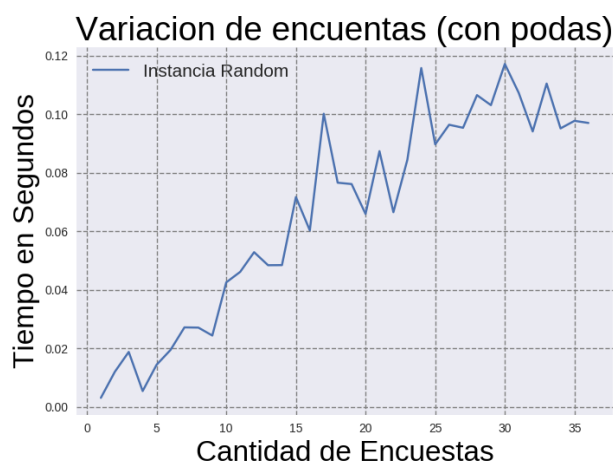
2.5 Experimentación

Para experimentar consideramos las mismas instancias que el algoritmo anterior. Las instancias fueron generadas de la misma forma que en la sección anterior.



Lo que podemos ver es que el mejor caso son las entradas positivas, esto se debe a que como la primera rama que recorre el algoritmo es determinante para la solución final, la poda que se encarga de no considerar soluciones peores o iguales poda la mayoría de las otras ramas, lo que resulta en una instancia muy buena. Mientras que las instancias random y negativas son bastantes parecidas.

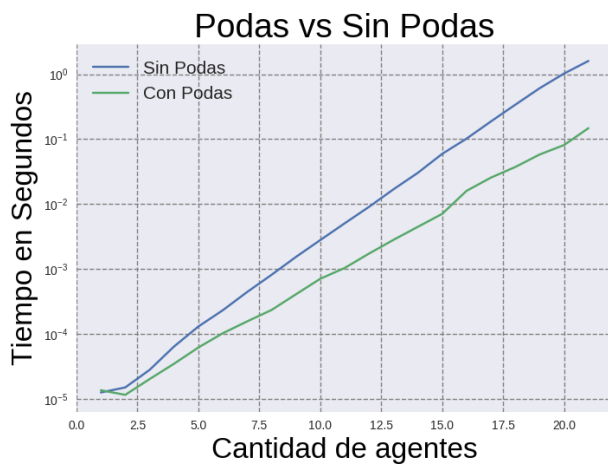
Otro experimento que quisimos ver fue el mismo que vimos en la sección anterior, que consiste en fijar la cantidad de agentes e ir variando la cantidad de encuestas. La forma de generar la muestra como la forma de tomar los tiempos son iguales a las descriptas en la sección anterior



Lo que podemos ver es que a medida que aumentamos el número de encuestas también aumenta el tiempo del algoritmo. Con lo que podemos concluir que la cantidad de encuestas afecta directamente al algoritmo. Esto se puede ver en la idea conceptual de árbol que utilizamos antes, la idea sería que, a medida que crecen las encuestas, cuesta más pasar de un nivel a otro, ya que es ahí donde recorreremos las encuestas para ver la confiabilidad de los agentes.

3 Comparación

Ahora lo que nos interesa es comparar los 2 algoritmos descritos anteriormente. La idea de esto es ver como se comportan las podas frente a no tener podas. Para esto tomamos los tiempos de ambos algoritmos sobre la entrada random descrita en las secciones anteriores. La razón por la cuál optamos por la entrada random es porque nos parece el mejor caso para comparar los algoritmos, ya que no podemos saber que se está podando y que no. Y, además, según lo que vimos antes es un mal caso para las podas, mientras que no es un caso tan malo para la versión sin podas.



Lo que podemos ver en los gráficos es que el algoritmo con podas es más rápido que el que no tiene podas. Esto se debe a que las podas me ahorran recorrer ramas que no van a aportar una mejor solución al problema y, además, cuando hacemos el chequeo de si un agente es confiable, las podas hacen un chequeo más profundo que la versión sin podas.

4 Modificaciones al informe

Aqui vamos a ver las diferencias con la primera entrega del trabajo.

4.1 Backtracking

- Cambiamos la resolución del problema agregando algunos pasos que faltaban respecto de la implementación.
- Corregimos la demostración de correctitud.
- Se modifiko la subsección de complejidad.
- Además se agrego experimentación y se comentaron mejor los gráficos, estos últimos también fueron modificados.
- Se modifiko el pseudocódigo.

4.2 BacktrackingPodas

- Se agrego una parte de correctitud.
- Se agrego experimentación y una explicación sobre la misma.

4.3 Código

- Agregamos una poda que chequea que se pueda no agregar al agente al conjunto en la parte de NoConfiable, esto lo hicimos en ambos algoritmos.
- Se modifiko el else de Confiables ya que llamaba a NoConfiable, ahora no lo hace.
- Modificamos los pasajes a las funciones para que sean por referencia en vez de ser un puntero, se cambio el código para que respete esto.

4.4 Comparación de algoritmos

Modificamos los gráficos y la explicación de los mismos.