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

1.1 Descripcion del problema

El problema consiste en formar el grupo mas grande de agentes comfiables 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 vacia como tener muchas entradas (notar que la tabla no puede ser infinita). Una vez hechas todas las encuestas se procede a decir cual es el grupo mas 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 confia en otro agente, ese agente también debe estar en el grupo, la segunda es que si un agente dentro del conjunto desconfia de otro agente, este no puede ester 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 este en el conjunto y otro donde no este. A la hora de agregar un agente, llamemoslo 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 desconfie de i, luego veremos que i no desconfie de nadie que esta en el conjunto, también se verifica que la encuesta del agente tenga sentido lógico, o sea, que no desconfie de si mismo y, como puede hacer mas de una declaración, que no desconfie y confie de un agente. 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 contega 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. Notar que la rama correspondiente a no agregar el agente al conjunto siempre se recorrerá
- Al llegar a una hoja que esté en el último nivel (cuando la altura del árbol es igual a la cantidad de agentes), 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)
        return solucion
    if PuedoAgregarAgente then
        Paso Recursivo rama agregueAlAgente
    Paso Recursivo rama NoAgregoAlAgente
```

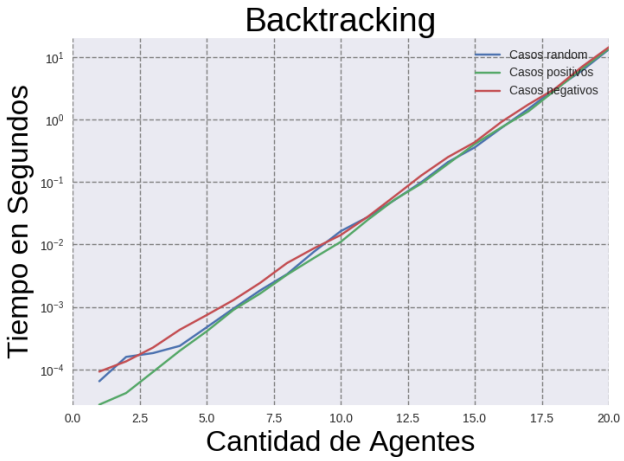
1.5 Demostración de correctitud

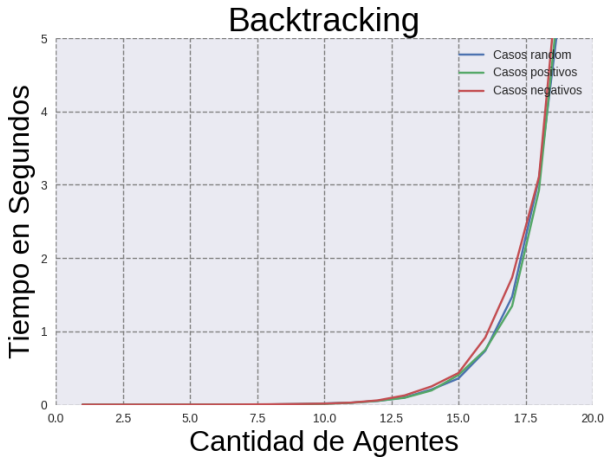
La técnica descripta anteriormente parte de pensar al problema como un árbol con todas las posibles soluciones. Luego al ver que el algoritmo recorre, por cada nodo, las 2 posibles ramas (notar que la rama de agregar solo la recorre si es posible) se puede afirmar que recorre todo el árbol. Entonces supongo que hay alguna solución que resuelve el problema pero que no esta en el árbol. Luego esto es absurdo ya que el árbol es completo ya que recorre todas las posibles combinaciones para formar las soluciones. Entonces por lo dicho anteriormente el algoritmo recorre todas las posibles soluciones y devuelve 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 (un agente) tengo 2 opciones que forme parte del conjunto o que no forme, entonces termino teniendo un árbol con 2^i nodos, donde "i" es la cantidad de agentes. Luego para pasar de un nivel del árbol a otro, hacemos $O(a^2 * i)$, donde "a" es el número de encuestas, pasos para ver si el agente es confiable y $O(1)$ para ver que no es confiable. Esto se debe a que tenemos que chequear lo que dijimos antes para ver si el agente forme parte del conjunto. 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 este confia 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 $O(2^i * (i * a^2) * (i^2 * a))$ $= O(2^i * i^3 * a3)$.

1.7 Experimentación





En las figuras podemos ver la comparación del algoritmo para diferentes entradas. La muestra fue, para cada uno de los casos 100 muestras de 22 agentes.

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 casos donde no nos interese su solución o sea un caso inválido. Para esto tenemos 2 podas, la primera consiste en no considerar conjuntos que no mejoran la respuesta y la segunda consiste en hacer un mejor análisis de la confiabilidad de los agentes. Para la primera consideremos un conjunto con "k" elementos en el, previamente habían "a" agentes mientras que "i" representan los agentes procesados, la poda consiste en que si $k + (a - i) <= s$, donde s es mi mejor solución hasta el momento, no sigo recorriendo ese rama porque se que no voy a conseguir nada mejor. La otra consiste en hacer un mejor chequeo de la confiabilidad además de lo descripto en el ejercicio anterior, vemos que si el agente a ser agregado confia en otro agente, este último debería formar parte del conjunto por lo que vemos si ya está en el conjunto o si puede ser agregado, para esto vereficamos que no haya nadie que desconfie de este agente, si alguien no confia en él no se agrega al agente al conjunto.

2.2 Pseudocódigo

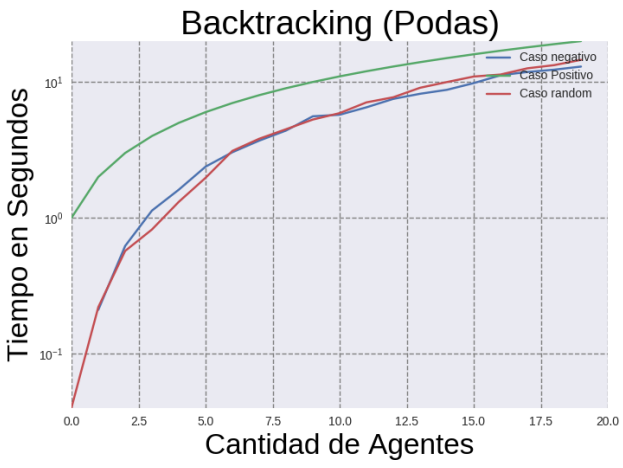
Algorithm 2 BacktrackingPodas

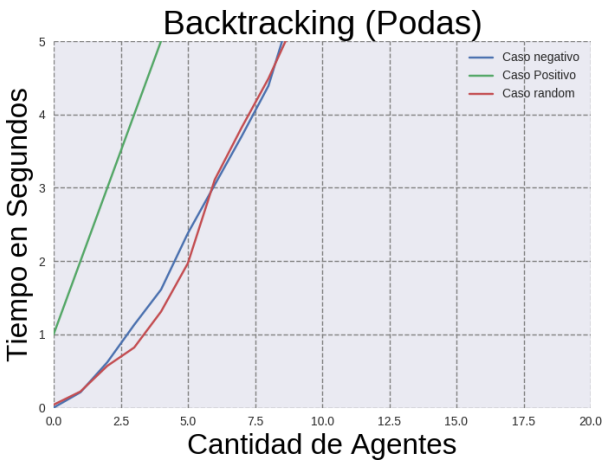
```
procedure BacktrackingPodas(vector(pair(agente, agente)) encuestas, agentes, ConjAgentes)
    if Si no me quedan agentes para evaluar then
        if longitud(ConjAgentes) > solucion then
            solucion = longitud(ConjAgentes)
        return solucion
    if PuedoAgregarAgente ∧ PuedeSerMejorSolución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

2.3 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 misma que el ejercicio anterior. $O(2^i * i^3 * a^3)$.

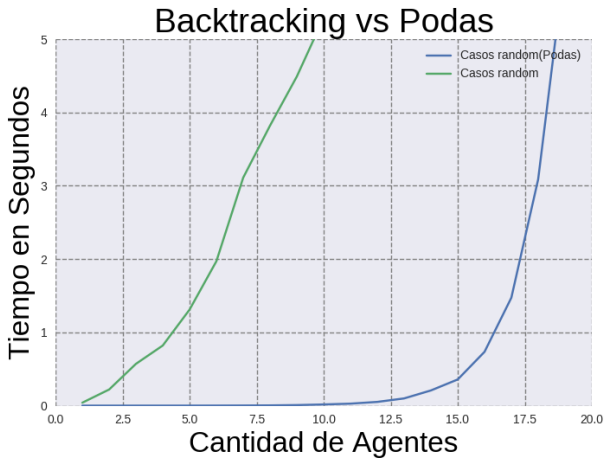
2.4 Experimentación





Al igual que en el ejercicio anterior vemos los resultados del algoritmo para diferentes entradas. La muestras fueron calculadas de la misma manera que en el ejercicio anterior.

3 Comparación



En la figura podemos ver como al tener podas el algoritmo funciona experimentalmente más rápido que la versión sin podas.