CERCA AMB RESTRICCIONS

Coneixement, Raonament i Incertesa

Enginyeria Informàtica 2021/2022 - UAB

Martí Caixal Joaniquet - 1563587

Ricard López Olivares - 1571136

Sergi Bons Fuses - 1571359

TAULA DE CONTINGUTS

1 BACKTRACKING

1.1 PREGUNTES

A. Quines són les variables?

Tenim tantes variables com paraules hi ha al Crossword. Cada una de les variables s'estructura com a una llista en forma d'atribut dins de la classe Word.

$$Word.letters = M_i,\ on\ i\ és\ el\ nombre\ de\ lletres\ de\ la\ paraula\ a\ omplir\ al\ crossword$$

B. Quin és el domini de les variables?

El domini que determinem per cada variable consisteix en totes les paraules del diccionari que coincideixen en la mida

$$D = \ parula \in Diccionari \mid mida(paraula) \equiv mida(variable)$$

C. Quines són les restriccions del problema?

La principal restricció es troba quan hi ha una intersecció entre paraules. La posició del taulell on es troben les dues paraules obliga que ambdues tinguin el mateix valor a aquella posició.

$$LletraN_{x,y} \in paraula1 \ = LletraN_{x,y} \in paraula2, \quad on\ (x,\ y)\ es\ la\ posició\ del\ crossword$$

D. Quina és la mida de l'espai de solucions inicial?

La profunditat és el nombre de variables que presenta el problema.

L'amplada és el producte del nombre de valors del domini de cada variable.
En un cas hipotètic amb 9 variables i el nombre de valors del domini per cada variable sigui 2, la fórmula queda de la següent manera:

$$Amplada = domini.count()^{variables.count()} = 2^9$$

El nostre problema, però, té diferents valors disponibles per cada variable. És a dir, el domini de la variable A pot tenir 5 valors mentre que el domini de la variable B en té 8 i el de la variable C en té 3.
En aquest cas el càlcul queda així:

$$Amplada = 5 \cdot 8 \cdot 3$$

Tant la profunditat com l'amplada depenen del taulell a omplir i el diccionari a utilitzar, així que no podem donar un valor exacte.

E. Has utilitzat alguna estratègia per reduir la profunditat mitjana de les branques?

Es va decidir utilitzar l'heurística de grau de variable. Es tracta de seleccionar la variable que està involucrada en el nombre més gran de restriccions sobre altres variables no assignades. En aquest cas és el nombre d'interseccions amb altres paraules. Ordenant les variables a LVNA utilitzant aquesta estratègia, a diferència d'ordenar-les per ordre del taulell, ajuda a millorar el temps d'execució. Més endavant és canviada per una altra.

1.2 Explicació de la resolució del problema

Per resoldre el problema hem estructurat les dades a tractar de la següent manera.

Per una banda tenim les paraules d'un diccionari. A mesura que llegíem cada una, l'afegim a un Numpy Array de 2 dimensions. La primera és la paraula, la segona dimensió és cada valor que conté la paraula. Cal també comentar que no es guarda la lletra, sinó el seu valor numèric en ASCII.
Per agilitzar la cerca de paraules s'ha decidit tenir N Numpy Array de 2 dimensions, cada un encarregat de guardar les paraules d'una mateixa mida.

Aquests Numpy Arrays es decideixen guardar a un diccionari, sent la Key la mida de les paraules que contenen.

Per altra banda també es té les paraules del taulell que s'han d'omplir. A part de guardar per cada una els valors que pot agafar, també cal guardar la seva posició, la direcció, la mida i les interseccions.
Les interseccions tenen també més d'un valor necessari a guardar: punt del taulell on es troba, índex de la paraula on es troba i un identificador de la paraula interseccionada. Aquest identificador ajuda després a veure si un valor compleix les restriccions amb una altra paraula.

Dit això, l'estructura de dades d'una paraula queda de la següent manera:

```
class Intersection:
    def __init__(self, coord, index, intersectedID):
        self.coord = coord
        self.index = index
        self.intersectedID = intersectedID


class Word:
    def __init__(self, pos, horizontal, length, remainingValues, idN):
        self.pos = pos
        self.horizontal = horizontal
        self.length = length
        self.remainingValues = remainingValues
        self.letters = [0] * length
        self.id = idN
        self.intersections = []
        self.intersectionsNumber = 0
```

L'atribut Word.letters és on es guarden els valors trobats de cada paraula durant l'algorisme.

A partir d'aquí s'ha resolt seguint l'algorisme facilitat a les transparències de teoria.

2 FORWARD CHECKING

2.1 PREGUNTES

A. Quina és la diferència entre els dos mètodes?

El Backtracking es limita a fer totes les combinacions possibles entre els valors de les variables. El principal inconvenient és que no s'anticipa a les restriccions. Pot ser que assigni un valor a la variable A, però això comporti que, a causa de les restriccions, més endavant no puguem assignar cap valor a la variable D.

El Forward Checking és capaç de solucionar l'inconvenient mencionat. Per cada valor que s'intenta assignar a la variable A, abans es comprova que aquest fet no comporti que altres variables restringides per A es quedin sense possibles valors a ser assignats. No només fa això, sinó que també s'encarrega d'eliminar els valors del domini de les variables restringides per A que ja no compleixen la restricció amb el valor assignat a A.
Aquest segon mètode és considerablement útil quan el nombre de valors a cada domini és molt elevat. Ens permet podar moltes branques de l'arbre que d'altra manera s'haurien d'explorar.

B. Com millora aquest respecte l'anterior?

Les proves de rendiment respecte les dues versions s'han portat a terme utilitzant el diccionari de 500K paraules per la raó mencionada al punt anterior.

La millora és molt important. Amb el taulell de 6x6 no es nota a simple vista, però calculant el temps amb Python es veu que la millora és unes 100 vegades més ràpid.

Fent proves amb taulells de 8x8 i 10x10, utilitzant el Forward Checking el temps d'execució sempre està per sota d'un segon. En canvi, fent servir només Backtracking el temps d'execució del 8x8 és de 5 minuts i el de 10x10 no acaba fins passada 1 hora.

Pel taulell A de 12x12 el backtracking no ha aconseguit acabar l'execució en cap moment, havent-la parat després de 3 hores al veure que no tenia un resultat òptim..

C. Heu canviat les variables, el domini o les restriccions del problema?

Variables:

Fins ara comprovàvem els valors de les variables ja assignades iterant per cada una d'elles. Al no haver de fer Forward Checking, l'accés no era gaire complicat de programar.

Si més no, el fet d'implementar el Forward Checking implica haver de fer moltes més comprovacions. Programar-ho tal com ho fèiem fins ara seria molt enrevessat així que ens hem vist obligats a afegir una nova variable que ens faciliti la tasca.

La variable afegida, doncs, és un Numpy Array de 2 dimensions on cada valor simula una casella del taulell. Els valors que té a cada iteració no són altres que els mateixos que estan guardats a cadascuna de les variables assignades. Tenint les dades estructurades d'aquesta manera millora tant el temps d'execució com el codi que cal escriure per comprovar les restriccions.

Domini:

Amb la implementació anterior del Backtracking, al no actualitzar el domini de cada variable, podíem tenir un sol domini per totes les variables d'una mateixa mida.

Ara ja no és així ja que a cada nova assignació d'una variable es modifica el domini d'altres variables. Per tant, s'ha decidit que cada clau del diccionari, en comptes de ser la mida de les paraules, serà un identificador únic per variable. Ens permet modificar el domini d'una variable, per exemple de mida = 5, i alhora mantenir intacte el domini de la resta de variables de mida = 5.

Aquesta modificació dona peu a implementar l'heurística MRV. Cada cop que actualitzem un domini d'una variable, aprofitem per guardar el nombre de valors restants al domini. Quan sigui el moment de seleccionar una nova variable no assignada, s'agafarà la que tingui el nombre de valors restants més petit.

Restriccions:

No ha fet falta canviar les restriccions.

D. Has canviat l'estratègia utilitzada per reduir la profunditat mitjana de les branques utilitzada en l'apartat anterior?

Sí, amb el Forward Checking hem decidit utilitzar el Minimum Remaining Values. Amb aquesta nova estratègia aconseguim millorar el temps d'execució per 10x.

Fent proves amb taulells de mides més grans la millora és encara més bona proporcionalment. El test amb taulell de 12x12 només aconsegueix finalitzar en pocs segons si apliquem aquesta estratègia. Sense aplicar cap estratègia però mantenint el Forward Checking, triga aproximadament 20 minuts.

2.2 Explicació de la resolució del problema

Exceptuant les òbvies modificacions necessàries per implementar el Forward Checking, s'ha fet el següent:

- Abans de seleccionar una nova variable fem una ordenació de les paraules de LVNA, sent primera la que té el valor de l'atribut remainingValues més petit.

- Guardar l'estat actual del Numpy Array de 2 dimensions que utilitzem per comprovar el domini. Els arguments, al ser passats per referència, es modifiquen també fora de la funció en qüestió. Quan modificàvem variables a crides més profundes i després resultava ser un camí sense solució, al tirar endarrere els valors es quedaven modificats. Fent un Copy.copy() i, en cas de fallar el camí, restaurar-lo, solucionem aquest problema.

3 CROSSWORD PUZZLE (AMPLICAICÓ)

3.1 PREGUNTES

A. Heu canviat les variables?

No, mantenim que cada paraula del crossword té la seva respectiva variable

B. Heu canviat el domini?

El domini de cada paraula no ha canviat. Si més no, l'ordenació dels seus valors sí s'ha vist afectat. Ara ja no estan ordenats alfabèticament, sinó que es fa una ordenació aleatoria.

La raó d'aquest canvi és la gran millora del temps d'execució que ens aporta. Creiem que funciona pel fet que les paraules, fins ara, en anar ordenades alfabèticament, eren molt semblants entre dues iteracions contigües. Per exemple, una paraula podia ser "PORTA" i la següent era "PORTE". La probabilitat que la següent tampoc complís les restriccions era molt elevada.

Ara, en no estar ordenades alfabèticament, la paraula que ve després de "PORTA" pot ser "TAULA". Si arribem a la segona paraula, significa que la primera no ha passat les restriccions. Com que la segona és molt diferent de la primera, és més probable que aquesta pugui satisfer les restriccions.

C. Heu canviat les restriccions?

Les restriccions no han estat canviades, són les mateixes que al Backtraking.

D. Heu hagut de canviar l'algorisme programat anteriorment? Per què?

El comportament i estructura general de l'algorisme segueix sent el mateix que el de l'apartat B on s'implementa el Forward Checking.

No obstant, s'ha afegit la línia de codi que s'encarrega de fer el "shuffle" als dominis.

E. Quina és l'estratègia utilitzada per reduir la profunditat mitjana de les branques?

Seguim utilitzant MRV.

3.3 Explicació de la resolució del problema

La reordenació aleatòria dels dominis es fa assignant una nova "seed" a la classe Random. L'inconvenient que trobem és que cada cas de crossword i diccionaris requerirà una "seed" diferent per tal de tenir una ordenació el més idònia possible. Com que és impossible preveure quina serà, deixem que l'algorisme s'executi durant 1 segon i comprovem si ha acabat o no. En cas de seguir executant-se, s'atura, es genera una nova "seed" i s'aplica de nou. Havent fet un nombre elevat d'execucions, de mitjana es requereix generar 4 seeds, i per tant, l'algorisme triga entre 4 i 5 segons. En cap moment hi ha hagut una execució on l'algorisme trigués més d'11 segons.

Per aconseguir-ho es fa ús del paquet "Multiprocessing" de Python. S'utilitza processos i no threads ja que els primers no comparteixen variables i els segons sí. El procés principal és l'encarregat de crear un procés que executa l'algorisme, fent-li un join passat 1 segon. Si el procés creat segueix en execució, el mata, reordena les paraules del domini i torna a crear-ne un de nou. Repeteix les accions fins que el procés amb l'algorisme hagi tingut temps d'acabar. Amb ordinadors antics hem observat que posant un "timeout" de 2 segons dona millors resultats.

També s'ha fet una versió alternativa utilitzant la senyal SIGALRM i així evitar crear més d'un procés. S'ha acabat descartant ja que només funciona amb sistemes UNIX i UNIX-like.

PROBLEMES I INCONVENIENTS

Com a punt principal, ens agradaria comentar la infinitat d'errades i bugs amb els que ens hem trobat. Els algorismes en sí no els hem trobat difícils d'entendre ni de programar. El problema ha estat a l'hora de comprovar posicions, buscar índexs relatius a altres variables, treballar amb més d'una dimensió a la vegada, etc… El resultat ha estat en trigar 30 minuts a escriure una funció i després haver de dedicar 3 hores a buscar els errors amb el depurador.

Un altra problema a destacar és a l'hora de passar arguments a les funcions. Pel que hem trobat per l'internet, Python passa variables per referència. Fins ara havíem treballat amb llenguatges on es passa el valor per defecte. Fins que no vam caure-hi, vam passar-nos hores mirant de trobar el punt on es modificava la variable sense èxit.

CONCLUSIÓ

Com a aspecte més important, hem entès la necessitat d'analitzar el problema i fer un bon disseny i estructura general del programa. És el primer treball que se'ns demana fer des de 0. Fins ara sempre ens donaven l'estructura del programa i només havíem d'omplir el contingut de les funcions. No érem conscient de la mà de problemes que pot presentar un programa mal estructurat.

Implementar els Forward Checking ens ha permès veure la importància de definir bé les variables i, sobretot, el domini. Ens ha servit per veure de primera mà la necessitat de tenir una complexitat baixa i com afecta al temps d'execució.