Ramificació i Poda (Branch and Cut)

November 29, 2010

Algorísmica Avançada. Pràctica Cinquena. Tardor 2010-2011

Els algorismes enumeratius, o de *branch and cut*, són la última artilleria de la que disposem per resoldre problemes difícils. Això vol dir que tan sols utilitzem aquesta tècnica algorísmica quan no hi ha més remei. Per això, tot sovint els algorismes matemàtics, després de fer mans i mànigues per resoldre aquest tipus de problemes, utilitzen l'expressió de "recórrer" als algorismes enumeratius.

1 Marc Filosòfic: Enfrontament als Problemes Difícils

Tal com s'ha explicat a classe de teoria, els algorismes enumeratius són els algorismes que requereixen més temps. Ja es va dir que dins l'univers de l'optimització combinatòria, és a dir, de maximització o minimització de funcions reals amb variables discretes, sempre teníem el recurs de l'enumerabilitat. Com que sempre podem expresar les solucions dels problemes com una xifra, sempre podem fer algorismes que comptin totes les xifres possibles fins trobar-ne alguna que sigui solució, si n'hi ha. O alguna d'òptima, si n'hi ha varies.

Adoneu-vos-en que comptar, el sol fet de comptar fins un nombre, ja és un problema difícil, en el sentit de que tan sols augmentant en una unitat el nombre de xifres del nombre fins al que volguem comptar, el temps que triguem a comptar-hi ja es multiplica. Tot això us ha de fer pensar que la quantitat d'informació que donem quan diem un número és de l'ordre del seu logaritme. Per il·lustrar-ho amb un exemple, quan vèiem el problema de la motxilla en programació dinàmica vem observar que tenint una eficiència nW no era polinòmic, ja que així com n sí que creix linealment amb la mida de les dades, W no. Creix més de pressa. Fent créixer la capacitat màxima de la motxilla, fem creixer la mida de les dades multiplicant-la pel logaritme de W. Per tant, que l'eficiència de la motxilla sigui nW vol dir que és exponencial respecte la mida de les dades, i per tant no és una solució polinòmica. Hi ha qui li diu *pseudo-polinòmica*, per indicar que no és pitjor que això. En qualsevol cas, la motxilla no és un problema ben resolt en el sentit que no és polinòmic.

Tot i així, no hem de confondre els termes. Que no sigui polinòmic, o que no se sàpiga resoldre en temps com a màxim proporcionals alguna potència de la mida de les dades, no vol dir que no el poguem resoldre òptimament per una gran quantitat d'instàncies. Podem trobar algorismes exactes per problemes difícils sempre que la mida de les instàncies sigui prou petita. Això ho fem amb els algorismes de ramificació i poda. Fixeu-vos que diem *algorisme exacte*. Això és perquè movent-nos entre problemes difícils, sovint val més la pena trobar solucions aproximades. Els algorismes que s'hi conformen trobant solucions aproximades es diuen *algorismes heurístics*, o directament *heurístiques*. Aquesta paraula vé del grec, i està relacionada amb "inventar", amb idees astutes. No és difícil imaginar que l'avantatge que tenen les heurístiques respecte els algorismes exactes és el temps, i la pega, que probablement no donen la solució òptima.

Els termes de *ramificació* i *poda* que donen nom aquesta tècnica fan referència a l'arbre d'exploració. Això és el T_{DFS} que dibuixa el nostre recorregut en un graf implícit, que vol dir un graf "infinitament" gran.

Respecte la ramificació, a diferència dels recorreguts en profunditat vistos en temes anteriors, ara no ens cal saber quants nodes té el graf, ni quantes arestes incideixen en cada node. Tan sols ens armem amb un node inicial i la manera de passar d'un node a cada un dels seus successors. Convé tenir molt clar l'ordre dels successors per evitar cicles en l'exploració. A aquest ordre se li diu criteri de *branching*. Sovint vé marcat pel node més prometedor, que es coneix calculant les fites.

Llavors, la poda. La poda ens ha de merèixer tot el respecte. Gràcies a la poda ens estalvien explorar grans parts del graf implícit. Recordeu que un mètode per solucionar el problema ja el tenim, i que el nostre adversari arribats aquest punt és el temps. És a dir, la poda ens obre les portes a reduir el temps. Tot el nou coneixement que poguem adquirir que ens pugui servir per reduir el temps de càlcul descartant opcions el canalitzarem a través de la poda. La més elemental de les podes és la poda basada en la millor solució en curs. És tan famosa que hi ha qui li diu PBMSC. Es tracta de deixar d'explorar les branques que a mig camí (és a dir a mitja solució) ja donen resultats pitjors que alguna altra solució, que aquesta sí que és completa, obtinguda d'alguna altra manera, ja sigui una heurística, o una solució obtinguda a l'haver arribat al final de l'exploració per alguna altra branca anterior. Finalment, respecte l'eficiència, cal tenir clar que els procediments de poda que implementem haurien de ser lineals, o quadràtics a tot estirar. És lògic. No pot ser que perdem més temps descobrint quines exploracions ens podem estalviar, que fent-les. Per això normalment els criteris de poda utilitzen relaxacions del problema. Una relaxació d'un problema vol dir ignorar alguna restricció del problema. És clar que si ignorem restriccions el valor de la solució serà millor que el que seria tenint aquestes restriccions en compte. Per tant, quan a mitja solució fem un càlcul d'aquesta solució ideal, i és pitjor que l'actual, ens oblidem de seguir en aquella branca, podem podar.

2 El Problema del Viatjant

Es dóna aquí l'enunciat del problema del viatjant:

Donat el graf complet de n nodes, $\mathcal{K}_n$, i una funció real i positiva de distàncies definida sobre les arestes $f : E(\mathcal{K}_n) \to \mathbb{R}^+$, trobar el cicle Hamiltonià de mínim cost.

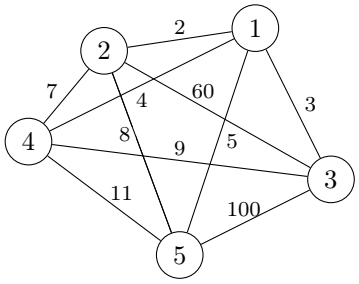
El cicle Hamiltonià d'un graf és un cicle que passi per tots els nodes sense repetir-ne cap.

3 Feina

1. En aquesta pràctica se us demana que implementeu un algorisme de ramificació i poda que dongui la solució òptima al problema del viatjant per instàncies petites. Recordeu que cal tancar el cicle.

Els algorismes enumeratius s'acostumen a presentar recursivament. Efectivament, l'esquema algorísmic de ramificació i poda conté una crida recursiva a l'adjacència de cada node. No parlarem de l'eficiència dels algorismes de ramificació i poda perquè finalment hem perdut la batalla. Aconseguiu com podeu les solucions òptimes que això ja és molt.

D'entrada, no cal que implementeu cap entrada de dades. Proveu que funcioni en un graf com el de la figura. Quan obtingueu una solució, llavors ja implementareu rutines per entrar les dades.



Caldrà fer una estructura que representi l'estat actual de decisions preses, que vol dir un node de l'exploració. I també implementar un total de cinc procediments, a part del programa principal i els de lectura de dades, que fareu posteriorment. Tenint en compte que estem parlant d'un problema de minimització, els procediments haurien de servir per:

- *heurística()* per donar fites superiors del valor òptim.
- *fita()* per la poda, o en altres paraules, per donar fites inferiors (ideals, impossibles, relaxant el problema).
- *aresta()* en l'exploració, implementa el canvi de node actual havent pres una nova decisió, o sigui, utilitzar una aresta concreta.
- *branch_and_cut()* que realitzi l'exploració recursiva.
- *motxilla()* constructor de l'objecte, que prepara els paràmetres i fa la crida inicial a branch_and_cut().

4 Lliurament

El termini de lliurament d'aquest exercici és el diumenge 19 de desembre de 2010, a les 23.55 hores. Caldrà entregar tan sols un arxiu python, *.py, posant el nom de l'autor en la primera línia, i amb el format clàssic de *ex5_CognomsNom_G.py*, posant la lletra del grup enlloc de la *G*. Procureu evitar l'ús de caracters de vuit bits en els arxius fonts. A part de la ñ, la ç, i les vocals accentuades, també és un caracter de vuit bits la l·l.