Intel·ligència artificial:
- Planificació -

Delicado Alcántara, Luis
Conejo Micó, Xavier
Sanchez Ferreres, Josep

20 de desembre de 2014

Índex

1 Model del domini

En aquest punt parlarem del domini que utilitzem en les diferents extensions, incloent-hi el bàsic.

1.1 Variables

Bàsic: actual, indica el nombre de ciutats que ha visitat fins al moment d'execució ciudades-totales, conté el nombre total de ciutats que ha de visitar

Extensió 1: Inclou les del Bàsic min-dias-totales, conté el nombre mínim de dies que ha de tenir el viatge min-dias-ciudad, conté el nombre mínim de dies que s'ha d'estar en una mateixa ciutat max-dias-ciudad, conté el nombre màxim de dies que es pot estar en una mateixa ciutat dias-totales, indica el nombre de dies que conté el viatge fins al moment d'execució dias-ciudad ?c - ciutat, indica per cada ciutat els dies que si aquesta fins al moment d'execució

Extensió 2: Inclou Extensió1 interes ?c - ciudad, conté per cada ciutat l'interès d'aquesta interes-total, indica el número d'interès acumulat per les ciutats visitades fins al moment d'execució

Extensió 3: Inclou Extensió1 precio-hotel ?h - hotel, conté per cada hotel el preu per passar una nit en aquest precio-vuelo ?v - vuelo, conté per cada vol el seu preu precio-total, indica el preu del viatge fins al moment d'execució max-precio, conté el preu màxim que pot arribar a tenir el viatge min-precio, conté el preu mínim que ha de tenir el viatge

Extensió 4: Inclou Extensió2 i Extensió3

1.2 Predicats

Totes les diferents extensions tenen els mateixos predicats, els quals expliquem a continuació:

`es-de ?c - ciudad ?h - hotel`**:** Ens serveix per establir la relació per a indicar a quina ciutat pertany un hotel

`va-a ?v - vuelo ?c1 - ciudad ?c2 - ciudad`**:** Fa una relació entre dues ciutats, de manera que indica que hi pot haver un vol entre aquestes, on ?c1 és la ciutat origen i ?c2 la destí

`ciudad-empty ?c - ciudad`: Indica si una ciutat no ha estat visitada

`esta-a ?c - ciudad`: Ens indica en quina ciutat ens trobem en aquest moment d'execució

`inicial`: Ens serveix per indicar si ens trobem en la situació que encara no hem visitat la primera ciutat del viatge

1.3 Operadors

En el cas dels operadors, és obvi que no pot ser que totes les extensions tinguin els mateixos, ja que cada una contempla més factors que l'anterior, per tant començarem explicant els operadors del Bàsic.

Aquest té només dos operadors, el primer (`ciudad-inicio`) s'encarrega d'iniciar el viatge en una ciutat inicial, de manera que actualitzem les variables perquè mantinguin la seva coherència, i el segon operador (`reservar-vol-hotel`) serveix per enllaçar una ciutat amb un altre mitjançant un vol. Veiem així que és necessari el primer operador, ja que no té cap ciutat abans que ella mateixa, i per tant és impossible relacionar aquesta ciutat inicial amb alguna d'anterior. D'aquesta manera anirem afegint ciutats al viatge fins que complim el nombre de ciutats requerit.

Veiem a continuació l'extensió 1, aquesta és molt semblant cas bàsic, però en diferencia que hem de contemplar els dies del viatge de manera que compleixi la nova restricció que el viatge tingui com a mínim un nombre de dies. Per tant els operadors seran els mateixos d'abans però ara hem de tindre cura d'actualitzar més variables. D'aquesta manera només ens queda aconseguir que el viatge compleixi que ha de tindre una duració mínima, per tant afegirem un nou operador (`mas-dias`), que en el cas que el dia no duri els dies mínims afegirà més dies.

Prosseguim amb l'extensió 2, la qual tindrà els mateixos operadors que l'extensió 1 de manera que complís els requisits d'aquesta mateixa. Però ara tenim que cada ciutat té un interès i hem d'aconseguir que la d'aquest de totes les ciutats del viatge sigui mínim, per això no ens fa falta cap operador nou, però si modificar els ja existents, de manera que només hem de fer anar sumant els interessos de les ciutats que anem visitant en la variable `interes-total`, ja que és en el problema on hem d'indicar que l'interès sigui mínim i no en el domini.

Amb l'extensió 3 passa molt similar que amb la 2, però en comptes d'interès tenim preus, per tant ara en comptes d'actualitzar l'`interes-total`, tenim que actualitzar el `precio-total`, i altre cop, no se'l domini qui s'encarrega d'aconseguir l'optimització del preu mínim, sinó en el problema on ho hem d'indicar.

L'extensió 4 és fer la unió de les extensions 2 i 3

2 Model dels problemes a resoldre

2.1 Objectes

En els models de problema dels cinc dominis d'aquesta pràctica trobem tres tipus d'objectes diferents:

Ciudad: Per una banda, tenim l'objecte ciudad, ja que, ens aporta la informació de les ciutats per on pot anar-hi el client per poder construir una ruta.

Hotel: Després tenim l'objecte hotel, aquest objecta, juntament amb el predicat `es-de` explicat anteriorment, ens aporta la informació d'on es pot allotjar el nostre client en una ciutat determinada.

Vuelo: Per últim tenim l'objecte vuelo, el qual identifica, utilitzant com en el cas anterior el predicat `va-de`, quin parell de ciutats estan connectades per poder viatjar d'una a un altre en el recorregut del client.

2.2 Estat inicial

En aquesta practica el tamany de l'estat inicial a aumentat cada cop que s'afegia una extensió per tant explicarem per cadascuna que s'ha afegit respecta a al seva predecesora.

Cas Base:

Extensió 1:

Extensió 2:

Extensió 3:

Extensió 4:

2.3 Estat final

2.4 Generador d'instàncies

Com a complement al model del domini en PDDL, hem creat un script en bash que permet generar instàncies aleatòries del problema tenint en compte diversos paràmetres per acotar-ne la mida. L'objectiu d'aquest script és generar un arxiu .pddl complert i directament *parsejable* per *Metric-FF* amb una instància de problema associada a un dels dominis –o extensions–. Aquest script l'utilitzem posteriorment per a experimentar amb el temps d'execució del solver (veure apartat 5).

Els paràmetres que rep l'script són:

- Nombre de ciutats total. L'anomenarem N. Les ciutats es generen amb els noms *viatge-i* ($i \in \{1..N\}$)
- Nombre mínim de ciutats del viatge. L'anomenarem C.
- Nombre mínim de dies del viatge. L'anomenarem D.
- Nom de la instància del problema a generar.

Totes les altres dades es generen aleatòriament. A continuació expliquem amb quina estratègia s'han generat totes les dades detalladament.

Ciutats

L'script guarda els predicats referents a ciutats com un resultat intermig a l'arxiu `.tmp/ciutats.txt`

Per les ciutats el procediment és senzill. Es creen N ciutats i es decideix aleatòriament dos valors consistents entre 1 i 5 pel màxim i mínim de dies que es pot estar a cadascuna de les ciutats[1].

Hotels

L'script guarda els predicats referents a hotels com un resultat intermig a l'arxiu `.tmp/hotels.txt`

Per a cada ciutat s'escull un nombre aleatori h i es generen d'entre 1 a 4 hotels en aquella ciutat. Els noms dels hotels es generen consecutivament (hotel-1, hotel-2, ...), però com que cada ciutat té un nombre diferent d'hotels no és possible fer la conversió directa d'*hotel-i* a *ciutat-j* i cal mirar al fitxer resultant per saber en quina ciutat està cada hotel consultant els predicats pertinents.

[1] Teniem dubtes sobre si l'enunciat es referia a màxim i mínim individuals per cada ciutat o màxim i mínim únic per totes les ciutats. Després de pensar-hi ens va semblar més coherent l'última opció i ho vam implementar així

El preu de cada hotel es decideix aleatòriament. Cal notar que el fet de crear diversos hotels amb preus diferents és simplement per provar si les extensions d'optimització son capaces de trobar l'hotel més barat de cada ciutat.

Vols

L'script guarda els predicats referents a vols com un resultat intermig a l'arxiu `.tmp/vols.txt`

Per cada parell de ciutats diferents (i, j), amb probabilitat fly_factor l'script genera un vol de *ciutat-i* a *ciutat-j* amb un preu aleatori que intenta ser coherent amb el preu dels hotels.

Per a generar el graf de vols, cal assegurar que aquest sigui connex. Per fer-ho, ens basem en un resultat obtingut en l'assignatura l'Algorísmia: "Amb alta probabilitat, si s'escullen $n\ln(n)$ arestes en un graf, on $n = |V|$, aquest graf serà connex.". Per tal que en mitja el nombre de vols s'aproximi a aquest $n\ln(n)$ calculem el fly_factor com:

$$fly\_factor = \frac{N\ln(N)}{\frac{N(N-1)}{2}}$$

Com que la probabilitat d'escollir una aresta és fly_factor, en mitjana és evident que el nombre d'arestes escollides és $N\ln(N)$. Podriem argumentar que aquesta manera de generar-ho no és exactament la mateixa ja que es poden escollir menys arestes de les que s'ha calculat. Cal veure, però, que tampoc és un requisit que el graf sigui completament connex, només cal que contingui una component connexa K amb $|K| >= C$. És interessant, doncs, generar aquest possible entrebanc per l'algoritme i veure si és capaç de resoldre'l per tal de generar unes instàncies de problema menys trivials[2].

Generar el fitxer pddl

Un cop creats els tres fitxers intermitjos, aquests s'ajunten en un fitxer pddl juntament amb tots els predicats i fluents necessaris que s'hagin d'inicialitzar per tal que funcioni el solver en el nostre domini. Hem posat especial cura en que el fitxer pddl de sortida es trobi ben tabulat i estructurat per tal de facilitar-ne la lectura si fós necessari.

Com que l'objectiu de l'script és generar el fitxer pddl complet, també s'inclou el goal pel problema, que simplement es troba *hardcoded* dins l'script.

[2]Si detectéssim que un joc de proves és irresoluble en generariem un altre. Aquesta feina es podria automatitzar però hem escollit no fer-ho ja que en cap de les proves realitzades s'ha donat cap cas on el problema generat acabés no tenint solució

3 Descripció de la metodologia de treball

4 Jocs de prova

4.1 Bàsic

4.2 Extensió 1

4.3 Extensió 2

4.4 Extensió 3

4.5 Extensió 4

4.6 Extensió 4

5 Experiment: Evolució del temps d'execució per problemes de mida creixent

En aquest apartat ens plantejem estudiar com evoluciona el temps d'execució del solver *Metric-FF* per a instancies del nostre problema de mida creixent en la extensió 3. Per fer-ho, hem generat les instàncies amb l'script explicat en 2.4 lleugerament modificat[3]. En els següents apartats es descriuen els passos que hem seguit per realitzar l'experiment juntament amb els resultats obtinguts.

5.1 Plantejament i hipòtesi

En essència, el que s'està resolent és un problema d'optimització com el que es podria resoldre amb una cerca local. De fet, el solver que utilitzaem fa servir tècniques de cerca local i heurística. És raonable, doncs, plantejar-nos com evolucionaria el temps d'execució per a mides creixents del problema i estudiar el comportament del solver.

Creiem que *Metric-FF* tindrà un temps quadràtic (o com a molt cúbic) per a les instàncies del problema. No esperem trobar el comportament de greedy característic de Fast Forward en els resultats ja que en aquest cas s'està usant el mode d'optimització i per tant assumim que l'algoritme ha de fer una cerca forçosament més exhaustiva.

5.2 Condicions de l'experiment

Per a preparar l'experiment hem generat 60 jocs de prova amb el nostre script amb mides de 5 fins a 300 ciutats.

Cal notar que per cadascuna de les N escollides hem generat 3 jocs de prova diferents per tal de fer la mitjana entre aquests, i que mesurarem els

[3]L'script genera instàncies de l'extensió 4 i crea predicats innecesaris

temps d'execució 5 vegades per cadascun dels jocs de proves. Aquest procés el fem per tal de normalitzar els resultats i eliminar qualsevol biaix que un problema particular pugui introduïr en les dades.

5.3 Resultats de l'experiment

5.4 Conclusions