FIB

Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Pràctica 2: Planificació

Grau en Intel·ligència Artificial

Algorismes Bàsics per a la Intel·ligència Artificial

Autors:

Pablo Barrenechea, Pau Hidalgo, i Miquel Rodríguez

5 de desembre de 2023

Índex

1 Altres consideracions

1.1 Desenvolupament del model

Hem començat a construïr els models partint des del més bàsic i a partir d'aquell anant afegint funcionalitats. En general, es podria dir que hem desenvolupat els models per iteracions, cada cop afegint més restriccions i nivells de dificultat.

Addicionalment, un cop vam tenir un model que complia amb l'extensió 2 (i la 3, la diferència entre les dues és mínima), vam intentar implementar també subclasses (subtypes) de forma retroactiva pels llibres predecessors i paral·lels. Tot i això, al final no els vam incorporar, ja que vam adonar-nos que no podíem tenir més d'un subtipus per objecte (però sí que podia haver-hi llibres paral·lels i predecessors a la vegada). Per tant, tot i que en alguns jocs de prova estan implementats aquestes subclasses, realment els dominis no els aprofiten.

Un altre detall que volem aclarir abans de començar amb els raonaments per les nostres decisions és el fet de que, òbviament, els dominis no van funcionar des del primer intent tal i com es presenten en aquesta pràctica.

Així doncs, cal destacar que a l'hora de dissenyar el model que utilitzaríem com domini, vam arribar a un punt en el que ens vam adonar de que estàvem dissenyant un domini de forma no seqüencial (un llibre pot ser assignat a qualsevol mes en qualsevol moment), però també es podria fer-se un disseny seqüencial respecte als mesos (començem al gener, assignem els llibres al mes actual, i passem de mes, fins arribar al desembre o complir el *goal*).

Donat que ambdós dominis s'executen d'una forma bastant eficient en un nivell bàsic, vam decidir de progressar ambdues versions de manera paral·lela, per després comparar la seva eficiència en nivells més alts a través dels experiments. Al funcionar amb lògiques molt similars a part del tema dels mesos, hem pogut aprofitar els avenços que fèiem en una versió per l'altra.

2 Domini

Aquesta secció tractarà de justificar de la manera més detallada possible el domini que hem dissenyat amb l'objectiu de representar les diferents propietats dels components que formaran el problema entregat al planificador. Tal i com està establert a l'enunciat, para a cada nivell de complexitat del problema, necessitarem implementar diferents dominis. Així doncs, en aquesta secció explicarem des del nivell bàsic fins al domini de la 3ra extensió.

D'entrada, ens centrarem en explicar en profunditat les diferents versions dels dominis no seqüencials, ja que van ser els primers que vam tenir acabats de desenvolupar. Més endavant, explicarem també com funciona la versió seqüencial, tot i que no en tanta profunditat ja que comparteix molta de la lògica amb l'altra versió.

2.1 Nivell bàsic

Aquest nivell demana un domini que permeti al planificador trobar un pla per llegir una sèrie de llibres que vindran amb uns determinats atributs definits en el problema. Els llibres podran tenir un atribut que determini si s'ha de llegir i un altre que determini si son predecessors d'un altre llibre. En cas que s'hagin de llegir o siguin predecessors d'algun llibre que s'hagi de llegir, el planificador haurà de ser capaç de assignar aquests llibres a un mes seguint la següent condició: els llibres que siguin predecessors de altres, hauran de ser llegits en qualsevol mes anterior al seu successor. En aquest nivell, un llibre només podrà tenir un predecessor.

2.1.1 Classes

Les classes utilitzades en aquest dominis seran:

- **book**: Pertany al tipus objecte, i representa tots els tipus de llibres (predecessors i no predecessors)
- **month**: Pertany també al tipus objecte i serveix com a representació dels mesos de l'any.

A l'hora de revisar i buscar optimitzacions en aquest codi, també hem provat l'opció de distingir entre els llibres predecessors com un subtipus de llibre, però pel que a temps d'execució respecta, no hi ha gaire diferència, tal i com podrem veure en la secció d'experiments.

Aleshores, la nostra decisió ha sigut la de determinar si un llibre es predecessor o no en els predicats, com s'explica en la següent subsecció.

2.1.2 Predicats

Donades les condicions i atributs que haurien de tenir els llibres, els predicats que hem considerat indicats son els següents:

- **read** ***?book***: Indica si un llibre ha sigut llegit o no.

- **to-read** ***?book***: S'encarrega de determinar si un llibre s'ha d'incloure en el pla de lectura.
- **assigned** ***?book ?month***: Determina l'assignació d'un llibre a un cert mes.
- **predecessor** ***?pred ?book***: Indica si un llibre (*pred*) es predecessor del llibre (*book*).

Els predicats **predecessor** i **to-read** són intrínsecament necessaris en aquest nivell, ja que el problema haurà de saber distingir entre els llibres segons les condicions del enunciat, que ens determinen aquests atributs. Els atres predicats **assigned** i **read** sembla que siguin duplicats (ja que un cop un llibre està assignat el considerarem llegit). La raó per duplicar un mateix atribut en dos predicats diferents és la següent: mentre que **assigned** és necessari per assignar un mes a un llibre, el predicat **read** l'aprofitarem per distingir entre els llibres que hagin sigut assignats dels que no (aquells que l'usuari havia llegit fora del pla de lectura). Així, ens estalviem buscar la variable *month* (segurament utilitzant un *exists*) en els casos en que només vulguem distingir llibres sense tenir en compte el mes.

2.1.3 Fluents

Tot i que en aquesta versió no són intrínsecament necessaris els fluents, aprofitant que els teníem a disposició n'hem utilitzat un per simplificar les relacions entre mesos. En el nostre cas, hem assignat un valor a cada mes tal que 0 - gener, 1 - febrer..., fet que ens permet comprovar que un llibre estigui assignat en un mes anterior. Els valors d'aquest fluent són constants i per tant no afegeixen tampoc complexitat de més al problema i sí que ens estalvien haver d'usar algun predicat més tant al domini com a la definició de Problema.

- **number_month** ***?month***: Fluent descrit anteriorment.

2.1.4 Accions

Finalment, en el que a les accions respecta, hem dividit el nostre domini en dos operadors:

- **assign_to_read**: És l'acció encarregada de fer que els libres que siguin predecessors a llibres que compleixin el predicat **to-read**, també compleixin aquest predicat.
- **assign_to_month**: S'encarrega d'assignar un mes a tots aquells llibres que compleixin el predicat **to-read**.

En el cas d'aquest nivell, el nostre domini no necessita més que aquestes dos accions des d'un punt de vista lògic, ja que utilitzant aquestes accions, el planificador aconseguirà tant que tots els llibres que s'hagin de llegir però no compleixin el predicat **to-read** el compleixin com que tots els llibres que s'hagin de llegir se'ls hi assigni un mes per llegir-los.

L'acció **assign_to_read** seguirà la següent lògica:

$$\forall b, b' \in books(to\_read(\mathbf{b}) \land pred(\mathbf{b'}, \mathbf{b}) \land \neg read(\mathbf{b'}) \land \neg to\_read(\mathbf{b'})) \rightarrow to\_read(\mathbf{b'})$$

És a dir, assignarà el predicat **to-read** a tots els llibres (b) que encara no estiguin llegits (assignats al pla de lectura) ni per llegir, i siguin predecessors de llibres que si que siguin candidats al plan de lectura (**to-read**).

En canvi, l'acció **assign_to_month** seguirà una lògica mes complicada:

$$\begin{aligned}
&\forall b \in books \forall month \in months[\neg read(b) \wedge to_read(b) \wedge \\
&\quad [[\neg \exists b' \in books(pred(b', b))] \\
&\qquad \vee \\
&\quad [\exists b'' \in books(pred(b'', b) \wedge read(b'') \wedge \\
&\qquad [(\neg to_read(b'')) \\
&\qquad\quad \vee \\
&\qquad (\exists month'' \in months''(assigned(b'', month'') \wedge (month < month'')))])]]] \rightarrow \\
&\rightarrow assigned(a, month_a) \wedge read(a)
\end{aligned}$$

De manera textual, el que fa l'acció és el següent: divideix en dos possibles pre-condicions que podran complir un mes m i un llibre b, tal que la primera possible condició sigui que no hi hagi cap predecessor a aquest llibre; i la segona, que en cas de haver-n'hi, per tots els predecessors que existeixin de b, el mes en el que s'assignarà el llibre ha de ser un nombre major al del mes de cadascun dels seus predecessors. Així doncs, el domini s'assegurarà de que sempre que s'assigni un mes, aquest sigui posterior al de tots els predecessors. A més, cal remarcar que b no pot estar llegit i ha de estar assignat per llegir (aquesta comprovació es fa al principi amb la intenció de descartar tots aquells llibres que no puguin aplicar per aquesta acció abans d'entrar dins d'*exists*, estalviant així memòria i temps de comput. Caldria afegir també el fet que no cal utilitzar un **forall** a l'hora de buscar predecessors, ja que segons les normes del enunciat, només pot haver-hi un predecessor per cada llibre. Així doncs, amb un *exists* ja cobrim el nombre màxim de predecessors (1), i amb la primera part del **or** cobrim l'altre possibilitat (0 predecessors).

Ambdós operadors son clarament necessaris en aquest cas, ja que l'exercici demana afegir al plan tots els llibres necessaris de manera que els predecessors estiguin assignats a qualsevol mes anterior als seus successors. Així doncs, sí o sí haurà d'existir un operador que determini els llibres necessaris a llegir i un altre per ordenar-los segons el criteri (és a dir, les nostres dues accions).

2.2 Extensió 1

En la primera escalada de nivell, el enunciat canvia les condicions dels predecessors respecte al nivell bàsic. Ara, un mateix llibre pot tenir més d'un predecessor. Tot i que pugui semblar que el enunciat pràcticament no ha canviat, el codi requereix d'algun canvi.

2.2.1 Classes

Pel que a les classes o tipus respecta, no hem afegit cap canvi, seguirem tenint els subtipus d'*object*:

- **book**: Els llibres del problema.
- **month**: La variable que representa el mes en el que s'afegirà un llibre al *planning*.

2.2.2 Predicats

Els predicats tampoc rebran cap canvi, ja que els conjunts que volem diferenciar no varien en comparació amb l'apartat previ:

- **read *?book***: Determina si un llibre ha sigut afegit al plan de lectura.
- **to-read *?book***: Distingueix els llibres que s'hagin d'afegir al plan de lectura.
- **assigned *?book ?month***: Estableix el mes en el que s'afegirà al plan un llibre *book*.
- **predecessor *?pred ?book***: Indica si un llibre (*pred*) es predecessor del llibre (*book*).

Els predicats ens permetran distingir els següents conjunts: "*assignats (al plan de lectura)*", "*predecessors*" i "*per assignar (al plan de lectura)*". Amb aquests conjunts i la informació individualitzada de cadascun dels predicats (determinant quin llibre està assignat a quin mes, o es predecessor de quin altre llibre) ja podrem resoldre tots els problemes de la següent manera:

b *predecessors* de llibres *per assignar* $\rightarrow$ per assignar.

b *per assignar* $\forall$ *predecessors* de b **assignats** en un *mes* $\rightarrow$ *assignar* **b** en un mes posterior a *mes*.

Així doncs, no cal afegir més predicats.

2.2.3 Fluents

Els fluents en aquesta versió tan sols els usàvem per establir l'ordre dels mesos i per tant tampoc hi haurà cap canvi.

2.2.4 Accions

Finalment, en les accions sí que caldrà fer una petita modificació. L'acció **assign_to_read** és exactament la mateixa que en l'apartat anterior. En canvi, en el codi pel domini bàsic havíem utilitzat

un **exists** a l'hora de comprovar que el predecessor únic del llibre estigués assignat a un mes anterior al d'aquest llibre en la funció **assign_to_month**, que en aquest cas no ens serà útil. Això es degut a que l'**exists** només comprovarà si n'hi ha algun predecessor que compleixi aquesta condició. Però, l'enunciat ens demana que la lectura d'un llibre b sigui posterior a la de tots els llibres b_i tal que $predecessor(b_i, b)$. Així doncs, com que busquem comprovar aquesta condició per tots els predecessors, utilitzarem la comanda **forall** de pddl. Aquesta ens permet iterar per tots els llibres. La usarem conjuntament amb **imply**, que ens permet establir que si compleix una condició també n'ha de complir d'altres. En aquest cas, la condició inicial a complir és que el llibre pel qual estem iterant sigui predecessor, fet que implica que ha d'estar llegit i: o bé no estava per llegir, o bé està assignat en un mes anterior (la lògica és la mateixa que en l'apartat anterior). A més, la comprovació que es feia abans de l'**exists** (pels casos en els que el llibre que es vulgui assignar no tingui cap predecessor,ja que el **exists** retornaria fals en cas de no haver-hi cap predecessor) també haurà de ser esborrada. Això es degut a que el **forall** juntament amb l'**imply**, al contrari que el **exists**, si que retornarà *true* en cas de no haver-hi cap predecessor. Així doncs, a més d'aquests dos canvis, la resta d'accions quedaran intactes.

2.3 Extensió 2

Aquesta segona extensió no modifica el concepte i condició dels llibres predecessors del primer nivell, sinó que introdueix un concepte nou: el paral·lelisme. Es tracta d'un atribut commutatiu, en que en cas que dos llibres són paral·lels entre sí i un calgui llegir-ho, llavors també s'haurà de llegir l'altre en o bé el mateix mes, o bé el mes previ, o bé el mes posterior. Així doncs, el domini haurà de ser modificat per poder diferenciar els llibres que siguin paral·lels i poder-los assignar al mes corresponent.

2.3.1 Classes

Pel que fa a les classes, vam pensar d'afegir-ne alguna de nova, però com hem vist abans,el fet d'afegir subtipus realment no plantejava beneficis importants (ja que igualment usem un predicat que acaba funcionant com a "filtrat").

- **book**: Mateix funcionament que en els nivells anteriors.
- **month**: Mateix funcionament que en els nivells anteriors.

2.3.2 Predicats

Els predicats seguiran la mateixa estructura que en els apartats anteriors, només que degut a la nova necessitat de distingir els llibres paral·lels entre si, hem creat un nou predicat: **parallel**.

- **read** ***?book***: Mateix significat que en els apartats anteriors
- **to-read** ***?book***: Mateix significat que en els apartats anteriors
- **assigned** ***?book ?month***: Mateix significat que en els apartats anteriors
- **predecessor** ***?pred ?book***: Mateix significat que en els apartats anteriors
- **parallel** ***?par ?book***: Aprofitarem aquest predicat per distingir els llibres que siguin paral·els entre si. Com que es tracta de una relació commutativa, **parallel** $?a$ $?b$ = **parallel** $?b$ $?a$.

Aquests predicats, de la mateixa manera que en els apartats previs, tornen a ser el nombre mínim de predicats necessaris per descriure el domini demanat. L'excepció és el redundant **read** que ja s'ha explicat en les anteriors seccions, i que utilitzem amb l'objectiu de comprovar si un llibre està ja assignat sense haver de buscar el mes al que estigui assignat, tal i com s'hauria de fer si només tinguéssim el predicat **assigned**. Així doncs, la nostra manera de crear el domini serà el més senzilla possible, podent executar-se també de manera senzilla.

2.3.3 Accions

Les accions d'aquest nivell requerien d'una gran quantitat de modificacions en comparació a les accions dels apartats anteriors. La idea és exactament la mateixa, utilitzar una acció amb tal d'escollir quins llibres hauran de ser escollits pel plan de lectura, i una segona acció per escollir i assignar un mes a quests llibres complint les normes establides pel enunciat, per formar així un pla de lectura vàlid:

- **assign_to_read**: S'encarrega d'escollir si un llibre s'ha de tenir en compte a l'hora de incloure'l al plan de lectura. Per a aixo, comprovarà que el llibre sigui predecessor o paral·lel d'un llibre que s'hagi de llegir.
- **assign_to_month**: S'encarrega d'assignar un mes a tots aquells llibres que compleixin el predicat **to-read**. Tindrà en compte tant que el mes sigui posterior al assignat als seus predecessors, com que, en cas de que tingui llibres paral·lels, el mes de lectura de ambdues llibres sigui o el mateix mes, o un anterior o un posterior.

Per una banda, la representació de la funció **assign_to_month** en lògica de primer ordre es veuría de la següent forma:

$$\forall b \in books\ \forall b' \in books(\neg to\_read(\mathbf{b}) \wedge$$
$$(pred(b', b) \vee parallel(b', b) \vee parallel(b, b')) \wedge \neg read(b') \wedge \neg to\_read(b')) \rightarrow$$
$$\rightarrow to\_read(b')$$

Tal i com es pot comprovar, l'únic canvi en aquesta acció ha sigut un **or** i la comprovació dels llibres paral·lels a b al lloc on en els nivells anteriors només es comprovaven els llibres predecessors a b. Així, ens assegurarem que els llibres paral·lels a aquells que estiguin classificats per llegir, també siguin considerats en el plan de lectura.

Per una altre banda, des d'un punt de vista lògic,la estructura de la acció **assign_to_month** és la següent:

$$\forall b \in books \;\; \forall month \in months : (\neg read(b) \wedge to\_read(b) \wedge$$

$$\wedge [\forall b' \in predecessors(pred(b', b) \wedge read(b') \wedge \tag{1}$$

$$(\neg to\_read(b')) \tag{2}$$

$$\vee \tag{3}$$

$$(\exists month' \in months(assigned(b', month) \wedge (month > month'))) \;) \;\;] \tag{4}$$

$$\tag{5}$$

$$\wedge [\forall b'' \in parallels((parallel(b'', b) \vee parallel(b, b'')) \wedge \tag{6}$$

$$[\; read(b'') \wedge \tag{7}$$

$$([\neg to\_read(b'')] \tag{8}$$

$$\vee \tag{9}$$

$$[\exists month'' \in months(assigned(b'', month'') \wedge \tag{10}$$

$$((month = month'' + 1) \vee (month = month'' - 1) \vee (month = month'')) \;)] \;)]$$

$$\tag{11}$$

$$\vee \tag{12}$$

$$[\neg read(b'') \wedge to\_read(b'')] \;) \;\;] \rightarrow \tag{13}$$

$$\rightarrow assigned(a, month_a) \wedge read(a) \tag{14}$$

Resumint, aquesta funció es divideix en dos blocs principals: l'encarregat d'assegurar-se que el mes sigui correcte respecte als predecessors de **b**, i el que té en compte els paral·lels per la mateixa tasca. El cas dels predecessors és el mateix que en els apartats previs: un **forall** s'encarrega de comprovar que en cas de tenir predecessors i que aquests s'hagin assignat a un mes, el mes d'assignació de **b** sigui posterior al dels seus predecessors. En cas que el predecessor no estigui claassificat com *to-read*, voldria dir que el llibre ja estava llegit al inici del problema, significant que no tindrà cap mes escollit i que no influirà.

D'altra banda, la part de la funció que comprova els paral·lels té el següent funcionament. Tot comença amb un **forall** que comprovarà, primer de tot, que tots els llibres siguin o bé **parallel(**b''**,** b**)** o bé **parallel(**b**,** b''**)**. En cas de no ser-ho, no es tindran en compte a l'hora de determinar el mes (com és lògic). En canvi, en cas de ser paral·lels, aquesta part de la funció es dividirà en dos parts més petites, separades per un **or**. La primera part del **or** servirà pels casos en els que el llibre b'' estigui llegit, i la segona pels que no. Aquesta distinció no és necessària en el cas dels llibres predecessors, degut a la natura de la relació. És a dir, en el cas dels predecessors si comencem a pujar de predecessor en predecessor, sempre acabarem en un llibre sense més predeecssors. Aquest, al no tenir cap dependencia de predecessors podrà ser assignat a qualsevol mes, i després, en funció del més assginat a aquest llibre, s'assignaran la resta. En canvi, en el que la relació de paral·lelisme respecta, aquest cas no es donarà (degut a que la relació és commutativa). Així doncs, podrà haver'hi casos en els que dos llibres paral·lels estiguin classificats per llegir però no estiguin ni assignats, ni llegits.

Si no fessim el or del que estavem parlant, el planificador mai aconseguiría assignarli un mes a cap dels llibres, ja que dependería del mes del altre, i cap tindrà cap mes assignat. Així doncs, la nostra decisió considera també aquest cas, dividint entre els llibres paral·lels a b que ja estiguin assignats (línies de l'equació 7-11) i una pels paral·lels que no ho estiguin (línia 13).

La lògica utilitzada pel cas dels paral·lels sense assignar és tan senzilla como comprovar que ho siguin i no fer res més, ja que al no tenir mes assignat no tindran cap influencia en la assignació de b. Per detectar-ho només mirem que b'' no estigui llegit però si per llegir. Per l'altre banda, en els casos dels paral·lels ja llegits, comprovarem que estiguin llegits i arribarem a un altre **or** que distinguirà entre els llibres assignats al inici del problema (aquells llegits però no per llegir), els quals no es tindràn en compte a l'hora de determinar el mes, i els que sí que estiguin assignats per llegir. Finalment, aquests seràn els que determinaran el mes de b. Així doncs, buscarem un més al que b'' estigui assignat, i obligarem al més que el planificador estigui buscant per b a ser o el mateix mes assignat a b'', o un anterior (restar-li un) o un posterior (sumar-li un, aprofitant que mes es un **fluent**).

2.4 Extensió 3

La tercera i última extensió proposada per l'enunciat demana la implementació d'una nova variable: el nombre de pàgines. Cada llibre tindrà un nombre de pàgines específic, i l'objectiu del planificador ara també serà acumular no més de 800 pàgines en total en cada mes. Així, s'intentarà repartir una mica la quantitat de lectura per mes en el plan.

2.4.1 Classes

Pel que a les classes respecta, el domini que proposem té les mateixes classes que aquelles descrites en l'apartat previ, però se li afegirà, com era d'esperar, alguns fluents que explicarem més endavant:

- **book**: Tindrà el mateix funcionament que en els nivell previs.
- **month**: Tindrà el mateix funcionament que en els nivell previs.

2.4.2 Predicats

Pel que als predicats respecta, no haurà cap canvi en comparació amb el nivell previ:

- **read** ***?book***
- **to-read** ***?book***
- **assigned** ***?book ?month***
- **predecessor** ***?pred ?book***
- **parallel** ***?par ?book***

2.4.3 Fluents

En aquest apartat, sí que hi haurà més modificacions, ja que haurem de modelar quantes pàgines té un llibre, quantes pàgines s'han "ocupat" en un mes i quin és el màxim de pàgines. Per tant, ens queden els següents fluents:

- **number_month** ***?month***: Nombre assignat al mes
- **month_pages** ***?book***: Nombre de pàgines que ja s'han llegit en aquell mes
- **pages** ***?book ?month***: Nombre de pàgines que té un llibre.
- **maxpages**: Fluent sense paràmetres que tan sols serveix per poder modificar el nombre màxim de pàgines que es poden llegir en un mes (ja que consideràvem que aquest valor havia de ser modificable perquè no tothom serà capaç de llegir el mateix nombre de pàgines en el mateix temps, i a més permet un intent d'optimització que explicarem més endavant).

2.4.4 Accions

Finalment, una de les dues accions veurà un petit canvi. Les dues principals seguiran sent les mateixes:

- **assign_to_read**
- **assign_to_month**

Però, assign_to_month haurà de considerar el nombre de pàgines del mes i del llibre que hi vulgui assignar. Així doncs, la nostra implementació consisteix en una comprovació de que el llibre b no tingui massa pàgines pel mes al que es vulgui afegir de la següent manera: Un cop comprovat que el llibre b no estigui llegit i que estigui classificat per llegir, es comprovarà que la suma de pàgines del mes (**month pages ?*month***) i les pàgines de b (**pages ?*book***) no sigui major que el nombre màxim de *pages*. Així, assegurarem que el planificador mai superi el límit proposat per l'enunciat. Finalment, en l'efecte de l'acció, s'afegirà el nombre de pàgines del llibre escollit al nombre de pàgines del mes escollit. A més d'aquest parell de modificacions, no cal afegir res més, ja que així ja cobrim tot el que se'ns demana a l'enunciat. A més, afegir la comprovació que el la suma de pàgines del llibre i del mes no superi les 800, permetrà al planificador estalviar-se un parell de cerques que impliquen utilitzar **forall** i predicats lògics complicats.

Tot i que el domini tal i com està programat funcionarà bé, això no implica que no tingui les seves limitacions. Per una banda, en cas que els llibres no sumin més de 800 pàgines, el més probable es que concentri molts llibres en uns pocs mesos, no equilibrant bé la quantitat de pàgines per mes. A més, si el número de pàgines totals supera un llindar proper al les 9600 pàgines, serà impossible trobar un plan amb menys de 800 pàgines per mes, fent que el planificador digui que no n'hi ha una resposta possible. Cal remarcar que pot existir una possibilitat d'optimitzar que el nombre de pàgines per cada mes sigui semblant als dels altres mesos.

2.5 Optimització de fluents i balanceig de pàgines

Com hem mencionat anteriorment, per al quart nivell hem intentat aplicar una optimització dels fluents a partir de *(metric minimize)* que ens permetés balancejar el nombre de pàgines a llegir cada mes, tal i com es demana -en certa manera- a l'enunciat.

Hi ha diverses maneres d'enfocar aquest problema, cadascuna de les quals implica els seus reptes i, últimament, la impossibilitat d'aplicar-los sense tenir més temps i eines.

La primera, i més bàsica, és fer un càlcul de la desviació de les pàgines llegides en cada mes respecte a la mitjana. Dins d'aquesta opció, es poden fer servir diverses implementacions, deixant el càlcul al domini, o pre-calculant la mitjana de pàgines i donant-la com a informació en el fitxer del problema, entre altres. Totes aquestes es troben inevitablement amb les limitacions del programa que fem servir, *metricff*, que no només ens limita a operacions molt bàsiques (suma, resta, multiplicació, i divisió, de manera que no podem calcular la desviació estàndard sinó que hem de fer servir la desviació estàndard al quadrat), sinó que no és capaç de realitzar operacions no "lineals" (és a dir, que afectin el valor de fluents que es poden modificar en altres accions).

Una altra opció, és apropar-nos al problema amb la següent filosofia: en certa manera, 800 pàgines al mes -o el nombre que sigui- ja és una manera de balancejar les pàgines llegides cada més, ja que qualsevol límit tindrà aquest efecte. Si afegim una acció per a modificar el nombre màxim de pàgines, i busquem la seva optimització com a paràmetre, el planificador buscarà la solució en la qual s'hagi aplicat més vegades l'acció de reduir el nombre de pàgines, que ens hauria de donar una aproximació bastant bona a la millor manera de balancejar les pàgines. També podem aplicar la solució inversa, començar per un nombre de pàgines més baix que el nombre mínim requerit pels llibres, i utilitzar una funció per augmentar pàgines, sense necessitat d'optimitzar la fluent, de manera que -en teoria- el programa ens retornaria la primera configuració de pàgines per mes en la qual es pugui trobar una solució vàlida.

Fet i fet, totes les implementacions que hem intentat han resultat ser impossibles d'implementar a nivell tècnic, o impossibles d'executar en un temps raonable. Per aquests motius, tot i que en cert punt teníem implementada aquesta opció, ens limitem a explicar-la de manera teòrica, i no l'hem conservat en els fitxers de codi.

2.6 Dominis seqüencials

Com hem comentat abans, vam realitzar també una versió seqüencial del problema. A continuació, esmentarem les seves principals diferències.

2.6.1 Classes

En aquest apartat no trobem cap tipus de diferència entre aquesta versió i la no sqüencial.

- **book**: Tindrà el mateix funcionament que en els nivell previs.
- **month**: Tindrà el mateix funcionament que en els nivell previs.

2.6.2 Predicats

Els predicats també són pràcticametn idèntics als de les altres versions, amb un sol canvi: no tenim predicat assigned month num (és un fluent).

2.6.3 Fluents

En aquest apartat ja hi ha més canvis. Trobem els fluents següents (extensió 3):

- **number_month** ***?month***: Fluent descrit abans
- **month_pages** ***?book***: Guarda el nombre de pàgines que ja s'han llegit en aquell mes
- **pages** ***?book ?month***: Guarda el nombre de pàgines que té un llibre.
- **maxpages**: Fluent sense paràmetres que tan sols serveix per poder modificar el nombre màxim de pàgines que es poden llegir en un mes (ja que consideràvem que aquest valor havia de ser

modificable perquè no tothom serà capaç de llegir el mateix nombre de pàgines en el mateix temps).

- **assigned ?*book***: Guarda a quin nombre de mes està assignat aquell llibre.
- **monthnum**: Nombre de mes actual. Com que es seqüencial, comença a 0 i li anem sumant 1.

Com podem observar, hem canviat un predicat per un fluent (assigned) i hem creat un altre fluent nou que ens permet determinar en quin mes està actualment el planificador.

2.6.4 Accions

Les accions són:

- **assign_to_read**
- **assign_to_month**
- **change_month**

Observem com hi ha una acció nova. Aquesta és l'encarregada de canviar de mes. Com que el planificador busca minimitzar el nombre de passos, només la utilitzarà quan sigui estrictament necessària, o sigui que realment la única precondició que té és que el nombre de mes sigui més petit que 12 (per evitar passar-nos de l'any).

L'efecte d'aquesta acció consisteix en augmentar en 1 el valor de monthnum, el mes actual.

Les altres dues accions funcionen de manera molt similar, però per tal de realitzar la comprovació de si un llibre ha estat assignat a un mes no cal utilitzar l'exists, sinó que amb una simple comparació de igualtat, o de més petit entre assigned del llibre i monthnum ens cal.

Mencionar que d'entrada pot sobtar que en l'apartat de predecessors comprovi només si està assignat en un més diferent a l'actual. Cal recordar que aquesta versió és seqüencial, i per tant no podrà estar assignat a un mes posterior a l'actual ja que no tenim cap acció per "recular" de mes.

3 Problema

Un cop explicat el domini, amb les diferents variables, predicats, i accions segons el nivell demanat per l'enunciat, entendre les mecàniques dels nostres problemes és quelcom bastant senzill. Com a norma general, tots necessitaran els 12 mesos de l'any que utilitzarà el planificador i pertanyeran al tipus *month*, junt amb una sèrie de llibres que pertanyeran al tipus *book*. A més, depenent del nivell en el que ens trobem, els llibres podran obtenir els diferents atributs (predicats) explicats i definits en els seus respectius dominis. Si aquests atributs no son corresponents al seu domini, el resultat no serà vàlid, ja que els dominis son robustos només si se'ls donen dades o problemes que segueixin les restriccions dictades pels enunciats.

Finalment, l'objectiu del planificador serà que tots els llibres que estiguin assignats o classificats per llegir acabin llegits. Considerant que els dominis són, per si mateixos, lògicament robustos, amb obligar-los a "llegir" tots els llibres assignats per llegir, ja s'encarregaran els diferents dominis de que es compleixin les especificacions de normes dictades pels enunciats. Tot i això, cal remarcar que, com ja s'ha explicat, això només serà cert si les dades d'entrada estan adaptades a les restriccions del seu nivell corresponent.

3.1 Nivell bàsic

Donat que el nivell bàsic només permet un predecessor per cada llibre, els problemes d'aquest nivell s'hauran d'ajustar, fent que: $\forall b \in books[(\ \exists! b' \in books(pred(b, b'))) \vee (\nexists b' pred(b, b'))]$

3.1.1 Objectes

En aquest nivell, només caldran els dos següents objectes:

- *book*: Tots els llibres, tant predecessors com no predecessors pertanyeran a aquest tipus d'objecte.
- *month*: Només contindra els 12 mesos, del gener fins al desembre.

3.1.2 Estat inicial

En l'estat inicial es declara el següent conjunt de predicats:

- *monthnum*: A la versió seqüencial és obligatori, a la no seqüencial s'incorpora per temes de compatibilitat de fitxers. S'inicialitza a zero usant (= (monthnum) 0)
- *number_month*: S'inicialitzen tots els fluents de number_month amb els seus nombres corresponents.
- *predecessor*: Les relacions de predecessors entre llibres.
- *read*: Llibres que ja s'han llegit
- *to-read*: Llibres que es volen llegir

Inicialitzant tots els predicats de cada tipus esmentat anteriorment, ja tenim una representació del problema adequada que representa tot el que se'ns demanava inicialment.

3.1.3 Estat final

Evidentment, també hem de declarar quin és el nostre estat final. Inicialment vam pensar a posar els llibres que voliem llegir com a goal (i treure el predicat to-read), ja que és la forma més lògica de pensar-ho, però aquest fet provoca que no tinguem forma de controlar si s'ha llegit els predecessors. Per tant, finalment vam optar per utilitzar un forall juntament amb un imply, que itera per tots els llibres i si se'ls havia de llegir, comprova que estiguin llegits.

D'aquesta manera, el nostre domini tan sols ha de col·locar els predecessors com a to-read, i el goal ja comprovarà que estiguin llegits (no distingeix entre els de l'enunciat inicial per llegir i els que hem assignat nosaltres).

3.2 Extensió 1

En aquesta extensió no canvia res del fitxer del problema. Tots els predicats seran dels mateixos tipus, i com a molt hi haurà més relacions de predecessors (ja que aquesta és la diferència entre el nivell bàsic i l'1.

El goal continua sent el mateix.

3.3 Extensió 2

En aquesta segona extensió, sí que hi haurà un predicat nou a l'init:

- *parallel*: Determina les relacions de llibres paral·lels.

Comentar que, per com està fet el nostre codi, no cal determinar les relacions paral·leles a dues bandes: amb indicant que a és paral·lel de b ja entén també que b és paral·lel de a.

3.4 Extensió 3

En aquesta última extensió, hi ha tres predicats nous:

- *maxpages*: Fluent que determina el màxim de pàgines que es pot llegir en un mes
- *pages*: S'han de determinar quantes pàgines té cada llibre, assignant al fluent **pages ?book** un nombre
- *month_pages*: Fluent que controla el nombre de pàgines llegides per mes s'ha d'inicialitzar a zero

3.5 Problemes per al domini seqüencial

Per a que els problemes puguin ser executats per el domini seqüencial, cal inicialitzar -novament a 0- un altre predicat, *monthnum*, que indica en quin mes es troba actualment el planificador, el qual comença al Gener i acaba al Desembre, d'un en un, enlloc de poder assignar un llibre a qualsevol mes. En tots els dominis normals, hem implementat també aquest predicat tot i no fer-se servir, per tal de tenir compatibilitat entre versions.

4 Jocs de prova

En aquest apartat, expliquem dues parts separades del projecte, cadascuna amb la seva finalitat i implementació. Primerament, parlarem dels fitxers que utilitzem per a generar automàticament jocs de prova, útils posteriorment per a realitzar els experiments. D'altra banda, també hem creat per a cada nivell dos jocs de prova a mida, per a poder comprovar que tot el que es demana funcioni correctament.

4.1 El generador

Per a poder tenir una multitud de jocs de prova generats aleatòriament, i que siguin adequats per a totes les situacions que volem provar -és a dir, per a cada nivell del problema-, hem creat un generador en Python, en el qual podem definir tots els paràmetres necessaris.

Aquest té dos components principals, cadascun en un fitxer diferent. El primer, és el fitxer *test_generator.py*, el qual genera l'estructura del fitxer del problema en PDDL, i on es pot definir la configuració.

De fet, hi ha dues versions del generador. La primera, *test_generator.py*, és la primera que vam desenvolupar, amb més comentaris i que té la intenció d'utilitzar-se de manera més manual. La segona, *test_generator_v2.py*, no té comentaris, és més estricta en certs paràmetres per a assegurar-nos d'obtenir tests vàlids, i s'utilitza com a part d'un pipeline més complet, i no de manera manual.

4.1.1 Configuració

Per a poder generar més d'un joc de prova simultàniament, les variables que defineixen com serà el problema estan representades com a vectors, on cada element s'utilitzarà per al fitxer d'un problema diferent. Això ens permet també utilitzar expressions com *num_books = [10 + i for i in range(x)]*, especialment útil a l'hora de realitzar els experiments temporals, ja que ens permet crear ràpidament un munt de jocs de prova que augmenten de tamany progressivament. Així, tenim els següents vectors:

- level: Indica a quin nivell de la pràctica s'ha d'adaptar el test generat, sent 0 el bàsic i 3 la tercera expansió.
- num_books: Indica quants llibres ha de tenir el problema.
- predecessor_chance: Probabilitat per a cada llibre de que un altre llibre sigui predecessor seu (sempre que sigui possible), entre 0 i 1.
- parallel_chance: Probabilitat per a cada llibre de que un altre llibre sigui "paral·lel" a ell (sempre que sigui possible), entre 0 i 1.

I altres paràmetres, amb valors únics:

- domain: Nom del domini (ha de coincidir amb el que s'utilitza als fitxers de domini).
- random_seed: Llavor utilitzada per al generador aleatori.
- config_range: Si s'utilitza, es fa servir dins de les inicialitzacions dels vectors de configuració (*for i in range(config_range)*), de manera que indica el nombre de tests que es generaran. Si s'inicialitzen les llistes manualment, el que marca la quantitat és la llargada de la llista.
- sequential_program: *True* indica que el test s'ha generat per a utilitzar amb el domini seqüencial, és a dir, ha d'incloure la fluent *monthnum* en la declaració de l'*init*.

4.1.2 Graf de relacions

El segon fitxer, *graph_generator.py*, conté la classe *BookGraph*, que s'utilitza per a crear el graf dirigit de relacions entre els llibres.

Els seus paràmetres d'entrada són els següents:

- num_books: La quantitat de llibres -és a dir, nodes- que ha de tenir el graf.
- random_seed: Llavor utilitzada per al generador aleatori.
- predecessor_chance: Probabilitat per a cada llibre de que un altre llibre sigui predecessor seu (sempre que sigui possible), entre 0 i 1.
- parallel_chance: Probabilitat per a cada llibre de que un altre llibre sigui "paral·lel" a ell (sempre que sigui possible), entre 0 i 1.
- multi_par: Booleà que indica si podem definir llibres que siguin paral·lels a més d'un alhora o no, ja que aquest tipus de paral·lels provocaven un augment molt significatiu en els temps d'execució d'alguns casos.

Cal tenir en compte que *parallel_chance* s'aplica sobre el conjunt que queda un cop s'han aplicat la probabilitat de que tinguin una relació seqüencial, i per tant, un valor del 0.5 no correspondria a la meitat de les relacions possibles sinó a la meitat de les que quedin després dels predecessors.

La classe *BookGraph* té tres opcions per a afegir nodes i arestes:

- add_sequential_edge: Afegeix una aresta que descriu una relació seqüencial entre dos llibres (un és predecessor de l'altre).
- add_parallel_edge: Afegeix una aresta que descriu una relació paral·lela entre dos llibres (s'han de llegir simultàniament).
- add_independent_node: Afegeix un node sense cap aresta (el llibre no té ni predecessors ni llibres simultanis).

També compta amb una funció per a visualitzar el graf, pintant les fletxes que representen les arestes paral·leles de vermell discontinu, i les seqüencials de blau continu, útil durant el període de prova del generador, i per a visualitzar situacions concretes:

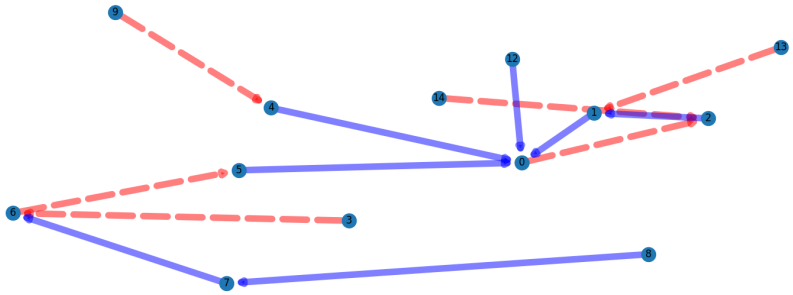


Figura 1: Graf de relacions per a 15 llibres. Els dos nodes independents, 10 i 11, queden fora de la imatge per a poder tenir una millor visualització.

La funció que genera el graf en sí és la funció *generate_graph*, que agafa com a input el nivell (0 a 3), i s'encarrega de generar les relacions que toquen en funció d'aquest i dels paràmetres de probabilitat.

El funcionament d'aquesta funció d'entrada sembla molt complexe, però simplement el que fa és iterar per totes les possibles arestes que permet aquell nivell i generar un nombre aleatori. Si aquest nombre està dins el percentatge indicat, intentarà crear la nova aresta d'entre les possibles, tot comprovant que no es generin cicles utilitzant la funció de la llibreria nx has_path. D'aquesta manera ens assegurem que el nostre graf és un DAG i que per tant, hi haurà una sol·lució pel problema.

Finalment, les funcions *get_sequential_edge_nodes* i *get_parallel_edge_nodes* ens retornen cadascuna una llista de tuples amb els llibres amb relacions seqüencials i paral·leles, respectivament. A més, *get_all_nodes* ens retorna tots els llibres del graf.

(A part, el generador de grafs compta amb funcions per a crear visualitzacions de cadascun dels jocs de prova fets a mà, més extensament explicat en futures seccions.)

4.1.3 Generació del fitxer PDDL

Així doncs, per a cada configuració que trobem en els vectors, es crea un nou fitxer (el nom del qual és *testX_Y.pddl*, on X és el nivell triat, i Y el número de fitxer generat en aquesta execució, començant per 0).

Per al nombre de llibres que haguem triat, es crea un objecte *bookX* (on X és el número del llibre).

Després, si el problema és de nivell 3, a cada llibre se li assigna una quantitat de pàgines aleatòries, com s'ha explicat anteriorment.

A continuació, utilitzant les funcions que ens proveeix la classe *BookGraph*, s'afegeixen totes les

relacions de seqüencialitat i paral·lelisme.

El següent pas, és designar els llibres que l'usuari ha llegit, i aquells que vol llegir. Per a aquesta tasca, partim d'unes quantes assumpcions bàsiques per a permetre el correcte funcionament del programa (és a dir, que el problema tingui solució):

- Condició: Tots els llibres, amb la seva quantitat de pàgines (extensió 3), han de poder ser llegits en els 12 mesos. Explicació: en cas contrari, es podria donar la situació on el planificador no es capaç d'assignar un mes a tots els llibres objectiu, si no dóna temps d'arribar fins a ells. Actuació: El número màxim de pàgines per a tots els llibres serà sempre inferior a 8000 (800 pàgines per mes, 12 mesos, 9600 pàgines), i per a un llibre individual tindrà com a molt 800 pàgines, per a assegurar-nos de que no passa la situació mencionada. Així doncs, el màxim de pàgines en el valor aleatori vindrà donat per la fórmula $maxim = min(8000/num\_llibres, 800)$, i el mínim per $minim = maxim//2$.

Es permet, tot i no tenir sentit a nivell real:

- Condició: L'usuari ha llegit un llibre X però no el seu predecessor X-1. Explicació: Se'ns demana que per a tot llibre del pla s'hagi llegit el seu successor, no necessàriament tots els anteriors. Si l'usuari vol llegir el llibre X+1, ja compleix la condició, i donat que X no està dins del pla de lectura (s'ha llegit prèviament) aquest no incompleix la regla tot i que X-1 no s'hagi llegit. Possible actuació: en generar els llibres llegits i a llegir, marcar tots els que siguin predecessors a un ja llegit o bé per llegir, o bé llegits.
- Condició: L'usuari haurà llegit un llibre paral·lel a un que tingui predecessors els quals no ha llegit. Explicació: això podria provocar que l'usuari no pugui llegir tots els llibres predecessors prou ràpid (extensió 3) per a poder llegir el paral·lel el primer mes de l'any, fet que incompliria la condició per als llibres paral·lels. Tot i això, novament, tan sols se'ns demana que siguin llegits els llibres paral·lels a aquells llibres *que estiguin en el pla de lectura*, cosa que si l'usuari els ha llegit prèviament, no passarà mai. Actuació: al assignar els llibres per llegir, es comprova si tenen paral·lels ja llegits. De ser així, es marquen com a ja llegits els seus predecessors, recursivament.

Tenint tot això en compte, s'escullen aleatòriament els llibres llegits i per llegir, i s'apliquen les restriccions prèvies. Un cop fet, acabem amb un graf semblant al que podem veure a la figura 2:

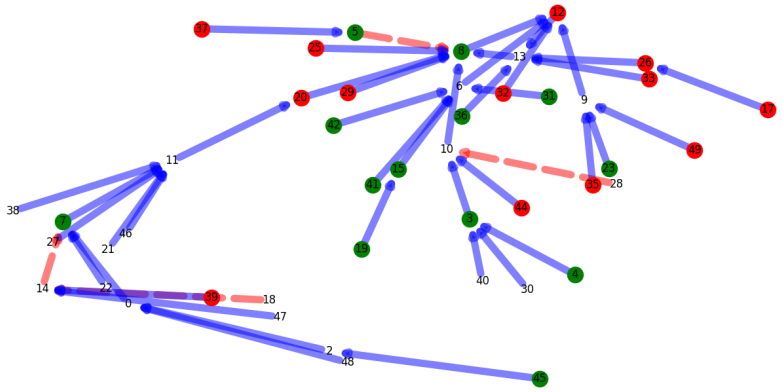


Figura 2: Graf de relacions per a 50 llibres, llavor aleatòria 42. Probabilitat de predecessor 0'5, probabilitat de paral·lel 0'3. Percentatge de llibres llegits 0'3, percentatge de llibres per llegir 0'3. Les fletxes blaves representen relacions de predecessors (el node origen és predecessor del node destí), mentre que les vermelles representen relacions paral·leles (sense ordre concret). Els nodes pintats de verd són aquells que l'usuari ja ha llegit abans de començar la planificació, els vermells aquells que vol llegir, i la resta no tenen color.

4.1.4 Ús del generador

Per a fer servir el generador, tan sols cal modificar les variables que es troben a la capçalera del fitxer *test_generator.py*, i executar el fitxer. Això ens crearà, en la mateixa variable on tinguem el fitxer, tots els jocs de prova d'acord amb els vectors de configuració creats. Obtindríem una sortida semblant a la següent imatge:

```
Test 0: 5 books, 0 sequential pairs, 0 parallel pairs, 1 read books, 1 books to read
Test 1: 6 books, 0 sequential pairs, 0 parallel pairs, 1 read books, 1 books to read
Test 2: 7 books, 0 sequential pairs, 0 parallel pairs, 2 read books, 2 books to read
Test 3: 8 books, 0 sequential pairs, 0 parallel pairs, 2 read books, 2 books to read
Test 4: 9 books, 0 sequential pairs, 0 parallel pairs, 2 read books, 2 books to read
Test 5: 10 books, 0 sequential pairs, 0 parallel pairs, 3 read books, 3 books to read
Test 6: 11 books, 0 sequential pairs, 0 parallel pairs, 3 read books, 3 books to read
Test 7: 12 books, 3 sequential pairs, 0 parallel pairs, 3 read books, 3 books to read
Test 8: 13 books, 4 sequential pairs, 0 parallel pairs, 3 read books, 3 books to read
Test 9: 14 books, 4 sequential pairs, 0 parallel pairs, 4 read books, 4 books to read
Test 10: 15 books, 7 sequential pairs, 0 parallel pairs, 4 read books, 4 books to read
Test 11: 16 books, 3 sequential pairs, 0 parallel pairs, 4 read books, 4 books to read
Test 12: 17 books, 7 sequential pairs, 0 parallel pairs, 5 read books, 5 books to read
Test 13: 18 books, 10 sequential pairs, 0 parallel pairs, 5 read books, 5 books to read
Test 14: 19 books, 10 sequential pairs, 0 parallel pairs, 5 read books, 5 books to read
Test 15: 20 books, 12 sequential pairs, 0 parallel pairs, 6 read books, 6 books to read
Test 16: 21 books, 16 sequential pairs, 0 parallel pairs, 6 read books, 6 books to read
Test 17: 22 books, 19 sequential pairs, 0 parallel pairs, 6 read books, 6 books to read
Test 18: 23 books, 21 sequential pairs, 0 parallel pairs, 6 read books, 6 books to read
Test 19: 24 books, 22 sequential pairs, 0 parallel pairs, 7 read books, 7 books to read
```

Figura 3

On se'ns indica per a cada joc de prova creat el nombre total de llibres, les arestes seqüencials i paral·leles, el nombre de llibres llegits, i el nombre de llibres per llegir.

Com s'ha mencionat anteriorment, si es volen utilitzar llistes per comprensió, es pot determinar la quantitat de fitxers generats amb la variable *config_range*.

4.1.5 Automatitzador

A més dels fitxers esmentats anteriorment, existeix també Automator.py. Aquest és un fitxer que permet, al mateix temps, utilitzar el generador i cridar el metricff, podent guardar així els resultats pel test si es vol, el temps d'execució...

En aquest fitxer està definida la classe BookGraphGenerator, que aprofita les funcionalitats del test_generator.py i la classe BookGraph de graph_generator.py per tal de crear un fitxer pddl donada una configuració. A més, incorpora el mètode run_metricff, que executa directament el domini creat. El resultat del planificador es guarda en una llista de tuples del tipus book,month en un fitxer results-nomdeltest.txt. D'aquesta manera, es poden accedir a les associacions finals creades pel planificador. Addicionalment, retorna el temps d'execució.

Usant aquesta classe i aquest últim mètode, en el mateix fitxer hi ha la funció sequence_of_experiments, que facilita el fet de realitzar diversos experiments. Rep com a paràmetre unes quantes llistes, executa els test corresponents i guarda els resultats en un csv (que es pot obrir després en excel i analitzar).

A l'apartat d'experiments explicarem més en profunditat com usar aquest fitxer.

4.2 Nivell bàsic

Havent vist el generador automàtic, considerem que és important tenir per a cada nivell jocs de prova fets manualment, que es puguin adaptar millor i on es vegi clarament la situació. Tots els jocs de proves es poden trobar a la carpeta *JocsDeProva*, amb el nom amb el següent format: JocDeProvaXY, on X indica el nivell (0 bàsic a 3, tercera extensió) i Y indica si és el primer test o el segon de cada nivell (1 o 2). Per a la informació de les relacions entre llibres, hem fet servir informació dels següents gràfics: The Published Works of Brandon Sanderson, Cosmere Connection/Reading Order Resource, ambdós de *Reddit*.

Per a fer això, imaginem un escenari on el nostre *usuari*, un àvid lector, ha completat de llegir l'última sèrie que tenia començada just a temps per aconseguir nous llibres durant el nadal, i demana als seus amics recomanacions per al pròxim any. Així, respon a la seva petició el seu estimat *amic*, un gran fan de l'escriptor Brandon Sanderson, la majoria dels llibres del qual formen part del seu *Cosmere* (l'univers on passen totes les històries).

Investigant una mica, l'*usuari* descobreix que el *Cosmere* conté dues sagues principals:

- Mistborn:
  - Els llibres principals, que es llegeixen en sèrie (però pertanyen a dues èpoques diferents, separades per una gran quantitat de temps):
    - *Mistborn: The Final Empire (1)*
    - *The Well of Ascension (1)*
    - *The Hero of Ages (1)*
    - *The Alloy of Law (2)*
    - *Shadows of Self (2)*

        * *The Bands of Mourning (2)*
        * *The Lost Metal (2)*
    - I els llibres secundaris:
        * *The Eleventh Metal*
        * *Allomancer Jak*
        * *Mistborn: Secret History*

- The Stormlight Archive:
    - Els llibres principals, que es llegeixen en sèrie:
        * *The Way of Kings*
        * *Words of Radiance*
        * *Oathbringer*
        * *Rhythm of War*
    - I els llibres secundaris:
        * *Dawnshard*
        * *Edgedancer*
        * *Horneater*

L'*usuari* recorda que l'*amic* ja li va recomanar fa temps que llegís la segona era de la saga de Mistborn, que comprèn des de *The Alloy of Law* fins a *The Lost Metal*. Tot i això, va abandonar després del primer llibre. Ara novament, l'*amic1* l'anima a continuar, ja que tot i que el primer llibre es fluixet, només millora a partir d'allà. També li diu que ja temps que *Mistborn: Secret History* és una lectura obligada, però que no oblidi que abans s'ha de llegir l'últim llibre de la primera era, *The Hero of Ages*, ja que sinó descobriria certes coses abans d'hora. Últimament, també l'ha sentit parlar amb molt d'entusiasme sobre el quart llibre de la sèrie *The Stormlight Archive*, cosa que li ha fet entrar ganes de llegir-se'l també, tantes que ja s'ha llegit el primer llibre, *The Way of Kings*.

Tenint tot això en compte, acabem amb el següent graf:

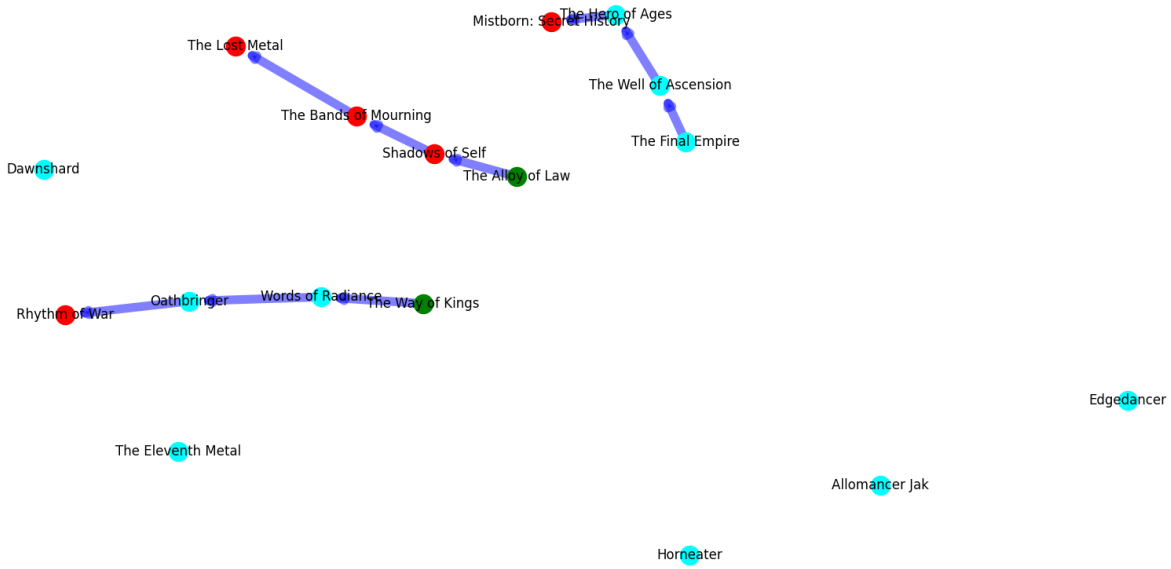


Figura 4: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 01.

Amb aquest joc de proves, busquem comprovar que els nostres dominis són capaços de tractar amb vàries cadenes de llibres simultàniament, així com amb llibres independents, tant si estan totes designades per a llegir, com si tan sols es vol llegir l'últim llibres d'aquestes, i tant si tenen algun llibre llegit inicialment com si no, i ens dona el següent resultat:

1. ASSIGN_TO_MONTH SHADOWSOFSELF OCTOBER
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING NOVEMBER
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_MONTH WORDSOFRADIANCE OCTOBER
7. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
8. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
9. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
10. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
11. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
12. ASSIGN_TO_MONTH THEFINALEMPIRE SEPTEMBER
13. ASSIGN_TO_MONTH THEWELLOFASCENSION OCTOBER
14. ASSIGN_TO_MONTH THEHEROOFAGES NOVEMBER

15. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY DECEMBER

16. REACH-GOAL

Amb el pla de lectura anterior preparat, el nostre *usuari* comença a buscar els llibres per llegir, quan es troba un fòrum on la gent comenta els seus llibres preferits, i es troba amb dos noms que li sonen, *Dawnshard* i *Horneater*. Així, decideix també llegir aquests dos llibres, que tenen relació respectivament amb *Oathbringer* i *Rhythm of War*. No només això, sinó que descobreix que l'autor té pensat continuar la sèrie de *The Stormlight Archive* amb sis llibres més, i per ser previsor introdueix el cinquè llibre, la continuació de *Rhythm of War* al sistema. Amb aquest canvi, obtenim el graf:

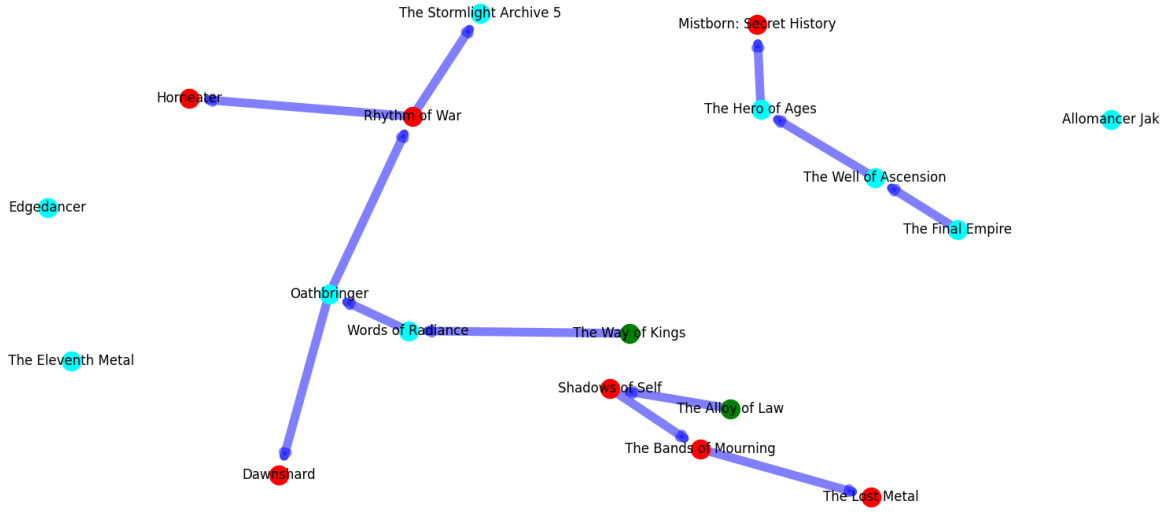


Figura 5: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 02.

L'objectiu d'aquest joc de proves es comprovar que tot continua funcionant bé quan un llibre és predecessor de més d'un llibre, situació que es pot donar quan una sèrie amplia els seus horitzons en més d'una direcció alhora.

Els resultats són els mateixos, cosa que confirma que el funcionament continua sent correcte.

4.3 Extensió 1

Anticipant-se a la lectura de tots els llibres, l'*usuari* cada vegada pren més interès en les diferents sagues, i l'intrincat univers on aquestes es troben. Així, descobreix que no només existeixen relacions entre els llibres d'una mateixa saga de diferents èpoques, sinó que al passar totes les sagues en un mateix univers, el *Cosmere*, hi ha llibres d'una saga que contenen relació a llibres d'una altra. També troba llibres d'una tercera saga, *Warbreaker*, que conté un llibre amb el mateix nom i un segon llibre que va després d'aquest, *Nightblood*.

Cercant informació, i parlant amb l'*amic*, descobreix que entre les sagues de *Mistborn*, la de *The Stormlight Archive*, i la de *Warbreaker*, i dins d'elles mateixes existeixen vàries relacions:

- Relacions entre sagues:
  - *Warbreaker* té connexions amb *Words of Radiance*
  - *Mistborn: Secret History* conté referències de *Oathbringer*
  - D'altra banda, *The Hero of Ages* aporta contribucions a *Oathbringer*
- Relacions intra sagues:
  - També cal llegir *Bands of Mourning* prèviament a *Mistborn: Secret History*

Que fa que obtinguem el següent graf:

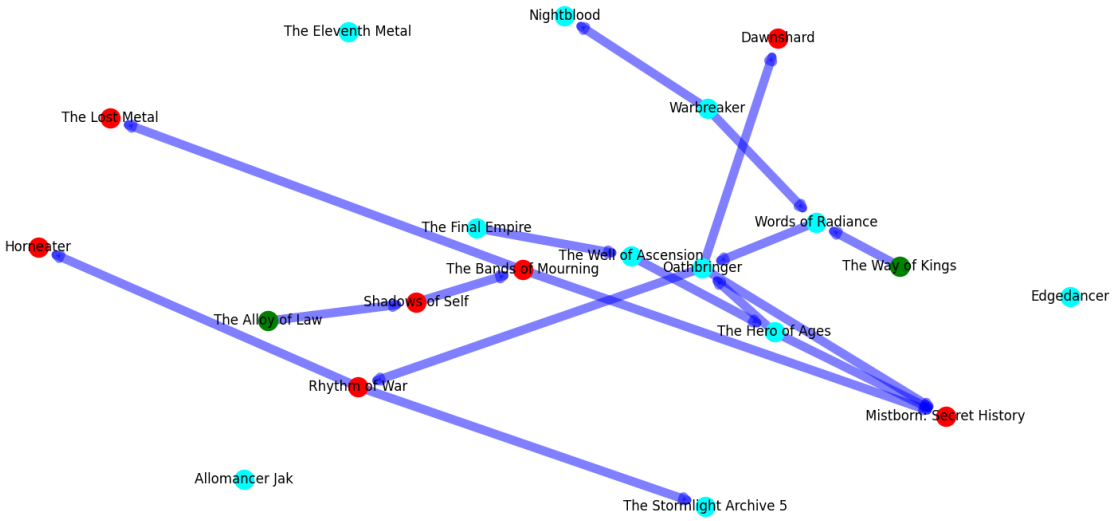


Figura 6: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 11.

Amb aquest joc de proves, busquem comprovar que efectivament un llibre amb més d'un predecessor no llegit, i marcat per a llegir, farà que s'assigni tots els predecessors en mesos anteriors.

Les assignacions són les següents:

1. ASSIGN_TO_MONTH SHADOWSOFSELF OCTOBER
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING NOVEMBER
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_READ WORDSOFRADIANCE WARBREAKER
7. ASSIGN_TO_MONTH WARBREAKER JANUARY
8. ASSIGN_TO_MONTH WORDSOFRADIANCE OCTOBER
9. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
10. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
11. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
12. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
13. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
14. ASSIGN_TO_MONTH THEHEROOFAGES MARCH
15. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
16. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
17. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY DECEMBER
18. REACH-GOAL

A mesura que investiga, també descobreix que els llibres que abans quedaven penjats, tenen tots el seu lloc en el *Cosmere*:

- *The Eleventh Metal* és un llibre que aporta informació molt important als fets que es llegeixen a *The Hero of Ages*
- De la mateixa manera, *Allomancer Jak* explica la història d'un important personatge en *Alloy of Law*
- *Edgedancer* fa d'interludi entre *Words of Radiance* i *Oathbringer*, una lectura gairebé obligada.

Novament, aquestes relacions ens compliquen el graf de llibres:

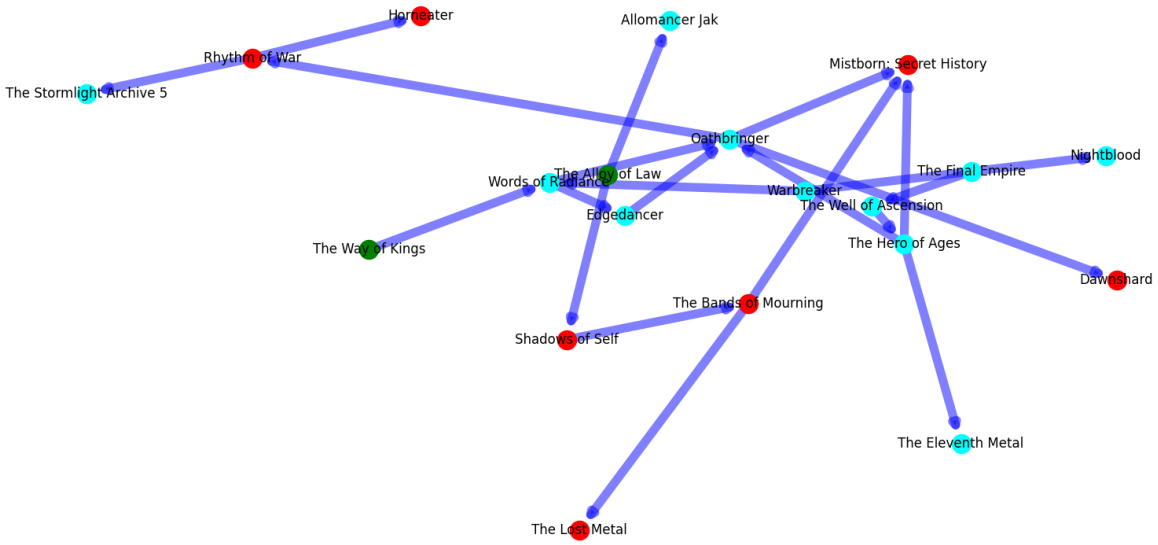


Figura 7: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 12.

Aquest joc de proves té l'objectiu de veure que el programa es continua comportant correctament un cop s'afegeixen múltiples "branques" (a partir de *Words of Radiance* es pot anar cap a *Oathbringer* o cap a *Edgedancer*, que també t'hi porta).

Amb aquests canvis, si que obtenim un resultat diferent, ja que es generen més dependències:

1. ASSIGN_TO_MONTH SHADOWSOFSELF OCTOBER
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING NOVEMBER
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_READ WORDSOFRADIANCE WARBREAKER
7. ASSIGN_TO_MONTH WARBREAKER AUGUST
8. ASSIGN_TO_MONTH WORDSOFRADIANCE SEPTEMBER
9. ASSIGN_TO_READ OATHBRINGER EDGEDANCER
10. ASSIGN_TO_MONTH EDGEDANCER OCTOBER
11. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
12. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
13. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE

14. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
15. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
16. ASSIGN_TO_MONTH THEHEROOFAGES MARCH
17. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
18. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
19. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY DECEMBER
20. REACH-GOAL

4.4 Extensió 2

Continuant amb la seva tasca d'investigació, l'*usuari* descobreix que les relacions prèvies no són tant senzilles com semblava, sinó que contenen elements que es poden descobrir en un ordre o en un altre, i per tant seria preferible llegir certs llibres alhora que altres enlloc de simplement assignar-los un ordre.

Així doncs, podem realitzar els següents canvis:

- Considerem que *Oathbringer* s'ha de llegir en paral·lel a *Mistborn: Secret History*, ja que així és més fàcil apreciar els petits detalls amb els que es connecten els dos llibres, fàcils d'oblidar si te'ls llegeixes amb massa temps de separació.
- Així mateix, podem considerar que *Warbreaker* té una funció més complementaria a *Words of Radiance* que no pas una de predecessor.

Amb aquestes modificacions, obtenim un graf molt similar a l'anterior, però amb les relacions paral·leles marcades en vermell.

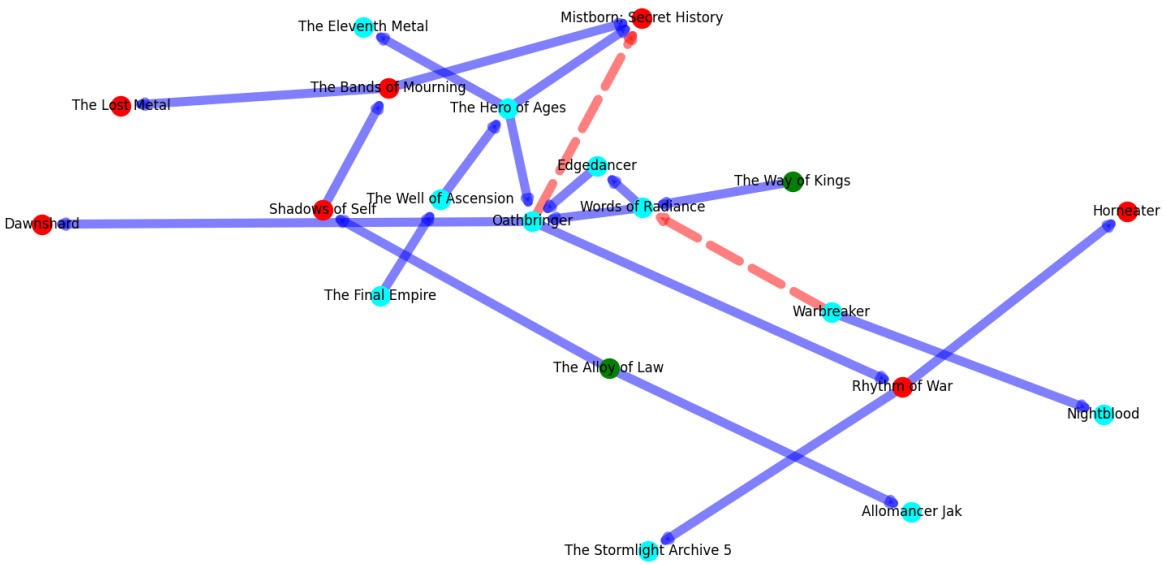


Figura 8: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 21.

L'objectiu d'aquest joc de proves es comprovar el correcte funcionament de les arestes paral·leles, per veure que el sistema és capaç de combinar els requisits dels llibres amb predecessors i aquells amb paral·lels.

Amb aquest graf, obtenim el següent pla:

1. ASSIGN_TO_MONTH SHADOWSOFSELF OCTOBER
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING NOVEMBER
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_READ WORDSOFRADIANCE WARBREAKER
7. ASSIGN_TO_MONTH WARBREAKER AUGUST
8. ASSIGN_TO_MONTH WORDSOFRADIANCE SEPTEMBER
9. ASSIGN_TO_READ OATHBRINGER EDGEDANCER
10. ASSIGN_TO_MONTH EDGEDANCER OCTOBER
11. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
12. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
13. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
14. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
15. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
16. ASSIGN_TO_MONTH THEHEROOFAGES MARCH
17. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
18. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
19. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY DECEMBER
20. REACH-GOAL

Després de moltes hores més de recerca, l'única informació addicional que pot trobar l'*usuari* és que també es pot considerar que *Mistborn: Secret History* és un llibre paral·lel i no pas posterior a *The Hero of Ages*, ja que els events més rellevants del llibre ja han passat en els dos primers llibres, *Mistborn: The Final Empire*, i *The Well of Ascension*, i la informació que conté és molt útil a l'hora d'entendre la trama de l'últim llibre de la trilogia.

Així doncs, podem canviar també la relació per una de paral·lela, obtenint el següent graf:

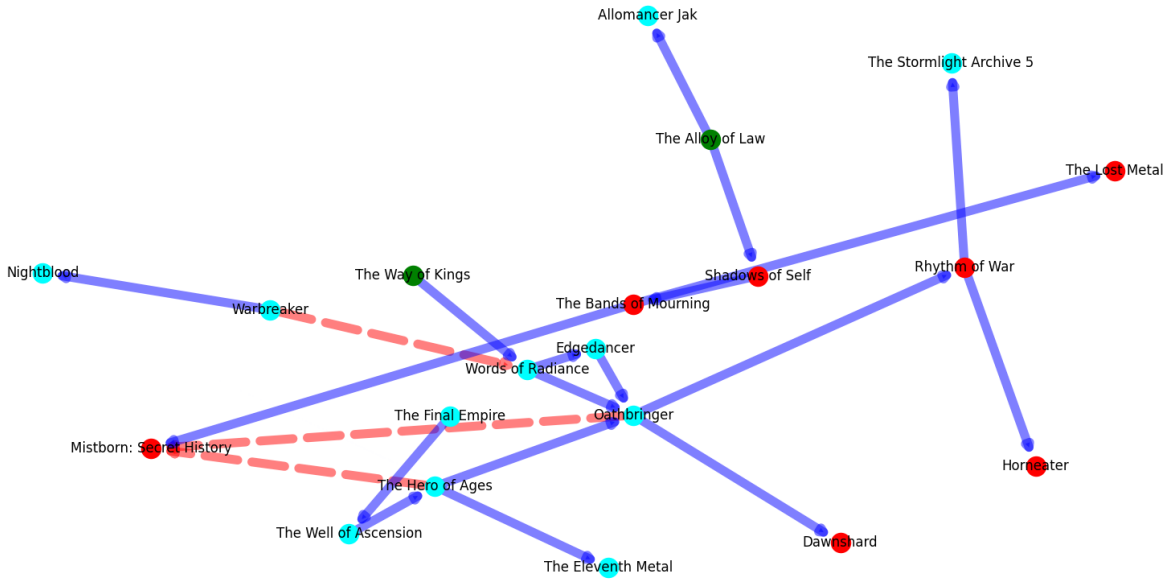


Figura 9: Graf que conté la situació de lectura de l'usuari per al *Cosmere*, nivel 22.

Finalment, amb aquest últim graf, podem comprovar que tot funcioni correctament quan entre els llibres que s'han de llegir per arribar a altres n'hi ha algun que té més d'un paral·lel, en aquest cas *Mistborn: Secret History*, que té dos llibres paral·lels.

Podem veure canvis en la col·locació dels paral·lels, ja que ara es troben limitats a estar en mesos pròxims als seus paral·lels:

1. ASSIGN_TO_MONTH SHADOWSOFSELF JANUARY
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING FEBRUARY
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_READ WORDSOFRADIANCE WARBREAKER
7. ASSIGN_TO_MONTH WORDSOFRADIANCE JANUARY
8. ASSIGN_TO_MONTH WARBREAKER JANUARY
9. ASSIGN_TO_READ OATHBRINGER EDGEDANCER
10. ASSIGN_TO_MONTH EDGEDANCER OCTOBER
11. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
12. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY OCTOBER
13. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER

14. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
15. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
16. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
17. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
18. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
19. ASSIGN_TO_MONTH THEHEROOFAGES OCTOBER
20. REACH-GOAL

4.5 Extensió 3

Havent arribat aquí, ens trobem amb el problema que el fet d'utilitzar les obres de *Brandon Sanderson* ens comporta per a l'enunciat de la pràctica: La majoria de llibres de la sèrie de *Mistborn* s'aproximen al límit mensual de 800 pàgines, mentres que cap dels llibres de la saga *The Stormlight Archive* baixa de les 1000 pàgines.

Això, evidentment, provocaria que el programa no trobés mai una solució, ja que assignant qualsevol dels llibres d'aquesta saga passariem el límit mensual de 800 pàgines, i fins i tot sense aquests, no tindríem gaire joc, ja que no hi hauria manera de llegir més d'un llibre en un sol més.

Degut a això, utilitzarem dues proves diferents: La primera, on els llibres tindran assignada la quantitat de pàgines real -excepte en aquells que passin de 800 pàgines, que en tindran 800-, i la segona on els llibres tindran tots una quantitat de pàgines aleatòria entre 200 i 400.

Així doncs, per al primer joc de prova, farem servir les següents quantitats:

- Mistborn: The Final Empire - 669
- The Well of Ascension - 640
- The Hero of Ages - 608
- The Alloy of Law - 352
- Shadows of Self - 400
- The Bands of Mourning - 480
- The Lost Metal - 528
- The Eleventh Metal - 672
- Allomancer Jak - 40
- Mistborn: Secret History - 240
- The Way of Kings - 800 (1007)
- Words of Radiance - 800 (1080)
- Oathbringer - 800 (1248)
- Rhythm of War - 800 (1232)
- The Stormlight Archive 5 - 800
- Dawnshard - 304
- Edgedancer - 272
- Horneater - 253
- Warbreaker - 592
- Nigthblood - 448

Si mirem la traça que trobem a continuació, podem veure que el planificador s'ha vist obligat a fer servir molts mesos que no tenien cap llibre assignat en versions prèvies, ja que els mesos que si que tenien assignacions no admeten més llibres:

1. ASSIGN_TO_MONTH SHADOWSOFSELF JANUARY
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING FEBRUARY
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ RHYTHMOFWAR OATHBRINGER
5. ASSIGN_TO_READ OATHBRINGER WORDSOFRADIANCE
6. ASSIGN_TO_READ WORDSOFRADIANCE WARBREAKER
7. ASSIGN_TO_MONTH WORDSOFRADIANCE JANUARY
8. ASSIGN_TO_MONTH WARBREAKER JANUARY
9. ASSIGN_TO_READ OATHBRINGER EDGEDANCER
10. ASSIGN_TO_MONTH EDGEDANCER OCTOBER
11. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
12. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY OCTOBER
13. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
14. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
15. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
16. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
17. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
18. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
19. ASSIGN_TO_MONTH THEHEROOFAGES OCTOBER
20. REACH-GOAL

Pel segon joc de prova, fem servir aquestes pàgines:

- Mistborn: The Final Empire - 296
- The Well of Ascension - 381
- The Hero of Ages - 321
- The Alloy of Law - 295
- Shadows of Self - 342
- The Bands of Mourning - 255
- The Lost Metal - 380
- The Eleventh Metal - 333
- Allomancer Jak - 238
- Mistborn: Secret History - 273
- The Way of Kings - 385 (345)
- Words of Radiance - 297 (366)
- Oathbringer - 272 (365)
- Rhythm of War - 381 (337)
- The Stormlight Archive 5 - 328
- Dawnshard - 264
- Edgedancer - 210
- Horneater - 394
- Warbreaker - 389
- Nigthblood - 272

Amb aquesta quantitat de pàgines, el planificador és capaç novament d'encabir molts més llibres en un mateix més de manera que tornem a veure tan sols uns quants mesos del principi i del final de l'any:

1. ASSIGN_TO_MONTH SHADOWSOFSELF JANUARY
2. ASSIGN_TO_MONTH THEBANDSOFMOURNING FEBRUARY
3. ASSIGN_TO_MONTH THELOSTMETAL DECEMBER
4. ASSIGN_TO_READ WARBREAKER WORDSOFRADIANCE
5. ASSIGN_TO_MONTH WARBREAKER SEPTEMBER
6. ASSIGN_TO_MONTH WORDSOFRADIANCE SEPTEMBER
7. ASSIGN_TO_MONTH NIGHTBLOOD NOVEMBER
8. ASSIGN_TO_READ MISTBORNSECRETHISTORY THEHEROOFAGES
9. ASSIGN_TO_READ THEHEROOFAGES THEWELLOFASCENSION
10. ASSIGN_TO_READ THEWELLOFASCENSION THEFINALEMPIRE
11. ASSIGN_TO_MONTH THEFINALEMPIRE JANUARY
12. ASSIGN_TO_MONTH THEWELLOFASCENSION FEBRUARY
13. ASSIGN_TO_MONTH THEHEROOFAGES AUGUST
14. ASSIGN_TO_READ MISTBORNSECRETHISTORY OATHBRINGER
15. ASSIGN_TO_MONTH MISTBORNSECRETHISTORY OCTOBER
16. ASSIGN_TO_READ OATHBRINGER EDGEDANCER
17. ASSIGN_TO_MONTH EDGEDANCER OCTOBER
18. ASSIGN_TO_MONTH OATHBRINGER NOVEMBER
19. ASSIGN_TO_MONTH RHYTHMOFWAR DECEMBER
20. REACH-GOAL

5 Experiments - impacte del tamany del problema

L'objectiu d'aquesta secció, es veure com es comporten a nivell temporal -per separat i entre ells- els nostres diferents dominis en funció de la quantitat de llibres amb la que han de tractar, per a poder fer-nos una idea de l'escalabilitat d'aquest tipus d'implementacions respecte a la mida del problema i a la seva complexitat.

5.1 L'experimentador

Per realitzar els experiments hem definit dos mètodes: o bé usant l'Automator.py esmentat anteriorment, o bé usant un pipeline per a realitzar execucions, recopilar el seu temps, i processar-lo per a tenir-lo en format CSV, que ens permeti utilitzar les dades. Tot això es realitza amb dos fitxers, un script de bash, *experiment_launcher.sh*, i un de Python, *process_results.py*.

5.1.1 Configuració de l'script de bash

Igual que amb el generador de jocs de prova, la idea per al experimentador és que pugui tractar amb una quantitat variable de combinacions de dominis i problemes. Al fer servir *bash* però, no podem fer servir les mateixes expressions que amb Python, o sigui que les llistes s'inicialitzen buides i després s'omplen amb un bucle, on hauríem de canviar les comprovacions en funció de la configuració del generador de tests. Així doncs, fa servir els següents vectors de configuració:

- domain_files: Indica el camí al fitxer de domini que s'ha d'utilitzar per a cada execució, Domini_basic o Domini_X on x indica el nivell de cada expansió, del 1 al 3.
- problem_files: Indica el camí als• fitxers de problema que s'executen en cada execució (cada iteració canvia el nombre del fitxer per tal que s'adapti al format donat pel generador de tests), i on s'indica també el nivell
- run_fluents: Indica si per a aquella execució concreta es fa servir com a paràmetre a optimitzar una fluent (s'ha d'afegir el paràmetre *-O* al executar el programa).

També tenim altres paràmetres de configuració:

- files_init: quantitat de fitxers generats per el *test_generator*, ha de ser equivalent a la variable *config_range*, si es fa servir, o a la llargada de les llistes si no.
- num_iterations: Nombre d'iteracions que es realitzen per a calcular el temps de cada execució.
- planner: Camí al executable de *metricff*.
- output_file: Camí al fitxer de sortida de les dades.

5.1.2 Funcionament i tractament de les dades de l'script de bash

Amb els paràmetres d'entrada personalitzats, l'experimentador ha de realitzar diferents passos, començant per donar informació sobre els paràmetres de l'execució. A continuació, per a cadascun dels fitxers que ha d'executar, realitza dues operacions. La primera és comprovar que el resultat de l'execució no és erroni, a través de la comanda

```
{if "$planner" -o "$domain\_file" -f "$problem_file" | grep -q "memory";}
```

que en cas de ser cert -és a dir, que el programa dona un error relacionat amb la memòria-, activaria la *flag out_of_memory*, tallant l'execució del programa.

En cas contrari, si l'execució es pot realitzar correctament, executa el nombre d'iteracions indicat, i per a cadascuna d'elles guarda fila a fila la sortida rellevant -el temps- al fitxer, amb la comanda

```
{ time "$planner" -o "$domain_file" -f "$problem_file"; } 2>&1 |

grep real | tr ',' '.' | tee -a $output_file}
```

La primera fila del fitxer conté en la primera fila la quantitat de fitxers i les iteracions per fitxer, per a facilitar la feina posterior.

A continuació, entra en acció el fitxer *process_results.py*, que ens permet fer el tractament d'aquest fitxer de dades, transformant les dades formatades (0m0.000s) a un vector de vectors, que contenen per a cada fitxer els resultats en segons, com a float.

El següent pas és per a cada fitxer, és a dir, per a cada vector, realitzar la mitjana dels valors.

Finalment, el programa converteix aquesta informació en dos fitxers en format CSV diferents, un que conté totes les dades, per a poder analitzar execucions concretes, i un fitxer que conté tan sols les mitjanes per a cada execució.

5.1.3 Configuració del script de python

L'alternativa al fitxer de bash per tal d'executar els experiments és l'automator.py. Aquest realitza alhora la tasca de crear el text, executar-lo, i guardar-ne els resultats.

Com que l'objectiu era usar-lo per tal de poder generar experiments, la aseva classe BookGraphGenerator rep molts paràmetres: des del nombre de llibres fins als percentatges o si es vol mostrar el graf de les relacions dels llibres.

Per generar el graf, com hem comentat abans, s'aprofita de la classe BookGraph. A més, amb el percentatge de llibres per llegir i llibres llegits n'assigna uns quants aleatòriament. Addicionalment trobem el mètode write_pddl_file, que com el seu nom indica, realitza precisament això: crear els arxius pddl. Comentar que aquí diferencia entre el si és seqüencial o no en alguns apartats, però realment no fa falta ja que finalment tots dos dominis estan adaptats per tal de funcionar amb el mateix tipus de fitxers.

Un altre mètode que té és el de guardar resultats. Com veure més endavant, aquest script té l'habilitat de llegir la sortida de l'execució i permet guardar en un fitxer results-nomdeltest.txt, les assignacions finals de cada llibre a cada mes, juntament amb el temps que ha tardat a trobar-les.

Finalment, el mètode probablement més interessant és el de run_metric_ff. Aquest utilitza la llibreria subprocess de python per tal d'executar el fitxer metricff.exe amb la configuració que li especifiquem. Provant amb diversos tests, vam adonar-nos que alguns tardaven molt més que la resta, fet que provocava que el temps d'experimentació agumentés molt. Degut a això, li vam assignar un timeout de 20 segons al procés, i si en menys de 20 segons no ha aconseguit acabar l'aturem i guardem com a temps 999.

En cas que sí que consegueixi acabar, mesurem el temps (temps inicial - temps final) d'execució real i decodifiquem la sortida.

5.1.4 Funcionament i tractament de dades de l'script de python

La forma d'utilitzar la classe definida en aquest problema és: crear l'objecte *BookGraphGenerator* amb tots els seus paràmetres, executar els mètodes *generate_book_graph()* i *write_pddl_file* i un cop fet això, cridar a *generator.run_metric_ff*. Com que executar manualment cadascuna de les combinacions pels experiments era inviable, vam decidir crear una funció que en bucle ens permet provar diverses configuracions de llibres, nivells i llavors. Aquesta mateixa funció també s'encarrega de guardar els resultats en un fitxer de nom *experiment-nomexperiment.csv*.

Per tal de processar aquests resultats, hi ha un segon arxiu de python dins la carpeta Experiments anomenat view_results que llegeix el csv que li indiquem i realitza el gràfic de barres per tal de visualitzar els resultats.

D'aquesta manera, podíem realitzar una gran quantitat d'experiments amb molt poca interacció humana, i limitats tan sols pel temps d'execució dels programes.

5.2 Els experiments - visió general

Així doncs, havent vist els fitxers que ens ajudaran a obtenir els resultats, podem començar els experiments. Com hem comentat en seccions anteriors, per a cada nivell hem realitzat dos dominis diferents, un que assigna els mesos arbitràriament i un que els assigna de forma seqüencial.

Els vectors i paràmetres de configuració que hem utilitzat per al primer experiment han sigut els següents:

Configuració:

- predecessor_chance = 0.4
- parallel_chance = 0.2
- read_books_percentage = 0.2
- books_to_read_percentage = 0.3
- seeds = [42,10, 249, 145]
- num_books = range(1,25)

Els vectors i paràmetres de configuració utilitzats no canviaran per als pròxims apartats a no ser que s'indiqui el contrari, de manera que només comentarem aquells que siguin nous o que hagin estat modificats, no repetirem tota la configuració.

Com hem comentat anteriorment, a partir dels resultats obtenim un fitxer amb totes les dades, que es pot trobar amb el format experiment-nom_experiment.csv

En el cas dels experiments realitzats amb el bash, els resultats es divideixen en dos fitxers diferents:

- Fitxer sencer: experiment_results_X_Y.csv
- Fitxer amb mitjanes: experiment_results_X_Y_average.csv

On X és el tipus de domini (bàsic, 1, 2, o 3) i Y és o bé "sequencial" en cas d'utilitzar el domini seqüencial, o bé no res en cas contrari.

Com veurem en l'experiment bàsic arribats a un cert nombre de llibres les execucions poden tardar molt de temps, de manera que hem decidit realitzar un primer experiment amb una menor quantitat de repeticions -5 repeticions per test-, i tan sols amb el programa normal, per a caracteritzar de manera general el comportament temporal.

Aquest experiment ha estat realitzat amb el fitxer de bash.

Així doncs, el resultat és el següent:

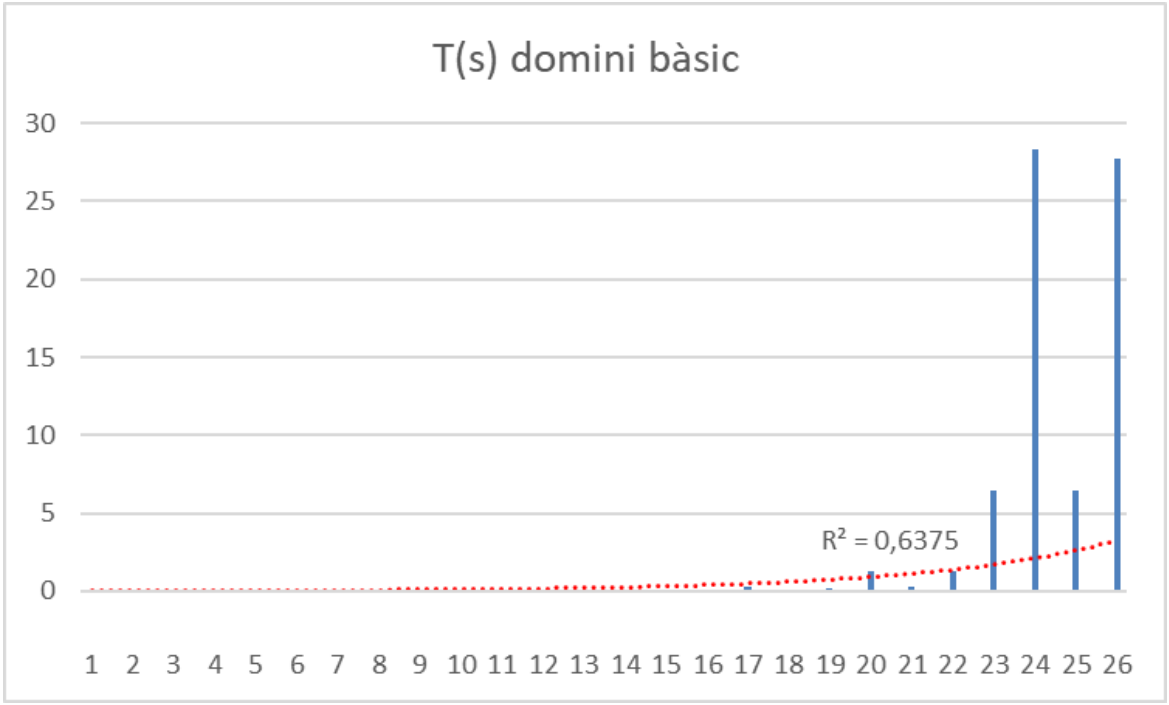


Figura 10: Gràfic de barres indicant el temps d'execució en segons de cada experiment. Cal tenir en compte que els números inferiors es refereixen al número de l'experiment i no a la quantitat de llibres. Es comença amb 10 llibres.

Com podem observar, fins que no arribem al voltant de 30 llibres, els temps d'execució són gairebé insignificants comparats amb els pròxims, sense arribar tan sols a un segon. En canvi, a partir de 34 llibres cap a endavant, el temps creix en gran mesura.

Canviant el funcionament del generador, aquest canvi es dona aproximadament als 25 llibres.

Així, podem veure vàries coses importants. La primera d'elles és que el temps creix de manera exponencial, tal i com indica la línia vermella. La segona, és que tot i aquesta tendència en el creixement, podem veure que entre tests consecutius, varia en gran mesura, i fins i tot pot baixar en un escenari amb més llibres. Això és degut a la naturalesa aleatòria de la generació dels tests, on un test amb un nombre major de llibres que un altre pot tenir "cadenes més senzilles", és a dir, presentar situacions que són més fàcils per al planificador de resoldre que un altre amb menys llibres.

Tot i això, podem arribar a la conclusió de que efectivament, el creixement és exponencial (és possible que poguéssim realitzar proves amb encara més llibres veiéssim un creixement més marcat encara), i que el problema no escala bé a partir d'un nombre de llibres moderat, cosa que concorda amb el que esperàvem, ja que ens trobem davant d'un problema similar al resolt en la pràctica anterior (on la complexitat temporal era semblant a aquesta) amb la diferència de que en aquest cas no som nosaltres els encarregats de triar l'heurístic ni d'escriure les funcions.

5.3 Nivell bàsic

Vist els resultats més generals, ara passem als concrets per al domini bàsic. Hem decidit experimentar amb un nombre de llibres des de 1 fins a 34 per tal d'observar com augmentaven els temps d'execució.

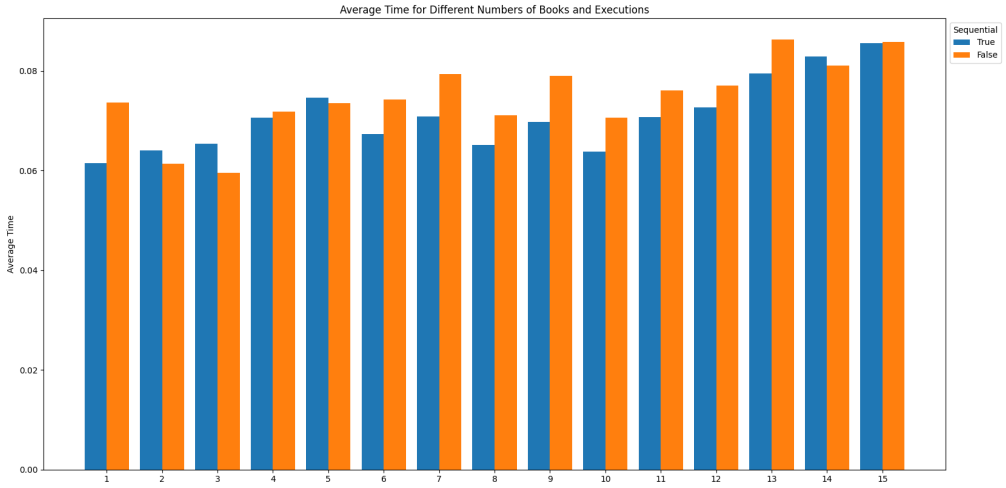
34 books, 19 sequential pairs, 0 parallel pairs, 10 read books, 10 books to read

El gràfics obtingut és el següent:

Figura 11: Gràfic de barres indicant el temps d'execució en segons de cada experiment, amb els programes normal i seqüencials.

En aquesta imatge, podem veure que obtenim uns resultats molt similars als de la imatge anterior. Aquesta conclusió era bastant òbvia, ja que al cap i a la fi, hem fet servir el mateix domini i els fitxers de problemes tampoc variaven tant.

Com ha passat amb l'anterior gràfic, costa molt distingir els primers experiments, ja que en els últims trobem un creixement tant gran respecte a la resta que és difícil conservar l'escala. Així, per a poder observar millor que passa en el principi, podem observar també un gràfic que conté tan sols els 15 primers registres:



Observem com en aquest cas, tots els temps d'execució són molt similars. De fet, no n'hi ha cap que passi de 0.08 segons, indicant que l'execució dels programes és pràcticament instantània. Realment no podem extreure moltes conclusions de la diferència entre el seqüencial i el no seqüencial, ja que els canvis poden ser deguts a factors d'atzar, però observant també el gràfic anterior sembla ser que les deus versions obtenen resultats pràcticament idèntics i com a molt, la seqüencial és més ràpida en alguns casos.

Aquests resultats coincideixen amb el que esperàvem, que el domini seqüencial fos capaç de com a mínim igualar la velocitat d'execució. Fins i tot és probable que en escenaris més complexos -amb paral·lels i més relacions- surti més a compte fer servir la versió seqüencial.

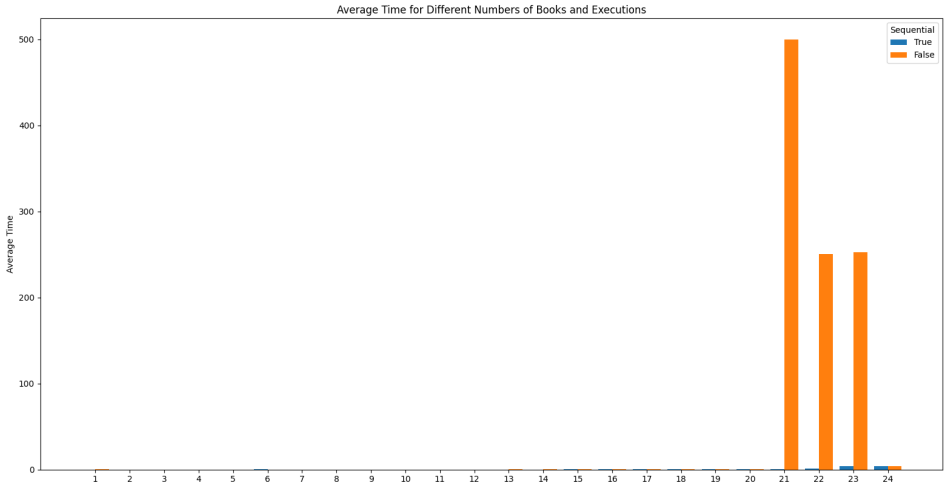
5.4 Extensió 1

Per a la primera extensió no hem requerit configuració addicional, ja que l'únic canvi entre el bàsic i l'1 es dona tan sols al generador del graf, de manera que només hem fet les següents modificacions:

- nivells = [1]

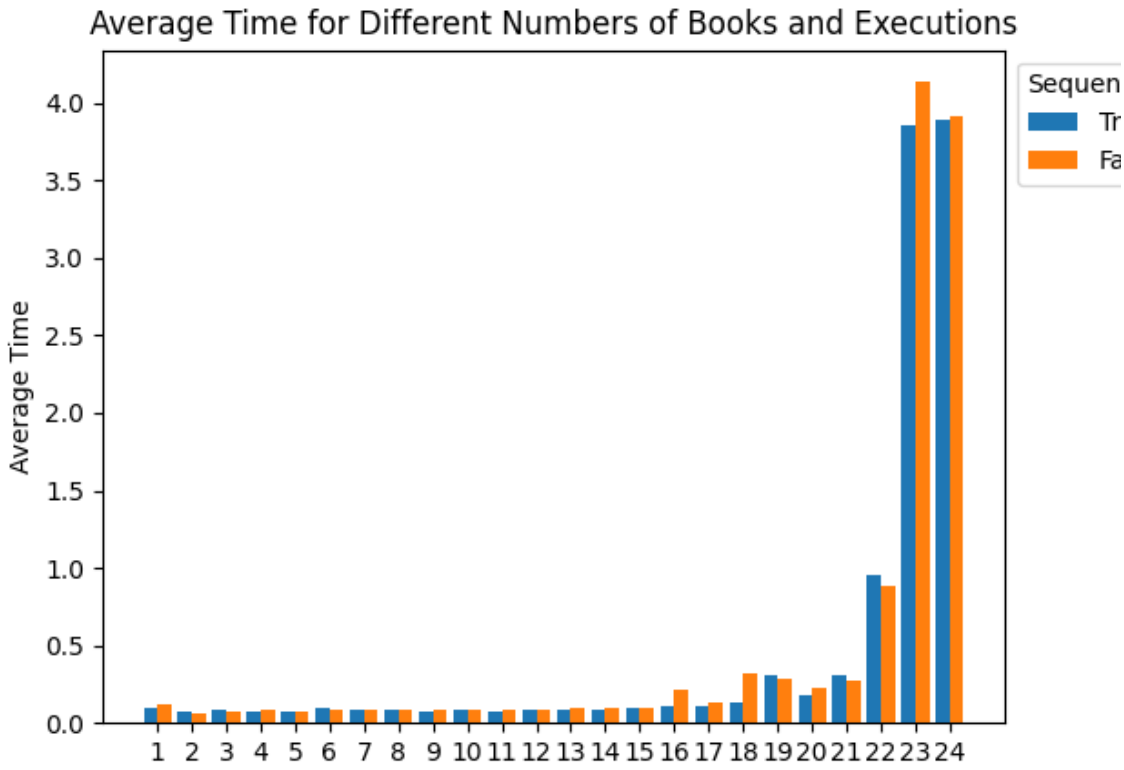
I la crida, igual que l'anterior, de sequence_of_experiments(name, books_num=books_list, levels = [nivells[e] for _ in range(len(books_list))], sequentials=[False,True], seeds = [42,10, 249, 145]).

Els resultats obtinguts són els següents:



D'entrada ja veiem que hi ha alguna cosa que no quadra en aquest gràfic. Recordem d'abans que havíem limitat el temps d'execució a 20 segons, i que quan els superava els hi assignàvem 999 segons.

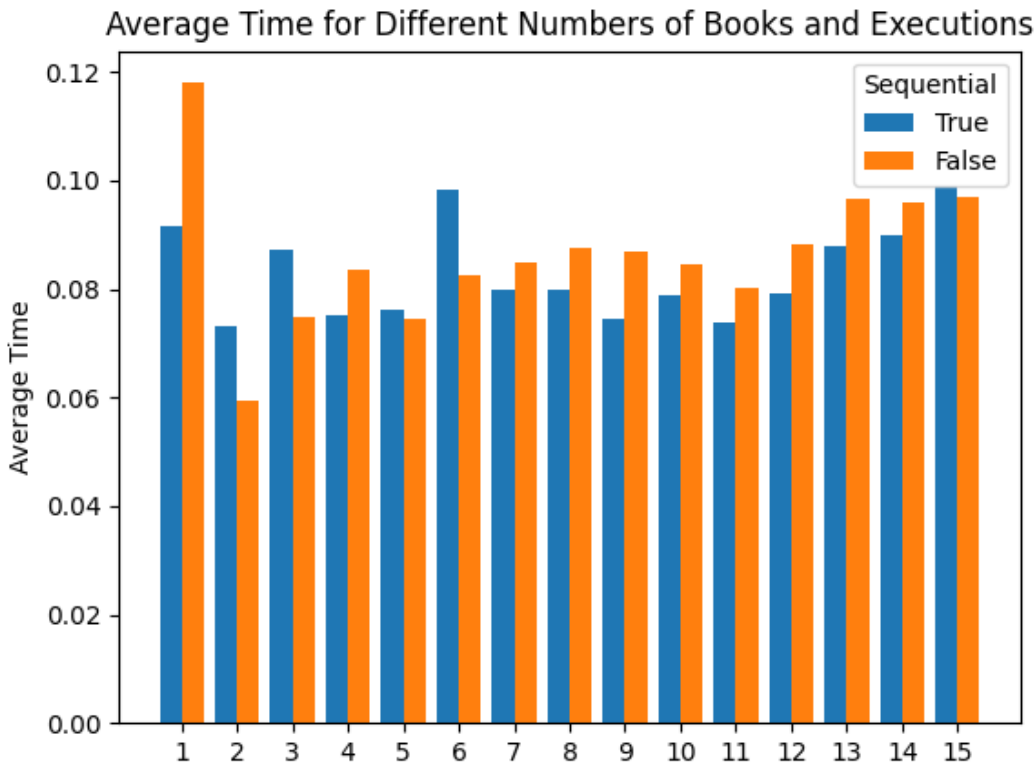
És per això que amb 21,22 i 23 llibres observem aquests pics. Si no tenim en compte aquests valors, el gràfic queda de la següent manera:



Aquests valors tenen molt més sentit i ens permeten extreure conclusions molt millors. D'entrada, observem el mateix creixement exponencial a partir de 21 llibres, quan els temps d'execució creixen moltíssim en comparació a la resta.

A grans trets, sembla ser que la versió seqüencial és millor que la no seqüencial altre cop, amb situacions com la que passa amb 18 llibres on el temps d'execució mig és de la meitat. A més, en el gràfic anterior no observàvem que aquesta versió passés dels 20 segons en cap situació.

Mirant més de prop els primers tests, veiem el següent:



A diferència del que havíem vist anteriorment, aquí trobem resultats més equilibrats, on a vegades va lleugerament més ràpid un domini o l'altre.

Comparant també amb el nivell bàsic, veiem com aquí els temps són lleugerament més lents comparats amb aquella versió. En aquest gràfic de fins a 15 llibres veiem com s'arriba a pràcticament 0.12 segons. Tot i això, amb tants pocs llibres és complicat extreure conclusions.

En conjunt, els resultats d'aquest experiment continuen amb la mateixa tendència, mantenint el temps a nivells molt baixos fins que arribat a cert punt, augmenta de manera exponencial, amb algunes inconsistències que, novament, creiem que són causades per la naturalesa aleatòria del generador de tests.

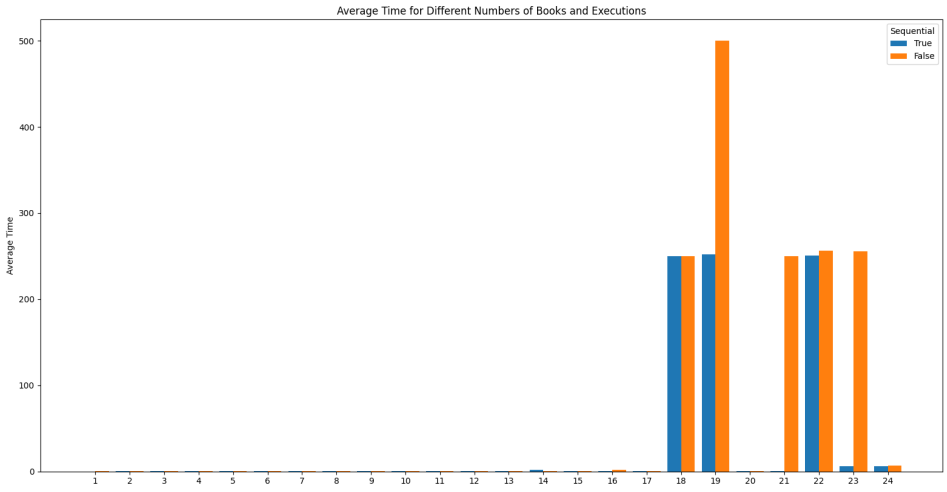
Comparat amb la secció anterior, ambdós domini seqüencial ha aconseguit mantenir el rendiment, de manera que ens cal passar als següents nivells per a treure més conclusions.

En general, però, un fet que ens ha sorprès és que els programes tarden pràcticament el mateix en l'extensió 1 que en l'extensió bàsica, indicant que l'augment en la complexitat dels predecessors no es tradueix en un augment tant gran en el temps d'execució del problema.

5.5 Extensió 2

Per a la segona extensió, hem seguit utilitzant els mateixos paràmetres. En aquesta, s'incorporen els llibres paral·lels, que tenen una probabilitat assignada de 0.2.

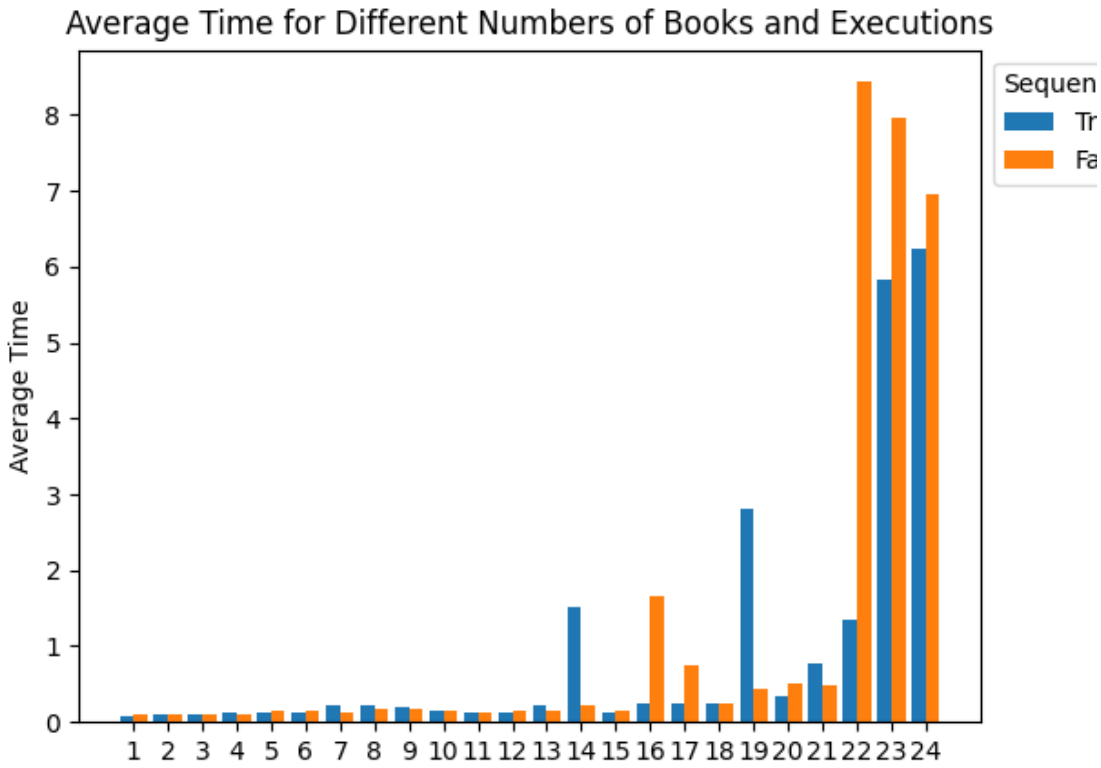
Els resultats són:



Podem observar com el fet d'incorporar paral·lels provoca moltes més situacions en les que el temps d'execució supera els 20 segons. A més, aquest tipus de situacions també apareixen en la versió seqüencial, que en l'apartat anterior no.

Observem també com el punt on apareixen situacions d'aquest tipus també és amb menys llibres, fet que ens pot indicar que els paral·lels afegeixen bastanta complexitat al problema.

Podem fer, igual que en els anteriors apartats, la gràfica sense tenir en compte els pics:



Aquesta gràfica ens permet comparar millor els temps d'execució amb l'apartat anterior. Observem com aquí sí que hi ha una diferència important, passant d'un màxim de 4 segons amb 24 llibres de l'apartat anterior al doble, 8 segons.

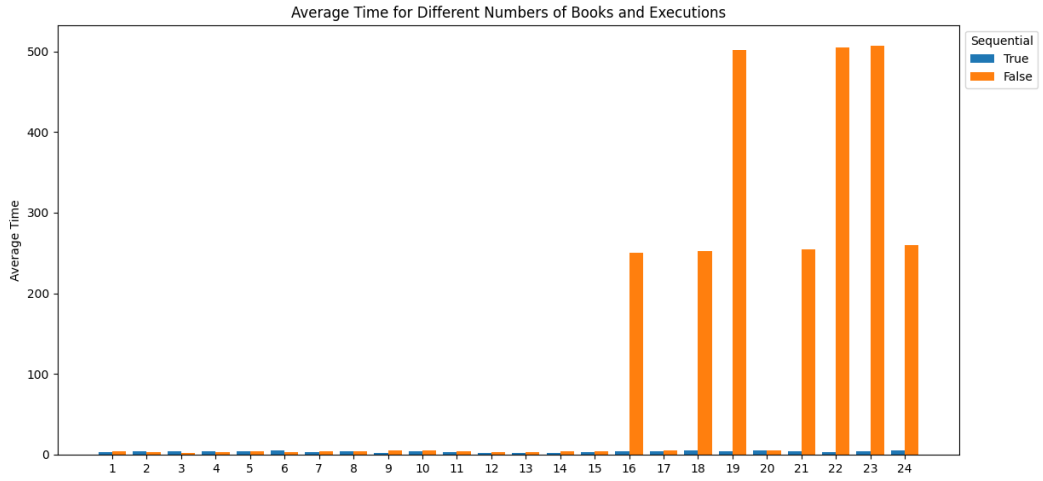
A més, en aquesta extensió, hi ha casos en els que la versió no seqüencial és molt millor a la hora de trobar un pla sol·lució. Això es deu a un cert factor d'aleatorietat, ja que pot ser que per casualitat el problema sigui capaç de trobar una sol·lució molt més ràpida del que esperaríem. A més, aquest fet també explica perquè, per exemple, amb 24 llibres no és significativament més lent que amb 23 (i de fet a la gràfica d'abans, amb 24 eren capaços els dos programes d'executar-ho en menys de 20 segons sempre i amb 23 no).

5.6 Extensió 3

Finalment, l'extensió 3 tan sols suposava afegir el nombre de pàgines màxim per mes i les pàgines de cada llibre. Això suposa una restricció addicional, però no és tan gran com la d'altres apartats.

A més, a la hora de generar el problema, ja ens asseguràvem d'anar en compte amb el nombre de pàgines per tal d'assegurar una sol·lució, fet que també ha reduït la complexitat del problema.

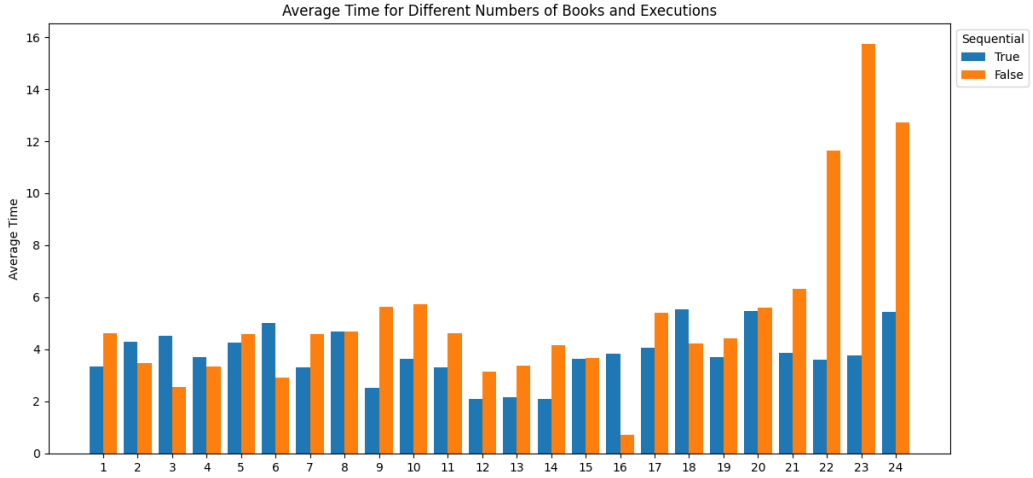
Els resultats són:



Observem altre cop els pics de les execucions que no han arribat a tardar menys de 20 segons. Tot i això, observem un comportament curiós en el cas del seqüencial: en aquest cas totes les execucions han tardat menys de 20 segons.

Cal recordar però, que hi ha un cert factor d'atzar a la hora de determinar els problemes que tarden més de 20 segons i que pot ser que amb algunes altres llavors sí que passi. Tot i això, sembla ser que aquesta versió té més resistència als problemes complicats que l'altra.

Altre cop, traient els pics:



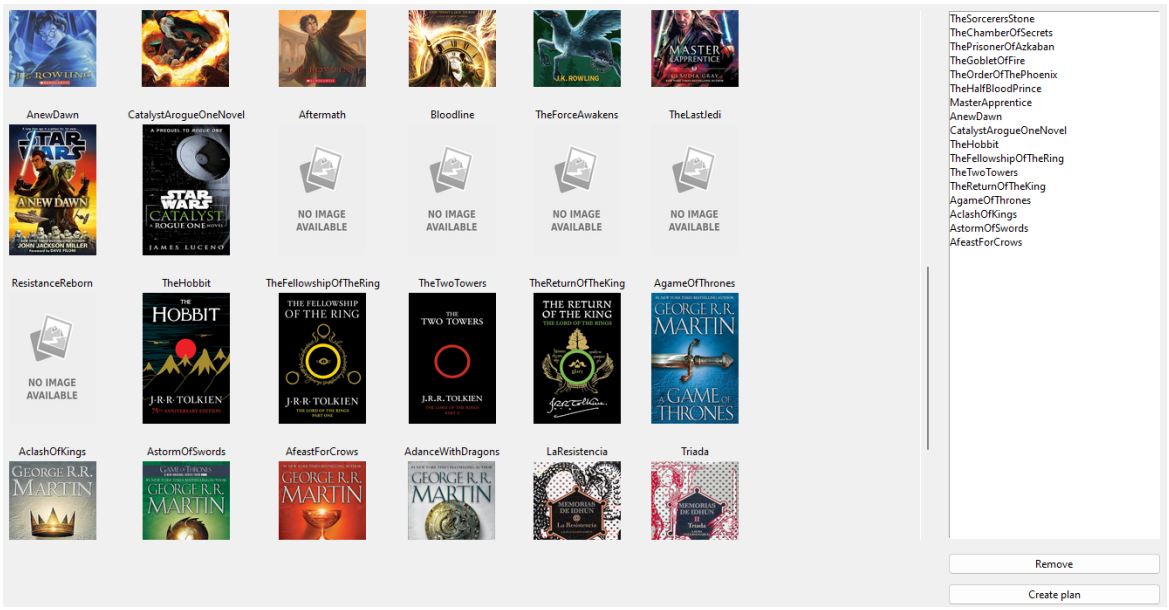
Els resultats d'aquesta extensió són inesperats. Amb menys llibres tarda molt més que les versions anteriors (4 segons, el que ens tardava en l'extensió 1 amb 24 llibres) però es manté molt més constant. De fet, la versió seqüencial tarda el mateix amb 24 llibres que amb 1, 2 o 3. L'altra versió sí que sembla ser que augmenta, però en aquest cas no de forma tant exagerada com en els anteriors.

Sembla ser doncs, que el fet d'afegir aquesta restricció ha provocat realment resultats més bons. Evidentment s'hauria de comprovar usant més casos, però per algun motiu aquesta restricció permet al planificador trobar de manera més ràpida la sol·lució.

6 Interfície gràfica

Després de realitzar tots els experiments i tenir una versió definitiva del programa, vam decidir anar un pas més enllà, ja que volem crear un producte que un usuari pogués realment utilitzar i que tingui sentit.

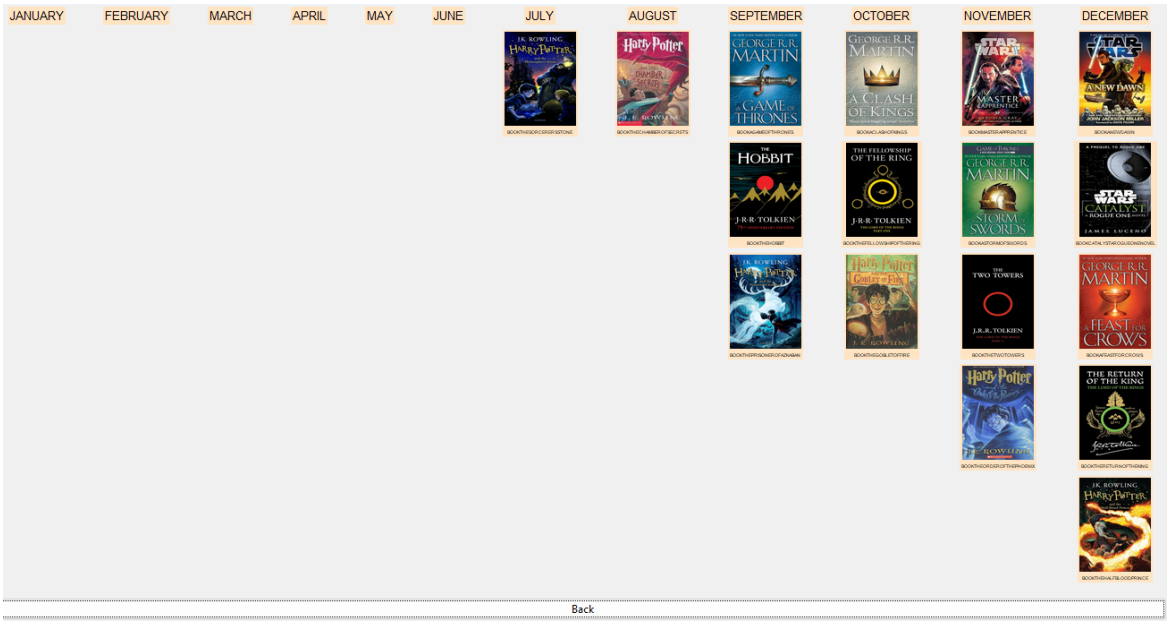
Per tal de fer això, hem creat un nou fitxer anomenat interface.py, situat dins de la carpeta Media, que si s'executa obre una interfície com la següent:



En aquesta trobem a l'esquerra un conjunt de llibres que l'usuari pot sel·leccionar i a la dreta una llista dels noms dels llibres sel·leccionats. Comentar que aquests llibres es llegeixen d'un dataset que hem creat nosaltres manualment, i que quan s'afegeix un llibre també s'afegeixen els necessaris per tal de poder-se'l llegir a la llista.

Més avall, trobem un botó per poder executar el programa. Aquest realment l'únic que fa es cridar Automator.py, on hem definit un mètode per tal de crear un problema donada una llista de llibres (els sel·leccionats), una llista de predecessors, una altra de paral·lels i una de les pàgines.

Seguidament, llegim la resposta obtinguda i mostrem per pantalla cada mes amb els llibres assignats a aquell.



Creiem que aquesta interfície gràfica, si bé es pot millorar molt i encara té alguns problemes,

permet entreveure com podria ser una aplicació real d'aquest problema i planificador en, per exemple, una llibreria o una empresa que tracti amb llibres. En general pensem que és una bona manera de visualitzar els resultats de l'experiment i que, al tractar amb llibres reals, és més fàcil imaginar-se una situació en la que pugui ser útil.

7 Conclusions

Després d'executar els diversos experiments, hem arribat a diverses conclusions. En general, hem vist com augmentar el nombre de llibres provoca un augment del temps d'execució exponencial.

Realitzant altres proves, també hem pogut comprovar com, si augmentem la probabilitat de que hi hagi predecessors i paral·lels, els temps també augmenten. En general, aquests predicats provoquen restriccions a la hora de crear el pla que provoquen que el planificador hagi de trobar una sol·lució més rebuscada, augmentant d'aquesta forma el temps que triga.

Addicionalment, i com que hem realitzat els experiments amb dos plantejaments diferents del problema, també hem pogut extreure conclusions del funcionament d'aquests. Segurament el millor dels dos és el model seqüencial, ja que sembla ser que és millor sobretot a mesura que augmenta la complexitat del problema, amb molts predecessors i paral·lels. Per aquest tipus de llibres, té més sentit realitzar-ho d'aquesta manera, ja que ens estalviem comprovacions i reduïm en gran mesura l'espai de cerca.

Un altre avantatge del seqüencial és que a nivell pràctic, si afegim més mesos no té un impacte molt important en el rendiment, mentre que en el no seqüencial sí ja que haurà de realitzar una cerca més gran, de manera que podríem adaptar el planificador a diferents períodes de temps sense tenir un canvi important en el rendiment.

Pel que fa a la velocitat, creiem que és possible que amb altres planificadors -i per tant amb altres implementacions- es pugui obtenir temps d'execució menors, però tot i això els dominis que hem desenvolupat podrien ser utiltizats en casos reals que fossin de tamany petit.

En general, hem vist dues maneres d'implementar un mateix problema, cadascuna amb els seus avantatges i desavantatges, i hem comprovat com a mesura que augmentava la mida també augmentava el temps, ja que -evidentment- que el planificador tingui més llibres disponibles provoca que hi hagi també més accions possibles. També podem veure que les restriccions de cada extensió també augmenten el temps de còmput ja que hi haurà algunes accions que no podrà fer i haurà de ser capaç en alguns casos de realitzar accions que d'entrada semblen innecessàries (canviar de mes, per exemple, realment no acosta a l'objectiu; o bé assignar llibre a per llegir) però que són clau per tal de poder assolir l'objectiu.