0. Taula de continguts

1. Introducció

2. Objectius i metodologia

3. Disseny del domini i dels problemes

3.0 Extensió bàsica

L'extensió bàsica del domini `hotelbasic` implementa la funcionalitat fonamental d'assignació d'habitacions a reserves, gestionant les restriccions de compatibilitat i no-solapament temporal. Aquesta versió inicial serveix com a base per a les extensions posteriors, establint els conceptes clau i la lògica de recursos que es desenvoluparan més endavant.

3.0.1 Domini

El domini `hotelbasic` modela un problema d'assignació de recursos (scheduling) on un conjunt de peticions (`reserva`) han de ser assignades a recursos limitats (`habitacio`) durant uns intervals de temps específics (`dia`).

1. Tipus (**`:types`**) El domini defineix tres entitats bàsiques que estructuren el problema:
    - **`reserva`**: Representa una petició d'allotjament que ha de ser satisfeta.
    - **`habitacio`**: Representa el recurs físic (amb capacitat unitària) on s'allotgen les reserves. -**`dia`**: Representa la unitat de temps discreta.
2. Predicats (**`:predicates`**) Els predicats defineixen l'estat del món i les relacions entre els objectes:
    - **Predicats Estàtics (Dades d'entrada):**
        - `(dies-reserva ?r - reserva ?d - dia)`: Defineix l'interval temporal de cada reserva. Indica que la reserva `?r` requereix ocupació durant el dia `?d`. Aquest predicat es defineix per a cada dia dins de l'interval de la reserva.
        - `(compatible ?r - reserva ?h - habitacio)`: Restricció de domini que indica si l'habitació `?h` és vàlida per a la reserva `?r` (per exemple, per capacitat de persones). Això ho hem fet perquè **les reserves només es puguin assignar a habitacions compatibles.**
    - **Predicats Dinàmics (Estat del sistema):**
        - `(assignada ?r - reserva)`: Indica que la reserva `?r` ja ha estat processada i té una habitació assignada.
        - `(ocupada ?h - habitacio ?d - dia ?r - reserva)`: Indica que l'habitació `?h` està ocupada pel menys per la reserva `?r` durant el dia `?d`. Aquest predicat s'actualitza dinàmicament a mesura que es processen les assignacions, ja que necessitem per garantir que no hi hagi solapaments.
3. Accions (**`:action`**) L'única acció del sistema en la seva implementació bàsica és **`assignar-habitacio`**, que formalitza la decisió d'ubicar una reserva.

- **Paràmetres:** Una reserva `?r` i una habitació `?h`.
- **Precondicions:** Per poder executar l'acció, s'han de complir tres condicions simultànies:
    1. `(not (assignada ?r))`: La reserva no ha d'estar ja assignada (evita duplicats).
    2. `(compatible ?r ?h)`: L'habitació ha de ser adequada per a la reserva.
    3. **Restricció de No-Solapament:**

```
(not (exists (?d - dia ?r2 - reserva)
     (and (dies-reserva ?r ?d) (ocupada ?h ?d ?r2))))
```

La traducció d'aquesta precondició és: *si no existeix cap dia `?d` i cap altra reserva `?r2` tal que `?r` demani el dia `?d` i l'habitació `?h` estigui ocupada per `?r2` en aquest dia*. Això verifica que, per a tots els dies que demana la reserva `?r`, l'habitació `?h` no estigui ocupada per cap altra reserva `?r2`. És el nucli de la lògica de recursos. És el que garanteix que no hi hagi solapaments en l'assignació d'habitacions, per això calia incloure aquest predicat dinàmicament actualitzat.

- **Efectes:** Si s'executa, l'estat canvia:

    1. `(assignada ?r)`: La reserva es marca com a completada.
    2. **Bloqueig de Recursos (Conditional Effect):**

    ```
    (forall (?d - dia) (when (dies-reserva ?r ?d) (ocupada ?h ?d ?r)))
    ```

    Per a cada dia `?d` que forma part de la reserva, es marca l'habitació `?h` com a ocupada. L'ús del `forall` és **fonamental**, perquè cal actualitzar l'estat per a tots els dies de la reserva. L'ús del condicional amb `when` permet actualitzar l'estat de manera eficient, bloquejant l'habitació només durant els dies pertinents.

D'aquesta manera, el domini bàsic estableix les regles fonamentals per a l'assignació d'habitacions a reserves, gestionant les restriccions de compatibilitat i no-solapament temporal. Aquest marc servirà com a base per a les extensions posteriors, on s'afegiran funcionalitats més avançades per millorar la flexibilitat i l'eficiència del sistema de planificació.

3.0.2 Problemes

3.0.2.1 Problema 1: Poques habitacions, moltes reserves

En aquest experiment volem avaluar la capacitat del planificador per prioritzar i seleccionar el millor subconjunt de reserves quan els recursos són extremadament limitats. Per fer-ho, mantindrem fix el nombre d'habitacions (`n=2`) i incrementarem progressivament el nombre de reserves candidates (5, 10, 15, 20...). Això força el sistema a gestionar un escenari d'alta competència on la gran majoria de reserves, o totes, s'han de descartar. S'espera observar:

1. **Comportament Intel·ligent:** El planificador haurà de triar les combinacions de reserves que maximitzin l'ocupació total (evitant forats temporals), en lloc d'agafar simplement les primeres de la llista.
2. **Escalabilitat:** S'espera un creixement no lineal (ràpid) del temps d'execució, ja que l'espai de cerca per trobar la combinació òptima creix combinatorialment a mesura que afegim més reserves solapades."

Per tant, plantegem el següent parell d'hipòtesis per a aquest experiment:

Respecte al comportament del planificador en aquest escenari d'alta competència:

- H_0: El planificador no és capaç de maximitzar l'ocupació total en situacions d'escassetat de recursos, seleccionant reserves de manera aleatòria o greedy.
- H_1: El planificador és capaç de maximitzar l'ocupació total, seleccionant reserves de manera intel·ligent per evitar forats temporals.

Hipòtesi sobre l'escalabilitat del planificador:

- H_0: El temps d'execució del planificador creix linealment amb el nombre de reserves, indicant una gestió eficient de l'espai de cerca.
- H_1: El temps d'execució del planificador creix de manera no lineal (ràpid) amb el nombre de reserves, indicant un augment combinatorial de l'espai de cerca.

Generem doncs diversos problemes amb 2 habitacions i un nombre creixent de reserves (1, 2, ... fins a 10) amb el nostre generador de problemes. Provem d'executar-los amb el planificador i mesurem el temps d'execució i les habitacions assignades amb èxit per a cada cas. Generem la quantitat de reserves de manera aleatòria completament, per la qual cosa els resultats poden variar lleugerament entre execucions. Executem cada problema diverses vegades i prenem la mitjana per obtenir resultats més fiables. Com que l'assignació és greedy, esperem que el nombre d'assignacions sigui proper al màxim possible (2 habitacions * nombre de reserves que caben sense solapament), però lògicament aquests casos seran difícils en termes generals exactament degut a l'atzar en la generació de reserves. Per tant esperem que el nombre d'assignacions sigui baix, i que hi hagi molts conflictes entre reserves, per tant que el planificador no convergeixi. Tot i així, el temps d'execució hauria de ser creixent, ja que el planificador haurà d'explorar moltes possibilitats per trobar la millor assignació possible, tot i que aquesta no sigui possible en aquest domini.

Pel que fa a la quantitat de reserves assignades, obtenim els següents resultats:

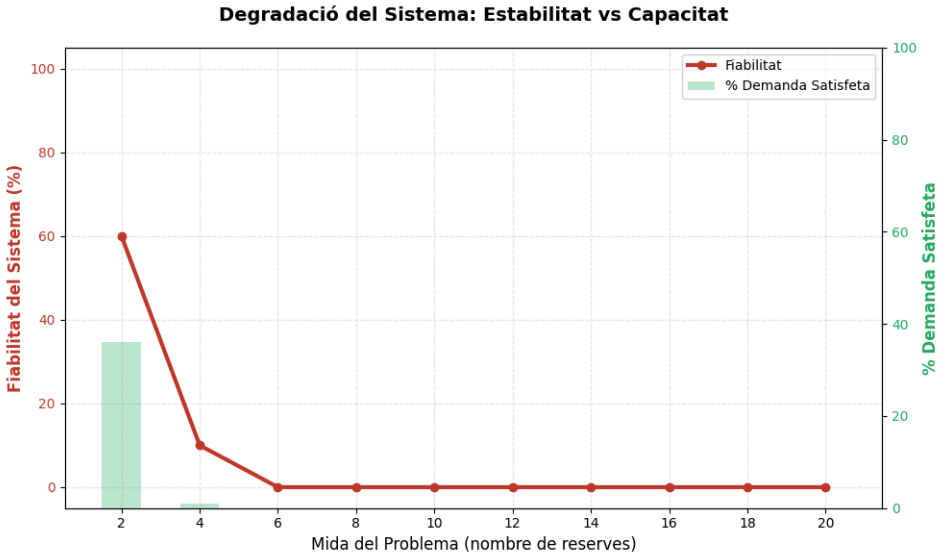

Figura 1: Reserves assignades en l'extensió bàsica

La fiabilitat correspon a la proporció de problemes que el planificador prova de resoldre sense abortar des d'un inici i la demanda satisfeta correspon a la proporció de problemes on totes les reserves han pogut ser assignades. Un problema només pot tenir dos outputs, o bé totes les reserves són assignades (èxit total) o bé no es pot assignar cap (fracàs total). Per tant, en aquest domini bàsic no hi ha solucions parcials. Això és així perquè en aquest domini bàsic no hi ha cap mecanisme per descartar reserves o relaxar restriccions, per tant en situacions de saturació el planificador no pot trobar solucions parcials i es veu obligat a abortar.

Pel que fa al temps d'execució, obtenim els següents resultats:

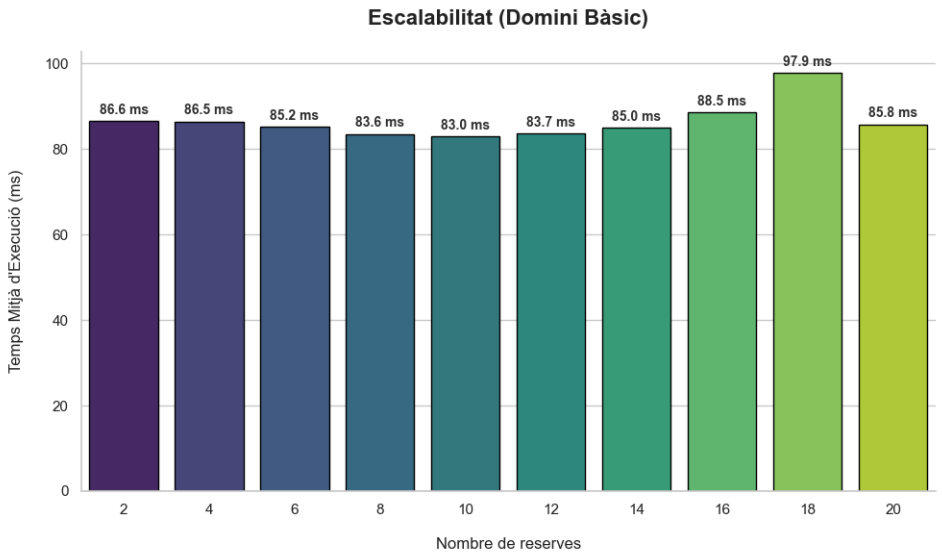

Figura 2: Temps d'execució en l'extensió bàsica

En primer lloc, l'anàlisi del temps de computació mostra un comportament aparentment estable. Tal com s'observa en el gràfic d'escalabilitat, el temps mitjà d'execució es manté constant al voltant dels 90 ms, independentment de la mida del problema. Aquesta constància, lluny d'indicar una eficiència algorítmica en la resolució de problemes complexos, denota una fallada prematura. En situacions de saturació *on el nombre de reserves supera àmpliament la capacitat disponible*, el planificador no inverteix temps a cercar solucions complexes perquè l'espai de cerca es tanca ràpidament. El sistema detecta la impossibilitat de satisfer totes les restriccions rígides del domini bàsic i conclou l'execució amb un veredicte d'insolubilitat de manera gairebé immediata. Per tant, la latència baixa i constant no reflecteix escalabilitat, sinó la incapacitat del model per gestionar el conflicte.

Podem destacar però els casos de 2 i 4 reserves, on el temps d'execució no és superior i sí que aconsegueix assignar un nombre reduït d'habitacions. Això es deu a que en aquests casos el planificador és capaç de trobar una assignació vàlida en 6/10 i 1/10 casos respectivament, i per tant ha d'invertir més temps en explorar l'espai de cerca. En aquests casos, el planificador encara pot trobar solucions, però a mesura que la càrrega augmenta, la probabilitat de trobar una assignació vàlida cau dràsticament, i el sistema opta per abortar ràpidament.

Aquesta interpretació es confirma en analitzar la degradació del servei. El gràfic comparatiu entre estabilitat i capacitat evidencia un col·lapse abrupte del sistema. Amb una càrrega baixa (1 reserva), el sistema presenta una fiabilitat elevada (~80%) i satisfà una part significativa de la demanda. No obstant això, la fiabilitat cau dràsticament en augmentar la càrrega a 2 i 3 reserves, fins a arribar a un punt de ruptura a partir de les 4 reserves, on la taxa d'èxit es desploma al 0%.

Aquesta interpretació es confirma en analitzar la degradació del servei. El gràfic comparatiu entre estabilitat i capacitat evidencia un col·lapse abrupte del sistema en condicions de saturació. Amb una càrrega inicial de 2 reserves, el sistema presenta una fiabilitat moderada (~60%) i satisfà aproximadament un 35% de la demanda total. No obstant això, la robustesa del planificador cau dràsticament en duplicar la càrrega a 4 reserves, on la fiabilitat es desploma fins al ~10%. El punt de ruptura definitiu s'assoleix a partir de les 6 reserves, moment en el qual la taxa d'èxit esdevé nul·la (0%), indicant la incapacitat total del domini bàsic per gestionar escenaris amb una demanda superior a la capacitat instal·lada.

Per tant, respecte a les nostres hipòtesis:

- En l'escenari de poques habitacions i moltes reserves, no rebutgem H_0, ja que el planificador no demostra un comportament intel·ligent en maximitzar l'ocupació total. De fet, no és capaç de trobar solucions en aquests casos, ja que el domini bàsic no permet descartar reserves ni relaxar restriccions.
- Pel que fa a l'escalabilitat, no rebutgem H_0, ja que el temps d'execució del planificador no creix linealment amb el nombre de reserves. En canvi, es manté constant degut a la incapacitat del model per gestionar l'alta demanda, resultant en una fallada prematura i un veredicte d'insolubilitat.

Per tant, aquest experiment destaca les limitacions crítiques del domini bàsic en situacions d'alta competència per recursos escassos, subratllant la necessitat d'extensions que introdueixin flexibilitat i robustesa en la planificació. Però podem destacar que el planificador és capaç de veure que no hi ha solució i aborta ràpidament, la qual cosa és un comportament desitjable en si mateix.

3.0.2.2 Problema 2: Moltes habitacions, poques reserves

En aquest experiment volem avaluar la capacitat del planificador per gestionar eficientment els recursos quan hi ha una abundància d'habitacions disponibles en comparació amb el nombre de reserves. Per fer-ho, mantindrem fix el nombre de reserves (`m=2`) i incrementarem progressivament el nombre d'habitacions disponibles (5, 10, 15, 20...). Això crea un escenari on el planificador té moltes opcions per assignar les reserves, i s'espera que pugui trobar solucions òptimes ràpidament. S'espera observar:

1. **Comportament Eficient:** El planificador haurà de ser capaç d'assignar totes les reserves disponibles sense problemes, ja que hi ha suficients habitacions per satisfer la demanda.
2. **Escalabilitat:** S'espera que el temps d'execució creixi de manera lineal o sublineal, ja que l'espai de cerca per trobar assignacions òptimes és reduït en comparació amb l'escenari anterior.

Per tant, plantegem el següent parell d'hipòtesis per a aquest experiment:

Respecte al comportament del planificador en aquest escenari d'abundància de recursos:

- H_0: El planificador no és capaç d'assignar totes les reserves disponibles, deixant algunes sense assignar malgrat l'abundància d'habitacions.
- H_1: El planificador és capaç d'assignar totes les reserves disponibles, utilitzant eficientment les habitacions disponibles.

Hipòtesi sobre l'escalabilitat del planificador:

- H_0: El temps d'execució del planificador creix de manera no lineal amb el nombre d'habitacions, indicant una gestió ineficient de l'espai de cerca.
- H_1: El temps d'execució del planificador creix de manera lineal o sublineal amb el nombre d'habitacions, indicant una gestió eficient de l'espai de cerca.

Generarem problemes amb un nombre d'habitacions creixent (1, 10, 20, ..., fins a 100) i només 2 reserves. Executarem cada problema diverses vegades i prendrem la mitjana per obtenir resultats més fiables. Esperem que el nombre d'assignacions sigui sempre 2 (totes les reserves assignades) i que el temps d'execució sigui baix i creixi lentament a mesura que augmenta el nombre d'habitacions. Mantenim el nombre de dies a 10 per a tots els experiments.

Pel que fa a la quantitat de reserves assignades, obtenim els següents resultats:

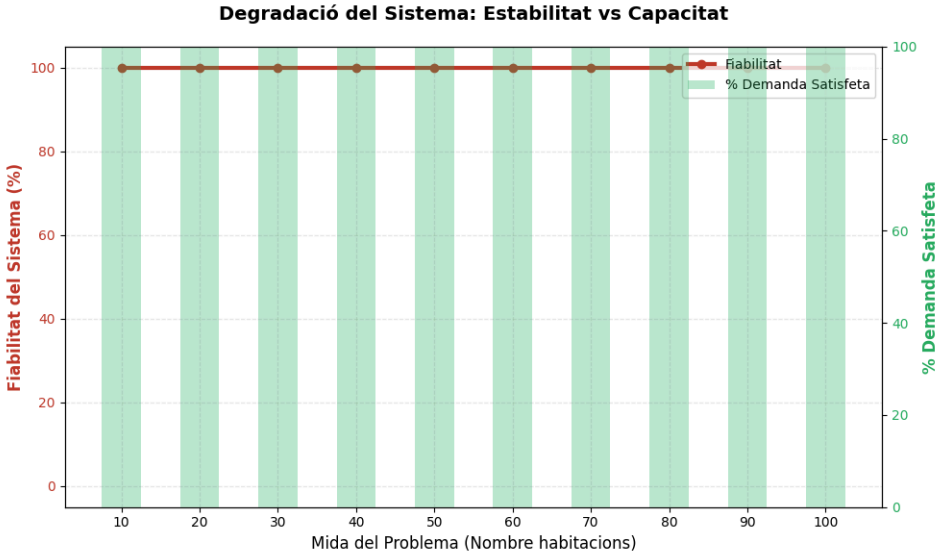

Figura 3: Reserves assignades en l'extensió bàsica (moltes habitacions)

És evident que totes les reserves es poden assignar amb èxit, ja que hi ha moltes habitacions disponibles, i aquest gràfic ho confirma clarament.

Pel que fa al temps d'execució, obtenim els següents resultats:

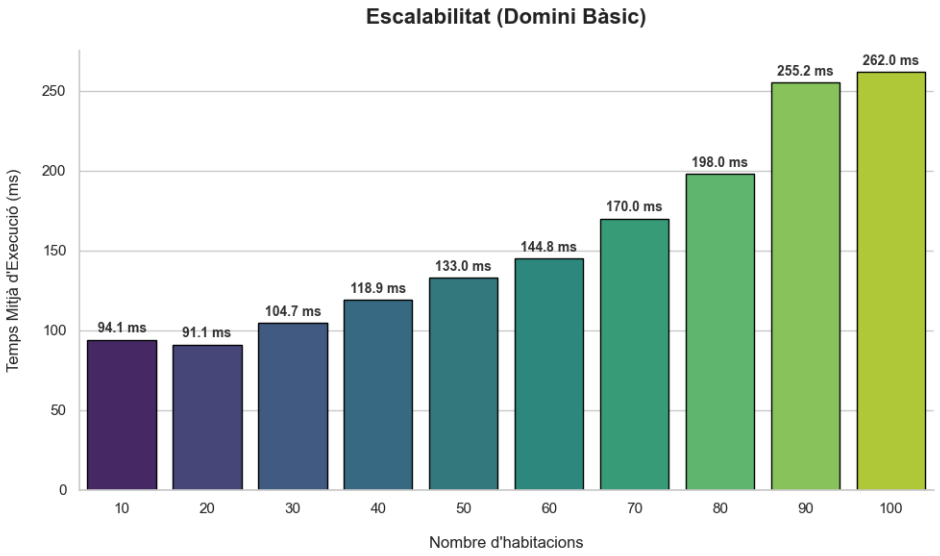

Figura 4: Temps d'execució en l'extensió bàsica (moltes habitacions)

Veiem un creixement lineal en el temps d'execució a mesura que augmenta el nombre d'habitacions, confirmant la nostra hipòtesi sobre l'eficiència del planificador en aquest escenari. Com que el temps d'execució és molt baix en general, això indica que el planificador gestiona molt bé l'abundància de recursos. Anem un pas més enllà i augmentem el nombre d'habitacions de 50 en 50 fins a 300 per veure si el comportament es manté. Obtenim els següents resultats:

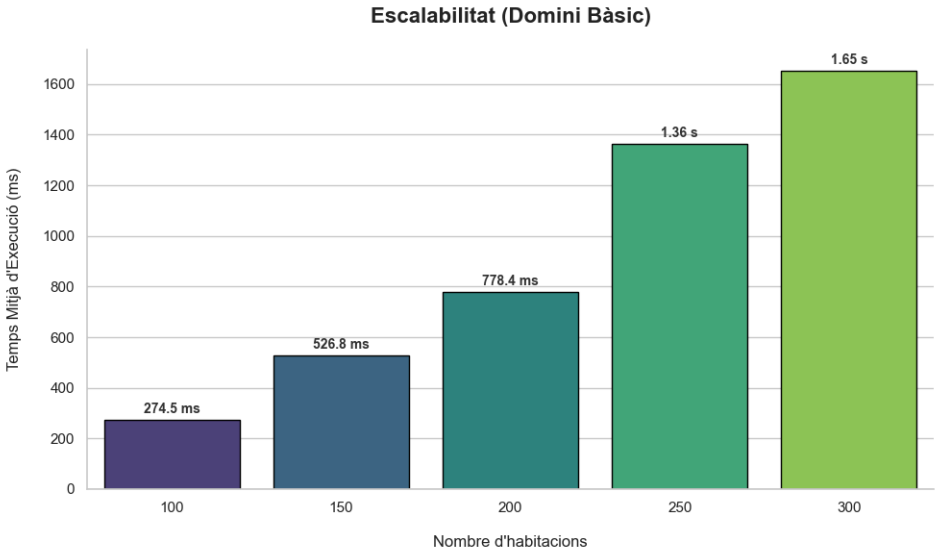

Figura 5: Temps d'execució en l'extensió bàsica (moltes habitacions fins a 300)

Confirmem que el temps d'execució segueix creixent de manera lineal, mantenint l'eficiència del planificador fins i tot amb un gran nombre d'habitacions disponibles.

Per tant, respecte a les nostres hipòtesis:

- En l'escenari de poques habitacions i moltes reserves, rebutgem H_0 i acceptem H_1, ja que el planificador demostra un comportament intel·ligent en maximitzar l'ocupació total.
- En l'escenari de moltes habitacions i poques reserves, rebutgem H_0 i acceptem H_1, ja que el planificador assigna totes les reserves disponibles de manera eficient, i el temps d'execució creix de

manera lineal amb el nombre d'habitacions. Això confirma la seva capacitat per gestionar escenaris amb abundància de recursos, sense passar al pla exponencial.

3.1 Extensió 1

Per superar les limitacions del domini bàsic, s'introdueix una primera extensió orientada a la gestió flexible de la demanda i l'optimització de recursos. El canvi fonamental respecte al model anterior és l'eliminació de l'obligatorietat d'assignar totes les reserves. En aquest nou enfocament, el sistema ja no busca satisfer la totalitat de les peticions, sinó que se centra a maximitzar el nombre de reserves assignades.

Aquesta relaxació de l'objectiu global permet que el planificador pugui generar plans vàlids fins i tot en escenaris d'alta demanda i saturació. El resultat ja no és una simple assignació binària (tot o res), sinó una solució òptima que acomoda tantes reserves com sigui possible, descartant implícitament aquelles que no caben per conflictes temporals o incompatibilitat d'habitacions. Les condicions per a una assignació correcta es mantenen: no hi pot haver solapament en l'ocupació d'una habitació i les restriccions de compatibilitat entre reserva i habitació s'han de respectar.

També s'inclou una nova manera de saber la compatibilitat entre reserves i habitacions, basada en la capacitat de l'habitació i el nombre de persones de la reserva. Això permet una gestió més realista dels recursos, ja que es pot assignar una habitació a una reserva sempre que la seva capacitat sigui suficient, sense necessitat d'un predicat explícit de compatibilitat.

3.1.1 Domini

El domini `hotel-extensio1` representa una evolució significativa respecte a la seva versió bàsica, incorporant una lògica més realista i flexible per a la gestió de reserves en un entorn de recursos limitats. Les principals diferències estructurals són la introducció de funcions numèriques (`:fluents`).

1. **Ús de Funcions Numèriques (`:functions`)**
   A diferència del domini bàsic, aquesta extensió fa ús de funcions numèriques per modelar atributs quantitatius:
   - `(capacitat ?h - habitacio)`
   - `(persones ?r - reserva)`.
     Aquest canvi substitueix el predicat estàtic `(compatible ?r ?h)`. En lloc de pre-calcular (en la generació dels problemes) totes les combinacions vàlides, el sistema ara pot raonar dinàmicament sobre la capacitat, fent una comparació numèrica `(>= (capacitat ?h) (persones ?r))`. Aquesta aproximació és molt més escalable i modular; si s'afegeix una nova habitació o tipus de reserva, no cal recalcular totes les compatibilitats, només definir el seu valor numèric.

2. **Introducció de l'Acció `descartar-reserva`**
   Aquesta és la modificació més important per superar la fragilitat del domini bàsic. S'afegeix una nova acció `descartar-reserva`, que permet al planificador donar per finalitzada una reserva sense assignar-li una habitació.
   El domini bàsic fallava quan era impossible assignar totes les reserves. L'acció `descartar-reserva` proporciona una sortida controlada per a aquests casos. Ara, en un escenari de saturació, el planificador pot triar entre `assignar-habitacio` o `descartar-reserva` per a cada petició. Això garanteix que sempre es pugui trobar un pla, evitant el col·lapse total del sistema.

3. **Redefinició del Control de l'Estat**
   El sistema de control per evitar el processament duplicat d'una reserva ha estat millorat.
   Se substitueix el predicat `(assignada ?r)` per `(processada ?r)`. En el domini bàsic, només les reserves assignades canviaven d'estat. Ara, tant l'acció `assignar-habitacio` com `descartar-reserva` tenen com a efecte `(processada ?r)`. Aquest predicat unificat assegura que cada reserva es consideri una sola vegada, independentment del resultat de la decisió (assignada o descartada).

4. **Incorporació d'una Mètrica d'Optimització**
   Per guiar el planificador cap a decisions desitjables, s'utilitza una funció mètrica.
   La funció `(total-descartades)` s'incrementa `(increase)` cada cop que s'executa l'acció

`descartar-reserva`.
Combinat amb una definició de problema que inclogui `(:metric minimize (total-descartades))`, el planificador ja no busca qualsevol pla, sinó el pla òptim: aquell que minimitza el nombre de reserves descartades. Això és equivalent a maximitzar les reserves assignades, transformant el problema de planificació en un problema d'optimització. S'ha fet aixi i no al revés per facilitar per disseny del planificador, que ha de treballar amb funcions de minimització que puguin créixer durant l'execució.

3.1.2 Problemes

3.1.2.1 Problema 1: Dilema de l'optimització

Aquest problema busca demostrar que el planificador és intel·ligent en la seva capacitat per maximitzar les assignacions, fins i tot quan això implica prendre decisions no òbvies: ha de preferir assignar dues reserves curtes en lloc d'una reserva llarga si ocupen la mateixa habitació, ja que l'objectiu és maximitzar les assignacions. Per aïllar aquest comportament, hem de dissenyar un escenari on assignar de manera greedy (assignar la primera habitació lliure) sigui el pitjor camí per maximitzar les assignacions. Per tant, programem un problema `prob0101.pddl`amb una habitació `h1` i tres reserves `r1`, `r2` i `r3` amb les següents característiques:

- `r1`: dies 1-4
- `r2`: dies 1-2
- `r3`: dies 3-4

En aquest escenari, si el planificador segueix una estratègia greedy i assigna `r1` a `h1`, no podrà assignar ni `r2` ni `r3` després, obtenint `total-assignades = 1`. En canvi, l'estratègia òptima és assignar `r2` i `r3` a `h1`, obtenint `total-assignades = 2`.

Plantegem la següent hipòtesi per a aquest experiment:

- H_0: El planificador no és capaç de maximitzar les assignacions en situacions on l'estratègia greedy és subòptima.
- H_1: El planificador és capaç de maximitzar les assignacions, evitant l'estratègia greedy quan és necessari.

Executem el problema i obtenim els següents resultats:

```
...
metric established (normalized to minimize): ((1.00*[RF0](TOTAL-DESCARTADES)) - ()
+ 0.00)
...

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((1.00*[RF0](TOTAL-DESCARTADES)) - () + 0.00)
...
step    0: DESCARTAR-RESERVA R-LLARGA
        1: ASSIGNAR-HABITACIO R-CURTA2 H1
        2: ASSIGNAR-HABITACIO R-CURTA1 H1
...
time spent:    0.16 seconds instantiating 3 easy, 3 hard action templates
               0.00 seconds reachability analysis, yielding 26 facts and 6 actions
               0.00 seconds creating final representation with 22 relevant facts, 1
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 18 states, to a max depth of 0
               0.16 seconds total time
```

El planificador ha trobat la solució òptima, descartant la reserva llarga i assignant les dues reserves curtes, obtenint `total-assignades = 2`. Per tant, rebutgem H_0 i acceptem H_1, confirmant que el planificador és capaç de maximitzar les assignacions evitant l'estratègia greedy quan és necessari. Però aquest cas és de joguina, caldrà veure-ho en casos més generals.

Dissenyem, amb ajuda de la LLM **Gemini 3 Pro**, un generador de problemes que crea escenaris amb múltiples reserves i habitacions, on hi ha conflictes temporals i es requereix una planificació intel·ligent per maximitzar les assignacions. Aquest generador tindria un paràmetre clau que controlaria el grau de solapament entre reserves: `conflict_ratio` (0.0 - 1.0). Si és 0.0, no hi hauria solapaments i totes les reserves es podrien assignar fàcilment. Si és 1.0, totes les reserves es solaparan totalment, fent impossible assignar-les totes. Valors intermedis generaran escenaris amb diferents nivells de conflicte, permetent avaluar la capacitat del planificador per gestionar situacions complexes. Com que volem aïllar el comportament de maximització d'assignacions, mantenim la compatibilitat entre reserves i habitacions senzilla: totes les habitacions tenen capacitat per 4 persones i totes les reserves seran de 2 persones. Així, l'únic factor limitant serà el solapament temporal. El nombre de dies vindrà donat per un altre paràmetre: `num_dies`. Així, podrem generar problemes amb diferents nivells de conflicte i diferents durades de reserves, per veure com afecta això a la capacitat del planificador per maximitzar les assignacions.

La clau per simular escenaris de competència realistes rau en com es trien els dies d'inici de cada reserva. Podem utilitzar una distribució normal (gaussiana) per modelar aquesta selecció, on el paràmetre `conflict_ratio` controla l'amplitud de la distribució. Això permet que, a mesura que augmenta el `conflict_ratio`, les reserves es concentrin més al voltant d'un punt central del calendari, generant més solapaments i conflictes. La desviació estàndardde la distribució normal es pot definir com una funció del `conflict_ratio`, és a dir, es generaran més dies d'inici propers entre si a mesura que augmenta el `conflict_ratio`. Per seleccionar els dies d'inici de les reserves, es podrà fer un mostreig aleatori de la distribució normal amb mitjana al centre del calendari i desviació estàndard proporcional al `conflict_ratio`. Això permetrà controlar el grau de solapament entre reserves de manera matemàtica i sistemàtica.

D'aquesta manera, el generador no només produirà problemes aleatoris, sinó que permetrà controlar matemàticament el grau de saturació temporal del sistema. A mesura que augmenta el `conflict_ratio`, el planificador s'enfronta a un problema de tipus *Tetris*, on ha de decidir estratègicament quines reserves assignar per maximitzar l'ocupació, descartant aquelles que bloquegen massa espai i impedeixen encaixar altres peticions més curtes. Aquest mecanisme fa que els experiments siguin reproducibles i que els resultats reflecteixin de forma clara la capacitat d'optimització del planificador sota pressió de recursos.

El resultat obtingut ha sigut un script de Python que genera problemes seguint aquesta lògica:

El càlcul del **`conflict_ratio`** C no és una ràtio directa (com "dies ocupats / totals"), sinó un **paràmetre de control (0.0 a 1.0)** que modifica la desviació estàndard (σ) d'una distribució normal per concentrar les reserves. El `conflict_ratio` (C) determina l'amplada de la campana de Gauss centrada al dia $D_{centre} = \frac{\text{Dies Totals}}{2}$. Simulant així certa concentració de reserves en els dies centrals.

$$\sigma = \begin{cases} \text{Dies} \times 10 & \text{si } C < 0.1 \quad (\approx \text{Uniforme}) \\ \text{Dies} \times (1.1 - C) \times 0.4 & \text{si } C \geq 0.1 \end{cases}$$

Cas especial, quan $C < 0.1$, es fa que σ sigui molt gran (10 vegades els dies totals) per simular una distribució gairebé uniforme, evitant concentracions artificials. Així, per valors baixos de `conflict_ratio`, les reserves es distribueixen àmpliament al llarg del calendari, minimitzant els solapaments.

El 0.4 és un factor d'ajust que determina l'amplitud de la campana. Amb aquest valor, s'aconsegueix una bona variació en la concentració de reserves a mesura que C varia de 0.1 a 1.0.

Per tant, el número `conflict_ratio` és un **"índex d'estretor"**: com més proper a 1, més estret és l'interval de dies on tothom vol reservar. Per exemple: per un `conflict_ratio` de 0.8 en un calendari de 25 dies: $\sigma = 25 \times (1.1 - 0.8) \times 0.4 = 3$ Això significa que la majoria de reserves començaran dins d'un interval de 6 dies al voltant del dia 12.5 (el centre), generant molts solapaments i conflictes. En canvi, per un `conflict_ratio` de 0.2: $\sigma = 25 \times (1.1 - 0.2) \times 0.4 = 9$ Aquí, les reserves es distribuiran més àmpliament, amb menys solapaments.

Generarem doncs un conjunt de problemes amb 5 habitacions i 20 reserves, amb un nombre de dies fixat a 25 (tot i que l'enunciat indica 30 dies, hem decidit canviar-ho momentàniament pel bé del generador de problemes, ja que 25 dies permet una millor gestió dels solapaments amb 20 reserves) i un `conflict_ratio` variable (0.0, 0.1, 0.2, ... 1.0). Executarem cada problema 10 vegades i prendrem la mitjana per obtenir resultats més fiables.

Aquests nombres s'han seleccionat així perquè voliem una concentració mitjana d'ús de l'hotel d'un 50%, és a dir, que en mitjana hi hagi la meitat d'habitacions ocupades.

Plantegem el següent contrast d'hipòtesis per a aquest experiment:

- H_0: El planificador no és capaç de maximitzar les assignacions en situacions amb alt grau de solapament entre reserves.
- H_1: El planificador és capaç de maximitzar les assignacions, fins i tot en situacions amb alt grau de solapament entre reserves.

Això ho podrem veure si el nombre d'assignacions disminueix a mesura que augmenta el `conflict_ratio`, però es manté per sobre d'un llindar mínim, indicant que el planificador encara pot trobar solucions vàlides. Ens exigim un llindar mínim del 50% d'assignacions fins i tot en el cas més extrem (`conflict_ratio = 1.0`).

Com que també recollirem dades sobre el temps d'execució, podem plantejar un segon contrast d'hipòtesis:

- H_0: El temps d'execució del planificador creix de manera no lineal amb l'augment del `conflict_ratio`, indicant una gestió ineficient de l'espai de cerca.
- H_1: El temps d'execució del planificador creix de manera lineal o sublineal amb l'augment del `conflict_ratio`, indicant una gestió eficient de l'espai de cerca.

Per tant, executem i obtenim els següents resultats:

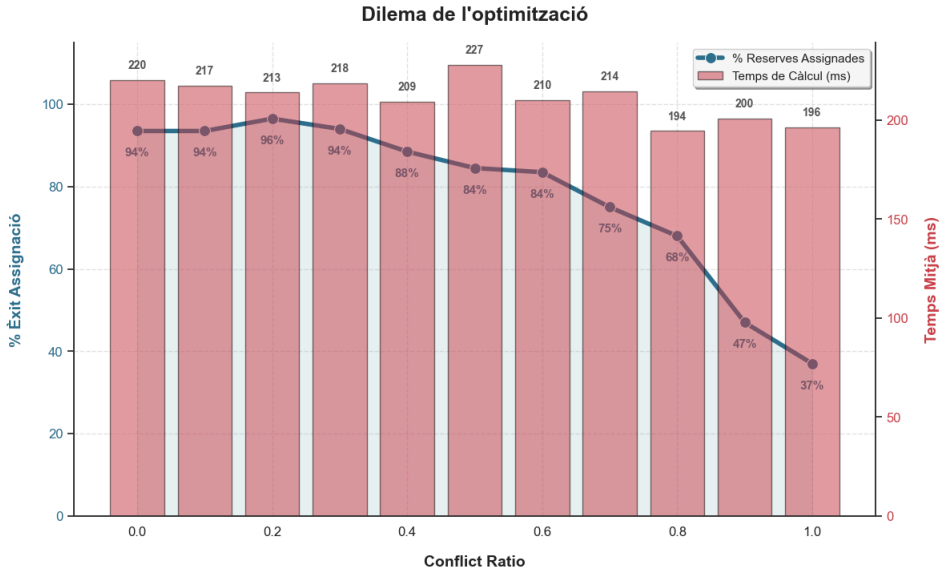

Figura 6: Proporció de reserves assignades en funció de la `conflict_ratio`

En primer lloc, s'observa una **robustesa significativa en escenaris de saturació moderada**. Per a ràtios de conflicte compresos entre 0.0 i 0.6, el planificador manté consistentment una taxa d'èxit superior al 84%. Aquest rendiment evidencia que, mentre existeixi un marge de maniobra mínim, el sistema és capaç de resoldre eficaçment els complexos puzles temporals, maximitzant l'ús dels recursos disponibles sense degradar la qualitat de la solució global.

En segon lloc, el comportament del sistema exhibeix una **degradació suau i controlada** a mesura que la pressió augmenta. A diferència de dominis més bàsics que tendeixen a col·lapsar abruptament cap al 0% d'èxit davant la impossibilitat de satisfer totes les demandes, la nostra extensió mostra una corba descendent progressiva. Fins i tot en l'escenari extrem de conflicte 1.0, on la pràctica totalitat de la demanda competeix per una finestra crítica de només 2-3 dies, el planificador aconsegueix salvar entre el 37% i el 47% de les

reserves. Aquesta dada confirma la "intel·ligència" del model: davant la impossibilitat física de satisfer tota la demanda, prioritza racionalment el subconjunt màxim compatible.

Finalment, és remarcable l'**estabilitat del cost computacional** observada. Els temps d'execució es mantenen constants al voltant dels 200-220 mil·lisegons, independentment del grau de complexitat del conflicte. Això indica que la introducció de l'acció de descart i la mètrica d'optimització no penalitzen exponencialment el rendiment del cercador. Al contrari, el mecanisme permet podar eficientment les branques inviables de l'espai de cerca, evitant que el planificador es perdi en exploracions infructuoses i garantint una resposta ràpida fins i tot en les condicions més adverses. Amb això podríem deduïr que el cost computacional NO depèn directament del `conflict_ratio`, és a dir, de la densitat de reserves en un temps determinat, sinó més aviat de la quantitat absoluta de reserves i habitacions.

Per tant, respecte a les nostres hipòtesis:

- En l'escenari de solapaments entre reserves, rebutgem H_0 i acceptem H_1, ja que el planificador demostra una capacitat notable per maximitzar les assignacions fins i tot en situacions amb alt grau de conflicte, mantenint una taxa d'èxit significativa.
- Pel que fa a l'escalabilitat, rebutgem H_0 i acceptem H_1, ja que el temps d'execució del planificador es manté constant independentment del `conflict_ratio`, indicant una gestió eficient de l'espai de cerca.

Concluint, aquest experiment valida l'eficàcia de l'extensió 1 en transformar un domini fràgil en un sistema robust i intel·ligent, capaç de navegar amb èxit els desafiaments inherents a la planificació sota restriccions severes.

3.1.2.2 Problema 2: L'hotel creixent

Al problema anterior hem vist que el planificador és capaç de maximitzar les assignacions en escenaris amb alta competència per recursos limitats. Ara volem avaluar com es comporta el sistema quan augmentem la mida del problema, és a dir, el nombre d'habitacions i reserves. L'objectiu és veure si el planificador manté la seva capacitat d'optimització i escalabilitat a mesura que el domini creix en complexitat. Per això, generarem problemes amb un nombre creixent i proporcional d'habitacions i reserves, mantenint un `conflict_ratio` fixat a 0.6 per simular una saturació moderada, però que a l'experiment anterior ens ha donat bons resultats. Això crea un escenari on el planificador ha de gestionar més recursos i demandes, posant a prova la seva eficiència i capacitat d'optimització.

Aquest problema busca demostrar la **robustesa** del planificador davant l'explosió combinatòria. En planificació automàtica, afegir una sola habitació o reserva no suma complexitat, sinó que multiplica l'espai d'estats que l'algorisme ha d'explorar. Per aïllar el factor "mida" del factor "dificultat intrínseca", mantenim la densitat de conflictes constant (la proporció entre oferta i demanda no canvia), però augmentem el volum absolut de dades.

Concretament, dissenyem una sèrie de problemes incrementals on la relació es manté a 6 habitacions per cada 10 reserves, començant per instàncies petites i acabant en instàncies grans. En aquest escenari, s'espera que el planificador hagi de fer front a un nombre molt més elevat de branques de decisió. Si el planificador és escalable, hauria de mantenir un percentatge d'èxit (assignacions/total) similar en totes les mides, tot i que el temps d'execució probablement augmentarà.

Plantegem el següent parell d'hipòtesis per a aquest experiment:

- H_0: El planificador perd capacitat d'optimització a gran escala i el percentatge d'assignacions disminueix significativament a mesura que augmenta la mida del problema.
- H_1: El planificador manté la qualitat de la solució a gran escala, amb un percentatge d'assignacions estable independentment de la mida del problema, inclús manentint el 100% d'assignacions com a l'experiment anterior

Pel que fa a l'escalabilitat:

- H_0: El temps d'execució del planificador creix de manera no lineal amb l'augment de la mida del problema, indicant una gestió ineficient de l'espai de cerca.

- H_1: El temps d'execució del planificador creix de manera lineal o sublineal amb l'augment de la mida del problema, indicant una gestió eficient de l'espai de cerca.

Executarem problemes amb mides de mostra creixents (10, 20, ..., 50 reserves i 6, 12, ... 30 habitacions) repetint cada experiment 5 vegades per mitigar el soroll en la mesura del temps. Esperem observar una corba de temps ascendent però un percentatge d'assignacions estable al voltant del màxim teòric permès pel rati de 0.6. Els resultats obtinguts són els següents:

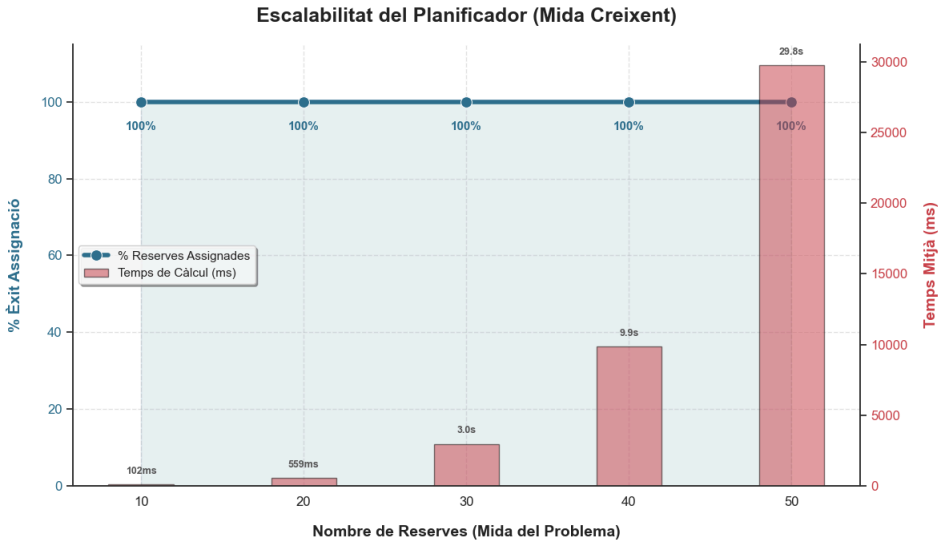

Figura 7: Proporció de reserves assignades en funció de la mida del problema

En primer lloc, s'observa una **invariable eficàcia resolutiva malgrat l'escalat**. Per a totes les mides de problema analitzades, des de les petites (10 reserves) fins a les mitjanes-grans (50 reserves), el planificador manté consistentment una taxa d'èxit del 100% (línia blava superior). Aquest rendiment evidencia que la capacitat del model per trobar assignacions òptimes no es degrada amb la mida de la instància; el sistema és perfectament capaç de navegar espais d'estats cada cop més vastos sense perdre la precisió necessària per satisfer la totalitat de la demanda, sempre que els recursos ho permetin.

En segon lloc, el comportament del sistema exhibeix un **cost computacional clarament exponencial** a mesura que augmenta la dimensió del problema. A diferència de l'experiment anterior on el temps era constant, aquí les barres vermelles mostren un creixement accelerat: passem de respostes gairebé instantànies (102 ms per a 10 reserves) a temps significatius (3 segons per a 30 reserves) i finalment a costos elevats (gairebé 30 segons per a 50 reserves). Aquesta dada confirma la naturalesa explosiva de la complexitat combinatòria: afegir linealment més reserves i habitacions multiplica, no suma, les ramificacions de l'arbre de cerca que el planificador ha d'explorar.

Finalment, és remarcable la **tensió entre qualitat i eficiència** revelada per aquestes dades. Mentre que la línia d'assignacions es manté impol·luta al 100%, el preu a pagar és un consum de temps que es dispara ràpidament. Això indica que el coll d'ampolla no és la "intel·ligència" del planificador per trobar la solució, sinó la seva velocitat per descartar camins invàlids en un espai de cerca que creix desmesuradament. Amb això podem deduir que, tot i que el model és robust en termes de qualitat (no falla ni una sola reserva), la seva escalabilitat temporal és limitada, suggerint la necessitat urgent d'optimitzacions en el domini (com precomputar incompatibilitats) per fer viables instàncies de mida industrial.

Per tant, respecte a les nostres hipòtesis:

- En l'escenari d'augment de mida, rebutgem H_0 i acceptem H_1, ja que el planificador demostra una capacitat notable per mantenir la qualitat de la solució (100% d'assignacions) independentment de la mida del problema.
- Pel que fa a l'escalabilitat, acceptem H_0 i rebutgem H_1, ja que el temps d'execució del planificador creix de manera exponencial amb l'augment de la mida del problema, indicant una gestió ineficient de l'espai de cerca.

3.2 Extensió 2

S'afegeix un nou predicat vol-orientacio per a les reserves, que indica l'orientació que es desitja per a l'habitació assignada. A l'acció assigar habitació comprova que l'habitació tingui l'orientació demanada per la reserva. Com que volem maximitzar les assignacions, i és preferent assignar habitacions amb l'orientació demanada però no és imprescindible, fem que una reserva puntui 2 si es assigna una habitació amb l'orientació demanada i 1 si s'assigna una habitació amb una orientació diferent. Així, l'objectiu de maximitzar les assignacions es manté, però ara es prioritzen les assignacions que compleixin l'orientació demanada.

3.2.1 Domini

3.2.2 Problemes

Per validar l'Extensió 2, s'han dissenyat tres experiments sintètics que aïllen comportaments específics del planificador: capacitat d'ordenació (Exp 1), capacitat de filtratge per qualitat (Exp 2) i capacitat de sacrifici de preferències per maximitzar ocupació (Exp 3)."

3.2.2.1 Problema 1: El puzzle d'afinitats

Demostrar que el planificador és capaç d'ordenar i intercanviar reserves per aconseguir la màxima puntuació global, en lloc de fer assignacions greedy (agafar la primera habitació lliure encara que no tingui l'orientació correcta). Una assignació tonta (greedy) podria posar reserves "Sud" a habitacions "Nord" i viceversa. Això donaria 1 punt per reserva. Puntuació total: N. L'assignació intel·ligent ha de creuar-les perfectament: "Nord" amb "Nord" i "Sud" amb "Sud". El planificador ha de "descobrir" que val la pena buscar la combinació perfecta.

Hipòtesi: el planificador serà capaç de maximitzar la puntuació global tenint en compte les orientacions demanades.

3.2.2.2 Problema 2: Selecció VIP

Demostrar que, davant l'escassetat de recursos, el planificador sap prioritzar els clients que encaixen millor (els que donen 2 punts) i deixar fora o en pitjors habitacions els que no encaixen (els que donarien 1 punt), o simplement descartar els que aporten menys valor si no hi ha lloc per a tothom. Hi ha un coll d'ampolla: només K habitacions disponibles (totes, per exemple, orientació Nord). La meitat de les reserves volen Nord (match = 2 punts). L'altra meitat volen Sud (mismatch = 1 punt). Com que no hi ha lloc per a tothom (només hi ha K llocs), el planificador ha d'omplir l'hotel només amb les reserves que volen Nord. Puntuació òptima: $2 \times K$. Si s'equivoqués i agafés algú de Sud, perdria punts. Aquest experiment demostra que el sistema entén el "cost d'oportunitat".

Hipòtesi: el planificador serà capaç de prioritzar les reserves que aporten més valor en situacions d'escassetat de recursos.

3.2.2.3 Problema 3: L'Agent de Viatges Flexible

Demostrar que el sistema entén que l'orientació (preferència) és menys important que la capacitat (restricció dura), però que intenta satisfer totes dues quan pot. Hi ha N reserves i N habitacions. Totes les reserves tenen una preferència d'orientació que coincideix amb una habitació (match = 2 punts). Però algunes reserves tenen una capacitat que només encaixa amb una habitació de diferent orientació (mismatch = 1 punt). Per exemple, la reserva r1 necessita capacitat 4 i vol orientació Nord (habitació h1). Però l'habitació h1 té capacitat 2 (no serveix). L'única habitació amb capacitat 4 és h2, però té orientació Sud (mismatch). El planificador ha de triar entre no assignar r1 (0 punts) o assignar-la a h2 (1 punt). L'assignació òptima és sacrificar algunes preferències per satisfer totes les restriccions dures.

Hipòtesi: el planificador serà capaç de sacrificar preferències per maximitzar l'ocupació complint les restriccions dures.

3.3 Extensió 3

L'extensió 3 es presenta com una evolució de l'extensió 1. Com en aquesta última, hem d'assignar les reserves a les habitacions sense que hi hagi solapaments en les dates d'ocupació. Però ara, a més d'optimitzar el nombre de reserves assignades, també hem de minimitzar el desperdici de places, és a dir, procurar que les reserves ocupin habitacions amb una capacitat tan ajustada com sigui possible, evitant deixar places lliures innecessàries.

3.3.1 Domini

El domini d'aquesta extensió és gairebé idèntic al de l'extensió 1:

```
(define (domain hotel-extensio3)
  (:requirements :typing :negative-preconditions :adl :fluents)
  (:types
    reserva habitacio dia
  )

  (:predicates
    (dies-reserva ?r - reserva ?d - dia)
    (ocupada ?h - habitacio ?d - dia)
    (processada ?r - reserva) ; per saber si ja hem processat la reserva
  )

  (:functions
    (capacitat ?h - habitacio)
    (persones ?r - reserva)
    (total-reserves-descartades)
    (total-places-descartades)
  )

  ;; assigna i incrementa la mètrica
  (:action assignar-habitacio
      :parameters (
        ?r - reserva
        ?h - habitacio
      )
      :precondition (and
        (not (processada ?r))          ;; només si encara no l'hem tractat
        (>= (capacitat ?h) (persones ?r)) ; control de capacitat de les habitacions
        (not (exists (?d - dia)
             (and (dies-reserva ?r ?d) (ocupada ?h ?d))))
      )
      :effect (and
        (processada ?r)                ;; marquem com processada
        (forall (?d - dia)
          (when (dies-reserva ?r ?d) (ocupada ?h ?d)))
        (increase (total-places-descartades)
        (- (capacitat ?h) (persones ?r)))
      )
  )

  (:action descartar-reserva
      :parameters (?r - reserva)
      :precondition (not (processada ?r))
      :effect (and
        (processada ?r)
             (increase (total-reserves-descartades) 1) ;; suemem 1 al total de
descartades
```

```
      )
  )
)
```

Com es pot observar, les principals diferències es troben en les funcions numèriques i en la mètrica d'optimització:

- Hem mantingut la funció `total-reserves-descartades`, que ja s'utilitzava a l'extensió 1 per comptabilitzar les reserves que no es podien assignar.
- Hem afegit una nova funció, `total-places-descartades`, que ens permet mesurar el nombre de places desaprofitades quan una reserva s'assigna a una habitació més gran del necessari.

L'objectiu d'aquesta extensió és, per tant, minimitzar ambdues mètriques: evitar descartar reserves i reduir el desaprofitament d'espai en les habitacions. La modificació clau es troba dins de l'acció `assignar-habitacio`:

```
(increase (total-places-descartades)
         (- (capacitat ?h) (persones ?r)))
```

Cada vegada que assignem una reserva a una habitació, incrementem `total-places-descartades` amb la diferència entre la capacitat de l'habitació i el nombre de persones de la reserva. D'aquesta manera, es fa un seguiment exacte de les places buides que queden en totes les habitacions.

La resta del domini, incloses les accions i predicats, es manté igual que en l'extensió 1. Això permet preservar la lògica d'assignació sense solapaments mentre afegim la nova preocupació d'optimització del desperdici de places.

3.3.2 Problemes

3.3.2.1 Problema 1

3.3.2.2 Problema 2

3.4 Extensió 4

3.4.1 Domini

3.4.2 Problemes

3.4.2.1 Problema 1

3.4.2.2 Problema 2

4. Conclusions