# Projecte d'Algorísmia:
# Mínim Conjunt Dominador d'Influència Positiva

*Jaume Casals, Adrià Fernández, Martí Juanola, Pol Sturlese*

Curs 2021-2022, Quadrimestre de Tardor

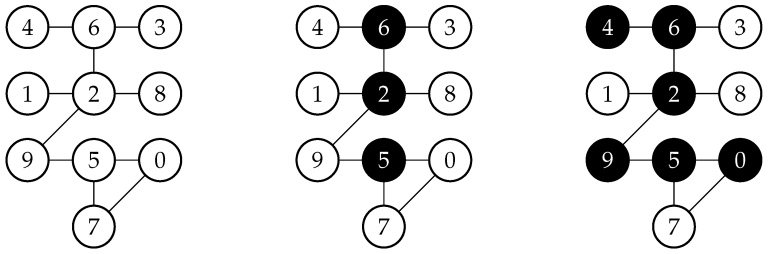
# Continguts

# 1 Introducció

El problema del Mínim Conjunt Dominador d'Influència Positiva ens planteja que, donat un graf connex i no dirigit $G = (V, E)$, trobem el conjunt $D$ de vèrtexs de mínima cardinalitat possible tal que tots els vèrtexs del graf tenen, com a mínim, el 50% dels seus vèrtexs adjacents dins del conjunt $D$. En altres paraules, sigui $g(v)$ el grau del vèrtex $v$, i sigui $a_D(v)$ el nombre de vèrtexs adjacents de $v$ que pertanyen a $D$, volem trobar el mínim conjunt $D$ que compleixi:

$$\forall v | v \in V \implies a_D(v) \geq \lceil g(v)/2 \rceil$$

Per exemple, en la Figura 1.b el conjunt dominant no és d'influència positiva donat que té vèrtexs que no compleixen la condició, com el vèrtex 6. Podem veure un exemple que sí que compleix la condició en la Figura 1.c.




(**a**) A simple, connected undirected graph

(**b**) A dominating set

(**c**) A positive influence dominating set

Figura 1: Exemple de Graf, Graf amb DS i Graf amb PIDS [6]

Aquest problema s'ha demostrat que és dins de la classe dels problemes NP-Difícil [15] [16], per tant el seu estudi és d'especial interés. També té aplicacions en les xarxes socials en línia (per exemple, intentant propagar un missatge emprant el mínim nombre d'usuaris possibles) [15] [16] i s'ha plantejat el seu ús en afrontar problemes d'addicció a l'alcohol [16].

El projecte està dividit en dues parts. La primera part consisteix en els apartats Algorismes de Comprovació, Algorismes Golafres i Cerca Local; mentre que la segona part la conformen els apartats Metaheurística i Programació Linear. Finalment, a l'apartat Conclusions hem comparat els resultats de tots els programes i hem elaborat sobre els coneixements adquirits durant l'elaboració d'aquest projecte.

# 2 Algorismes de Comprovació

Per al bon funcionament dels algorismes de generació de solucions s'ha hagut de fer dos algorismes per a poder comprovar la seva correctesa: Un per a comprovar si el set resultant és un PIDS i un altre per a saber si aquest PIDS és minimal.

Aquests algorismes s'usen bastant en alguna part de la pràctica, ja que alguns dels algorismes implementats genera moltes possibles solucions, i és important saber si aquestes són realment solucions o no. Per aquest motiu s'ha decidit emprar unes estructures de dades addicionals i així millorar-ne l'eficiència.

## 2.1 Estructures de Dades

Les estructures de dades que es mencionen a continuació han estat pensades per a ser usades tant en els algorismes de comprovació com per els algorismes que resolen el problema si es cregués convenient. Com a notació es prendrà $N$ com el nombre de vèrtexs del graf i $M$ el nombre d'arestes d'aquest.

**vector<unordered_set<int>>**: Aquest vector de sets no ordenats d'enters s'annomena *neighbors* i permet mantenir una estructura del graf en la qual l'accés és eficient. Cada element del vector representa un node del graf, i cada set desordenat representa els veïns que té cada node. La implementació que fa c++ d'un *unordered_set* és mitjançant funcions de hash, i, per tant, el temps mitjà d'accés és constant. Addicionalment, al no estar ordenats, tenen un cost inferior als elements de la classe *set*.

**vector<int>**: Aquest vector d'enters s'anomena *NND* i emmagatzema en cada una de les seves posicions (que representa un node del graf) el nombre de nodes veïns que es troba en el set dominador. L'accés i la modificació dels elements d'aquest vector té cost constant, cosa que el fa una bona opció si s'han de realitzar moltes consultes.
Aquest vector serveix principalment per a saber si un conjunt escollit és dominador o no. S'inicialitza amb N posicions buides (sent N el nombre de nodes) i s'omple després de la lectura d'una solució.

**vector<int>**: Aquest vector de booleans s'anomena $D$ i emmagatzema en cada una de les seves posicions (que representa un node del graf) si el node pertany al conjunt dominador escollit o no. L'accés i la modificació dels elements d'aquest vector té cost constant, cosa que el fa una bona opció si s'han de realitzar moltes consultes.

## 2.2 Comprovació de PIDS

Per a la comprovació de si un conjunt és dominador, s'ha creat una funció que donat un vector d'enters *NND* i el graf $G$, retorna *-1* si el conjunt és dominador o bé retorna la primera posició que es comprova que no compleix la restricció que com a mínim la meitat dels veïns pertanyin al conjunt dominador.
El pseudocodi de l'algorisme proposat és el següent:

**Algorithm 1** Algorsime de comprovació de conjunt dominador

**Input:** Graf $G = (V, E)$ no dirigit i connex, Vector d'enters *NND*
**Output:** *-1* si NND és un conjunt dominador, $i \in [0, N)$ altrament

1: **for** each $i \in [0, N)$ **do**
2: **if** $NND[i] < \left\lceil \frac{G.get\_nb\_veins(i)}{2} \right\rceil$ **then**
3: **return** $i$
4: **end if**
5: **end for**
6: **return** $-1$

NOTA: La funció $get\_nb\_veins(i)$ retorna el nombre de vèrtexs adjacents al vèrtex en posició i.

Per la implementació d'aquest algorsime s'han usat els vectors *neighbors* per la representació del graf $G$ i el vector *NND*, tots dos mecionats en l'apartat d'estructures de dades. Per tant, la funció $get\_nb\_veins(i)$ seria equivalent a fer la comanda $neighbors[i].size()$ en c++, que té un cost temporal constant.

L'algorisme fa un recorregut per tots els vèrtexs amb cost $O(N)$, i per cada un d'ells fa un accès al vector *NND*, una crida a la funció per consultar el nombre de veïns i unes operacions de divisió, tot amb cost constant. Per tant, el cost de l'algorisme serà de $O(N)$.

## 2.3 Comprovació de PIDS Minimal

Per a la comprovació de si un conjunt dominador és minimal, s'ha creat una funció que donat el graf *G*, un vector d'enters *NND* i un vector de booleans *D*, retorna *-1* si el conjunt és dominador o bé retorna la primera posició que es comprova que no compleix la restricció de minimalitat. Això ho fa comprovant per cada vèrtex del conjunt dominador si és pot eliminar o no del conjunt mantenint la restricció que tots els nodes tinguin almenys la meitat de nodes veïns pertanyebnt al conjunt dominador.

Com que l'algorisme comprova si un conjunt dominador és minimal, els vectors D i NDD hauràn de proporcionar una solució que impliqui la dominància del conjunt escollit.

El pseudocodi de l'algorisme proposat és el següent:

**Algorithm 2** Algorsime de comprovació de conjunt dominador

**Input:** Graf $G = (V, E)$ no dirigit i connex, Vector d'enters *NND*, Vector de booleans *D*. (*NDD* i *D* especifiquen un conjunt dominador de G)
**Output:** *-1* si NND és un conjunt dominador minimal, $i \in [0, N)$ altrament

1: **for** each $i \in [0, N)$ **do**
2:  **if** $D[i]$ **then**
3:  **for all** $v \in neighbors\_vertex(G, i)$ **do**
4:  **if** $NND[v] - 1 < \left\lceil \frac{G.get\_nb\_veins(v)}{2} \right\rceil$ **then**
5:  **return** $i$
6:  **end if**
7:  **end for**
8:  **end if**
9: **end for**
10: **return** $-1$

NOTA: La funció $get\_nb\_veins(v)$ retorna el nombre de vèrtexs adjacents al vèrtex en posició $v$.

NOTA: La funció $neighbors\_vertex(G, i)$ retorna les posicions dels veïns del vertex en posició $i$ en el graf *G*.

Per la implementació d'aquest algorsime s'han usat els vectors *neighbors* per la representació del graf *G*, el vector *NND* i el vector *D*, tots dos mecionats en l'apartat d'estructures de dades. Per tant, la funció $get\_nb\_veins(v)$ seria equivalent a fer la comanda $neighbors[v].size()$ en c++, que té un cost temporal constant.

El bucle interior (el que itera sobre tots els veins del vèrtex en posició $i$) s'ha implementat en c++ usant iteradors. Per tant seria un recorregut sobre el conjunt d'enters que retorna $neighbors[i]$.

L'algorisme fa un recorregut per tots els $N$ vèrtexs, i per cada un itera sobre les seves arestes. Per tant, amb els 2 bucles es té un cost de $O(N + M)$. Les comprovacions que es fan en les sentències condicionals són de cost constant (com s'ha explicat en l'apartat d'estructura de dades i en la funció de comprovació de si un conjunt és dominador, on la condició era quasi idèntica a la de la línia 4). Per tant, el cost total de l'algorisme és de $O(N + M)$, un cost lineal.

# 3 Algorismes Golafres

Un algorisme greedy o algorisme voraç és un tipus d'algorisme normalment utilitzat en problemes de combinatòria on l'objectiu és maximitzar o minimitzar un criteri. Aquests algorismes es basen a crear una solució pas a pas, on a cada iteració es millora la solució actual segons un criteri local. Cada cicle de l'algorisme es crea un subproblema derivat de la solució anterior afegint la millor opció actual per intentar arribar a un òptim.
Per mostrar un exemple, l'algorisme de la següent figura s'observa un algorisme voraç general basat en la construcció d'una solució en un graf.

**Algorithm 3** Algorsime golafre genèric

**Input:** Graf $G$ no dirigit i connex
**Output:** PIDS (Conjunt dominant d'influència positiva)

1: $D \leftarrow \emptyset$
2: **while** D is not a PIDS **do**
3: $v* \leftarrow argmax_{\forall v \notin S}\{funcioGreedy(v)\}$
4: $D \leftarrow D \cup \{v*\}$
5: $actualitzarDades(v)$
6: **end while**
7: **return** $D$

A cada pas s'escull un node per afegir a la solució a mig construir actual segons una funció greedy i funció voraç que avalua moltes opcions i escull la millor. Després s'afegeix el node a la solució i es fan els canvis necessaris per mantenir les estructures de dades necessàries actualitzades. Totes les implementacions presentades a continuació seguiran aquest esquema, per tant, només es mostrarà les funcions de funcioGreedy(v) i actualitzarDades(v) així com les estructures de dades utilitzades (a no ser que s'especifiqui el contrari).

## 3.1 Notació Algorismes Golafres

- N: Nombre de vèrtexs en el graf.
- M: Nombre d'arestes en el graf.
- D: Conjunt de vèrtexs pertanyents a la solució.
- h(v): Nombre de veïns de $v$.
- minNND(v): Nombre de veïns inicialment necessaris per estar satisfet: $\lceil h(v)/2 \rceil$
- NND(v): Nombre de nodes veïns pertanyents a D, és a dir, $h(v) \cap D$
- sat(v): Indica si el vèrtex v està satisfet, és a dir, si $NND(v) \geq minNND(v)$
- NNUnsat(v): Indica el nombre de nodes veïns de v que no estan satisfets.
- Les referències a les comprovacions de si un graf és PIDS o minimal, són algorismes explicats en l'apartat d'algorismes de comprovació en l'apartat anterior.

## 3.2 Golafre Ingenu

A continuació es mostra la funció greedy al Algorithm 4 i la funció d'actualitzar dades al Algorithm 5 del primer algorisme greedy que s'ha implementat. La idea principal és escollir a cada pas el vèrtex amb més nodes no satisfets i els afegeix al conjunt de nodes dominants. Es considera ingenu per la idea senzilla que controla l'algorisme i manca d'estructures de dades complexes que accelerin aquest algorisme.
En la funció greedy (Algorithm 4) es conta el nombre de vèrtexs tals que són adjacents del vèrtex $v$ i que el seu nombre de veïns que pertanyen a $D$ sigui major o igual que la meitat del seu nombre de veïns, és a dir comptar els veïns de $v$ que compleixin $NND(v) < minNND(v)$.
En la funció actualitzar dades (Algorithm 5) es suma 1 per cada veí $w$ al vector $NND(w)$. Això assegura que el vector $NND$ sempre estarà actualitzat.
Es pot veure que és correcte perquè a cada iteració s'afegeix un vèrtex a la solució i només surt del bucle

quan la solució forma un PIDS, que en cas pitjor serà quan tots els vèrtexs formin part de la solució. S'ha de notar que no s'ha utilitzat cap classe de preprocessament ni poda de vèrtex, ja que no s'adapta bé a la definició d'un algorisme golafre, però en cas de voler implementar qualsevol, s'ha de notar que les solucions millorarien.

**Algorithm 4** Funció Greedy pel golafre ingenu

**Input:** Vertex $v$ per avaluar
**Output:** Nombre de veïns sense influencia positiva del vertex $v$

1: $count \leftarrow 0$
2: **for** cada veí $w$ de $v$ **do**
3: **if not** $sat(w)$ **then**
4: $count \leftarrow count + 1$
5: **end if**
6: **end for**
7: **return** *count*

**Algorithm 5** Funció Actualitzar Dades pel golafre ingenu

**Input:** Vertex $v$ que s'ha afegit a la solució
**Output:** $\emptyset$

1: **for** cada veí $w$ de $v$ **do**
2: $NND(w) \leftarrow NND(w) + 1$
3: **end for**

### 3.2.1 Anàlisi dels Costos

Per com es planteja l'algorsime golafre genèric (Algorithm 3), el while (línies 2-6) recorrerà com a màxim $N$ del graf, ja que a cada iteració s'afegeix un vèrtex. A més a més, trobar el màxim (línia 3) afegirà un recorregut per tots els nodes a cada iteració, és a dir $\mathcal{O}(N * N)$.
La funció greedy troba el màxim node segons el paràmetre explicat anteriorment, per tant, haurà de recórrer $N$. Per cada iteració s'hauran de buscar tots els nodes adjacents, és a dir, com a màxim $M$.

La funció d'actualitzar dades recorre totes les arestes del vèrtex seleccionat, per tant, recorre com a màxim $M$. En total es faran $\mathcal{O}(N^2 * (N * M + M))$ operacions bàsiques com sumes, restes i assignacions a vectors que totes tenen temps constant. En conclusió, té una complexitat temporal de $\mathcal{O}(N^3 * M)$.

Pel que fa al cost espacial, s'han d'emmagatzemar $N$ nodes i $M$ arestes per la representació. A més a més d'un vector extra per l'emmagatzematge del vector $NND$. En total té un cost espacial de $\mathcal{O}(N+M)$.

## 3.3 Golafre Minimal

Aquest algorisme golafre es basa en un estat inicial on tots els vèrtexs pertanyen a $D$ i a cada iteració es retira un vèrtex que no sigui necessari perquè el graf sigui d'influència positiva (PIDS). Per aquest algorisme utilitzarem l'esquema presentat a continuació, que a part de la diferència anterior, només avalua un node per iteració.

**Algorithm 6** Algorsime golafre minimal

**Input:** Graf $G$ no dirigit i connex
**Output:** PIDS (Conjunt dominant d'influència positiva)

1: $D \leftarrow G$
2: **while** D is not minimal **do**
3: $v* \leftarrow funcioGreedy()$
4: $D \leftarrow D - \{v*\}$
5: $actualitzarDades(v)$
6: **end while**
7: **return** $D$

Tot seguit es mostra la funció greedy per l'algorisme golafre minimal al Algorithm 7. La idea central és buscar un vèrtex de la solució actual tal que en cas de ser eliminat de la solució, el graf resultants continuaria sent PIDS. Per trobar-lo, es mira cada node (linies 3-8) i es comprova per cada vèrtex $w$ adjacent si amb un vèrtex veí dominant menys continuaria estant satisfets (línies 4-7). En cas que no hi hagi cap $w$ que el necessiti (linies 9-11), el vèrtex $v$ serà eliminat.

En la funció d'actualitzar dades de l'algorisme golafre minimal al Algorithm 8, simplement es recorre cada node adjacent al vèrtex $v$ eliminat i es resta 1 per cada veí $w$ al vector $NND(w)$ (línia 2). Això assegura que el vector $NND$ sempre estarà actualitzat.

Es pot veure que és correcte perquè a cada iteració s'elimina un vèrtex de la solució i només surt del bucle quan no troba cap vèrtex que pugui ser extret sense que la solució deixi de ser PIDS, per tant, és correcte.

**Algorithm 7** Funció Greedy pel golafre minimal

**Input:** $\emptyset$
**Output:** Vèrtex que pot ser eliminat de $D$ i que $G$ sigui sent PIDS. $\emptyset$ en cas contrari.

```
1: for cada vèrtex v ∈ D do
2:    esUtil ← False
3:    for cada vèrtex w adjecent a v do
4:       esUtil ← NND[w] − 1 < minNND(h(w))
5:       if esUtil then
6:          break
7:       end if
8:    end for
9:    if not esUtil then
10:      return v
11:   end if
12: end for
13: return ∅
```

**Algorithm 8** Funció Actualitzar Dades pel golafre minimal

**Input:** Vertex $v$ que s'ha tret de la solució
**Output:** $\emptyset$

```
1: for cada veí w de v do
2:    NND(w) ← NND(w) − 1
3: end for
```

### 3.3.1 Anàlisi dels Costos

Per com es planteja l'algorisme golafre minimal (Algorithm 6), el while (línies 2-6) recorrerà com a màxim $N$ del graf, ja que a cada iteració s'elimina un vèrtex. Encara que no es podran eliminar tots els vèrtexs perquè un PIDS diferent de $\emptyset$ sempre haurà de tenir vèrtexs dominants, no es pot assegurar un límit superior inferior a $N$
La funció greedy troba un node segons el paràmetre explicat anteriorment, per tant, haurà de recórrer $N$. Per cada iteració s'hauran de buscar tots els nodes adjacents, és a dir, com a màxim $M$.

La funció d'actualitzar dades recorre totes les aristes del vèrtex seleccionat, per tant, recorre com a màxim $M$. En total es faran $\mathcal{O}(N * (N * M + M))$ operacions bàsiques com sumes, restes i assignacions a vectors que totes tenen temps constant. En conclusió, té una complexitat temporal de $\mathcal{O}(N^2 * M)$.

Pel que fa al cost espacial, s'han d'emmagatzemar $N$ nodes i $M$ arestes per la representació. A més a més d'un vector extra per l'emmagatzematge del vector $NND$. En total té un cost espacial de $\mathcal{O}(N+M)$.

## 3.4 Golafre Alternatiu

Aquest algorisme golafre alternatiu o com s'ha acabat anomenant "Golafre Alt", es basa en la mateixa idea que el primer algorisme golafre presentat: escollir a cada pas el vèrtex amb més nodes adjacents no satisfets, amb una diferència important: un component d'aproximació. En comptes de recórrer tots els vèrtexs per comprovar quin és el màxim segons l'estat actual, s'ofereix un paràmetre anomenat "profunditat" que permet controlar quina és la distància d'actualització de les dades respecte al vèrtex que s'ha modificat. És a dir, que depenent d'aquest paràmetre les dades emmagatzemades es mantindran més o menys actualitzades, provocant un impacte directe sobre la qualitat de la solució final i el temps emprat en calcular-la.
L'algorisme per la funció Greedy per golafre alternatiu (Algorithm 12) simplement retorna el node amb major nombre de nodes $w$ adjacents tals que $\not sat(w)$, és a dir, el vèrtex $v$ amb major $NNUnsat(v)$.
L'algorisme d'actualització de dades (Algorithm 10), actualitza el vector $NND$ a les posicions dels veïns del vèrtex afegit a la solució (línies 1-3) i fa el recalcul de la insatisfacció dels vèrtexs (Algoritm 11) des del vèrtex $v$ i amb una profunditat màxima, que és el que després ens permetrà després accedir al millor vèrtex segons aquest criteri (línia 4).
La funció de recalcular la insatisfacció és una manera recursiva d'implementar aquesta propagació d'actualitzacions del càlcul de la insatisfacció a partir del vèrtex afegit. El cas base és no fer res (línies 1-3), i el pas recursiu és calcular la insatisfacció pròpia del node i repetir per tots els seus nodes adjacents que no hagin sigut calculats prèviament (línies 4-9).
Es pot veure que és correcte perquè a cada iteració s'afegeix un vèrtex a la solució i només surt del bucle quan la solució forma un PIDS, que en cas pitjor serà quan tots els vèrtexs formin part de la solució.

**Algorithm 9** Funció Greedy pel golafre alternatiu

**Input:** $\emptyset$
**Output:** Vèrtex amb més nodes veïns no satisfets.

1: **return** $argmax_{\forall v \notin S}\{NNUnsat(v))\}$

**Algorithm 10** Funció Actualitzar Dades pel golafre alternatiu

**Input:** Vertex $v$ que s'ha tret de la solució
**Output:** $\emptyset$

1: **for** cada veí $w$ de $v$ **do**
2: $\quad NND(w) \leftarrow NND(w) + 1$
3: **end for**
4: $recalculUnsat(v, profunditat)$

**Algorithm 11** Funció actualitzar insatisfacció pel golafre alternatiu

**Input:** Vertex $v$ que s'ha actulatizat i profunditat a la que actualitzar
**Output:** $\emptyset$

1: **if** $profunditat = 0$ **then**
2: $\quad$ **return**
3: **end if**
4: **for** cada veí $w$ de $v$ **do**
5: $\quad$ **if not** $calculat(w)$ **then**
6: $\quad\quad Calcular\ Unsat(v)$
7: $\quad\quad recalculUnsat(w, profunditat - 1)$
8: $\quad$ **end if**
9: **end for**

### 3.4.1 Anàlisi dels Costos

Per com es planteja l'algorisme golafre genèric (Algorithm 3) , el while (línies 2-6) recorrerà com a màxim $N$ del graf, ja que a cada iteració s'afegeix un vèrtex, $\mathcal{O}(N)$.
La funció greedy troba el màxim node segons el paràmetre explicat anteriorment, però amb un precalcul

explicat a la funció d'actualitzar dades, podem accedir al màxim en temps constant $\mathcal{O}(1)$.

La funció d'actualitzar dades recorre totes les aristes del vèrtex seleccionat, per tant, recorre com a màxim $M$. Després crida la funció de $recalculUnsat(v, profunditat)$ que s'analitzarà el seu cost amb el teorema mestre.
La funció d'actualitzar la insatisfacció, fa $\mathcal{O}(M)$ crides cada iteració on es resta un al valor de la profunditat. La funció $Calcular\ Unsat(v)$ recorre les arestes del vèrtex $(M)$. La recurrència es podria escriure la següent figura. On $p$ és la profunditat i $T(p)$ el cost de la recurrència en funció de $p$. Com que el factor de ramificació $(M)$, és major que 1, el cost de la recurrència serà $\mathcal{O}(p * M^p)$.

$$T(p) = \begin{cases} \text{M*T(p-1)+M}, & \text{if p} \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

En total, tindrà cost de $\mathcal{O}(N * (p * M^p))$, és a dir, $\mathcal{O}(p * N * M^p))$, que amb una profunditat baixa $(p = [1, 2])$ es podria considerar $\mathcal{O}(N * M)$.

Pel que fa al cost espaial, s'han d'emmagatzemar $N$ nodes i $M$ arestes per la representació. A més a més s'han implementat les següents estructures de dades:

- Vector d'enters de mida $N$ per l'emmagatzematge del vector $NND$. $\mathcal{O}(N)$
- Vector de booleans de mida $N$ per l'emmagatzematge de la satisfacció dels vèrtexs. $\mathcal{O}(N)$
- Vector de "unordered_sets" d'enters on cada posició del vector representa el nombre de nodes no satisfets al voltant d'un vèrtex i dins de cada conjunt hi ha punters als vèrtexs que tenen la puntuació indicada per la posició del vector. Hi haurà tantes posicions en el vector com arestes màximes que hi ha en un node. $\mathcal{O}(N + M)$
- Vector d'enters de mida $N$ que indica la posició en el vector anterior de cada vèrtex. $\mathcal{O}(N)$
- Vector de booleans de mida $N$ que indica si ja s'ha calculat la insatisfacció de cada vèrtex. $\mathcal{O}(N)$

En total, no se supera el cost espacial $\mathcal{O}(N + M)$ de la representació bàsica.

## 3.5 Golafre Estocàstic

Aquest algorisme és idèntic a l'algorisme anterior però amb un component estocàstic per generar solucions aleatòries. En l'algorisme alternatiu s'agafa el millor vèrtex, però hi ha moltes situacions en què dos vèrtexs diferents tenen les mateixes valoracions, és aquí quan s'afegeix el component aleatori i s'escull qualsevol vèrtex amb la mateixa valoració màxima. És necessari per a la generació d'estats inicials en algorismes que necessiten aleatorietat en la generació d'estats inicials, com és el cas de l'algorisme metaheurístic implementat més endavant.
Les funcions d'actualitzar dades (Algorithm 10) i actualitzar les insatisfaccions (Algorithm 11) són idèntiques, i, per tant, no és necessari tornar-les a mostrar.
Es pot veure que és correcte perquè a cada iteració s'afegeix un vèrtex a la solució i només surt del bucle quan la solució forma un PIDS, que en cas pitjor serà quan tots els vèrtexs formin part de la solució.

**Algorithm 12** Funció Greedy pel golafre Estocàstic

**Input:** $\emptyset$
**Output:** Vèrtex amb més nodes veïns no satisfets.

1: $maxVal \leftarrow max_{\forall v \notin S}\{NNUnsat(v))\}$
2: **return** $randomElement_{\forall v \notin S}(v|NNUnsat(v) = maxVal)$

### 3.5.1 Anàlisi dels Costos

El cost temporal de l'algorisme és idèntic, ja que per generar nombres aleatoris es té cost temporal constant $\mathcal{O}(1)$ i agafar un element d'un unordered_set, mitjançant un iterador d'accés aleatori també té cost constant $\mathcal{O}(1)$. En conclusió, tindrà cost de $\mathcal{O}(N * (p * M^p))$, és a dir, $\mathcal{O}(p * N * M^p))$, que amb una profunditat baixa $(p = [1, 2])$ es podria considerar $\mathcal{O}(N * M)$.

El cost espacial és també igual, ja que no s'han afegit ni tret cap mena d'estructura de dades, per tant, $\mathcal{O}(N + M)$.

## 3.6 Golafre de Pan

La idea d'aquest algorisme tan conegut és recórrer cada vèrtex $v_i$ assignar el nombre de nodes mínim per assegurar que $v_i$ està satisfet ($\Delta_D(v)$). Per aconseguir els millors resultats, s'ordenen els nodes segons el seu nombre de nodes adjacents ascendentment (línia 3) s'assignen els $\Delta_D(v)$ nodes dominants pel vèrtex $v_i$ (línies 7-9) i s'actualitza $\Delta_D(v)$ i $NNUnsat(v)$ si és necessari (línies 10 - 17).
Es pot veure que és correcte perquè al final de l'algorisme, ha passat per tots els nodes i ha assegurat que és satisfet, cosa que fa que sigui PIDS.

**Algorithm 13** Algorisme Golafre de Pan

**Input:** Graf G = ($V$, $E$) connex i no dirigit
**Output:** PIDS pel graf G

1: $D \leftarrow \emptyset$
2: Calcular $\Delta_D(v)$ i $NNUnsat(v)$ per cada node $v \in V$
3: Ordenar $V = \{v_1, v_2, ..., v_N\}$ per $h(v_i)$ en ordre ascendent
4: **for** $i = 1$ **to** N **do**
5:  **if** $\Delta_D(v_i) > 0$ **then**
6:  $R \leftarrow \Delta_D(v_i)$
7:  **for** $j = 1$ **to** R **do**
8:  $u \leftarrow arg\,max_{w \in veïns_{\bar{D}(v_i)}\{NNDUnsat(w)\}}$
9:  $D \leftarrow D \cup \{u\}$
10:  **for all** $x \in veins(u)$ **such that** $\Delta_D(x) > 0$ **do**
11:  $\Delta_D(x) \leftarrow \Delta_D(x) - 1$
12:  **if** $\Delta_D(x) = 0$ **then**
13:  **for all** $y \in veins(x)$ **do**
14:  $NNDUnsat(y) \leftarrow NNDUnsat(y) - 1$
15:  **end for**
16:  **end if**
17:  **end for**
18:  **end for**
19:  **end if**
20: **end for**
21: **return** $D$

### 3.6.1 Anàlisi dels Costos

De manera intuïtiva, la línia 2 és $\mathcal{O}(n)$ i la línia 3 és $\mathcal{O}(nlgn)$. Per les línies 4-14 es pot veure que la mida de D és com a màxim $N$, $R$ serà com a màxim $M$. A més a més, les línies 10-11 seran executades quan $u$ sigui afegit a D, i el tems d'execució d'aquestes línies és proporcional a $h(u)$. Com que cada node $u$ serà afegit com a mínim un cop, el temps total per les línies 10-11 és $\mathcal{O}(M)$. Pel mateix raonament, les línies 13-14 només s'executaran quan $\Delta_D(x) = 0$ i el temps d'execució d'aquestes línies és proporcional a $h(x)$. Com que per cada node $x$, $\Delta_D(x)$ arribarà a 0 com a màxim un cop, el temps total de les línies 13-14 és $\mathcal{O}(M)$. En conclusió, la complexitat espacial del Algorithm 13 és de $\mathcal{O}(Nlgn+M)$.

Pel cost espacial, la part de la representació bàsica del graf i la seva solució que ocupa $\mathcal{O}(N + M)$. I despès s'ha implementat un vector de parelles per fer la conversió de la representació antiga a la representació ordenada que ocupa espai $\mathcal{O}(N)$. En total, ocupa $\mathcal{O}(N + M)$.

## 3.7 Experiments i Resultats

Experiments: En la següent taula es poden veure uns resultats dels algorismes implementats on es mostra la mida del conjunt de vèrtexs dominants a la columna val i el temps emprat per trobar la solució a la columna t. L'algorisme Golafre Alternatiu s'ha fet amb un valor de profunditat de 5, perquè és un bon equilibri entre temps i la qualitat de la solució. El Golafre Estocàstic fet 10 cops, per assegurar una mitjana no molt allunyada de la realitat. Aquests experiments han sigut fets en el mateix ordinador, però en un ordinador diferent dels resultats a l'apartat de conclusions, per tant, els temps són comparables entre els algorismes golafres però no amb els temps dels altres algorismes.

| Resultats | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Benchmark | Ingenu | | Minimal | | Alternatiu | | Estocàstic | | Pan | |
| nº elements a PIDS i s | Val | t | Val | t | Val | t | Val(avg) | t(avg) | Val | t |
| soc-gplus | 8341 | 1.219 | 11788 | 0.701 | 8351 | 2.481 | 8335.9 | 2.651 | 9997 | 0.015 |
| graph_CA-CondMat | 9847 | 8.202 | 13084 | 2.028 | 9930 | 1.238 | 9926.0 | 1.405 | 9967 | 0.005 |
| graph_CA-AstroPh | 7083 | 28.76 | 10017 | 2.722 | 7122 | 14.40 | 7121.3 | 14.40 | 7241 | 0.015 |
| graph_CA-HepPh | 4880 | 4.036 | 6499 | 0.509 | 4919 | 2.591 | 4911.4 | 2.613 | 4952 | 0.005 |
| graph_actors_dat | 3196 | 4.162 | 5975 | 0.551 | 3260 | 2.668 | 3261.6 | 2.821 | 3380 | 0.009 |
| ego-facebook | 1978 | 1.269 | 2049 | 0.084 | 1994 | 0.469 | 1994.3 | 0.470 | 1980 | 0.004 |
| socfb-Brandeis99 | 1533 | 3.060 | 1914 | 0.116 | 1533 | 6.722 | 1534.6 | 6.506 | 1700 | 0.008 |
| socfb-Mich67 | 1491 | 1.102 | 1859 | 0.062 | 1488 | 2.201 | 1481.1 | 2.181 | 1641 | 0.003 |
| graph_jazz | 81 | 0.000 | 99 | 0.00 | 84 | 0.001 | 84.0 | 0.001 | 97 | 0.000 |
| graph_football | 68 | 0.000 | 74 | 0.00 | 70 | 0.000 | 69.0 | 0.000 | 76 | 0.000 |

Es pot veure que els millors resultats són oferts per l'algorisme golafre ingenu i que el programa més ràpid és l'algorisme de Pan. S'ha decidit escollir l'algorisme ingenu per la comparació entre algorismes pels seus bons resultats.

# 4 Cerca Local

## 4.1 Principis de la Cerca Local

"Descriu qualsevol algorisme que explora l'espai de possibles solucions de manera seqüencial, movent-se cada pas des d'una solució actual a una de propera". Així és com es descriuen els algorismes de cerca local en el llibre *Algorithm Design* [9]. Tot i que la descripció donada és molt general, captura la idea d'aquests algorismes.

Aquest tipus d'algorismes són utilitzats per resoldre problemes d'optimització de complexitat elevada i difícils de computar, basats també en aplicació d'heurístiques per la minimització o maximització dels paràmetres a optimitzar. Tal com es comentava en la citació anterior i de manera genèrica, els algorismes es mouen entre les diferents solucions, espai o conjunt que s'ha anomentat espai de solucions, partint d'una solució inicial i intentant millorar-la trobant solucions semblants. Aquests canvis seran aplicats per operadors o mutacions definides i l'aplicació d'aquests vindrà determinada per heurístics o altres motors d'optimització.

Per tant, de manera simplificada, però general, qualsevol algorisme de cerca local té els següents elements [8]:

- Espai de cerca constituït per totes les solucions del problema.
- Conjunt de solucions factibles, subset de l'espai de cerca.
- Relació de veïnatge, definida entre dues solucions i que es basa en la similitud dels elements d'aquestes.
- Conjunt finits d'estats de memòria.
- Funció inicialitzadora, que proporciona l'estat/solució inicial i sol contenir una component estocàstica.
- Funció d'*Step*, utilitzada per modificar un estat i obtenir-ne un de veí i sol contenir una component estocàstica.
- Predicat de terminació, utilitzada per decidir quan la cerca s'ha de terminar.

Per tant, donats aquests elements, els problemes que seran bons candidats per aplicar-hi algorismes de cerca local seran aquells que tinguin una fàcil representació d'estat, un cost asimptòtic baix per discernir si un estat és solució o no ho és, i que també mostrin facilitat a l'hora de modificar solucions generant-ne d'altres de semblants.

Per resoldre problemes de combinatòria o optimització computacionalment difícils (NP-Difícils) però, hi ha 3 grans mètodes principals que es poden aplicar tal com menciona [8]: La possibilitat de trobar un subproblema d'aquest que és resoluble de manera eficient, utilitzar algorismes d'aproximació eficient i per últim utilitzar algorismes estocàstics, on s'incluen algorismes de cerca local. Per tant, ja que hi ha alternatives a utilitzar algorismes de cerca local quins beneficis i inconvenients té?

Tal com es menciona al llibre d'Algorithm Desig [9], l'avantatge que tenen és que la definició generalitzada d'aquests permet la implementació de manera relativament senzilla d'aquests algorismes sobre quasi qualsevol problema de computació difícil, a diferència d'altres algorismes d'aproximació com podrien ser algorismes golafres. Per contra, però, no és gens fàcil demostrar la qualitat d'un algorisme de cerca local, el que provoca que sigui difícil discernir algorismes bons dels dolents.

Mencionar també la rellevància de l'aleatorietat en aquest camp de l'algorísmia. És un element interessant que els algorismes de cerca local siguin estocàstics i no deterministes, d'aquesta manera no sempre s'explora el mateix camí de solucions i solució final sinó que l'exploració de l'espai de solucions, tot i que regida per probabilitats i normes, és menys previsible i ajuda a sortir de mínims o màxims locals, i per tant ajuda a obtenir una solució més propera a l'òptima.

## 4.2 Cerca Local aplicada a PIDS

Abans d'aplicar algorismes de cerca local al problema de PIDS, ja es poden fer observacions que importants que afectaran la implementació.

En primer lloc, es pot determinar que existeixen $2^N$ possibles configuracions del problema donats $N$ nodes, no totes solucions, però que, per tant, l'espai de solucions té mida asimptòtica de $O(2^N)$. Això es pot veure fàcilment, ja que cada node de $N$ pot ser o no ser de PIDS, és a dir que cada node té 2 estats.

També és important veure que els estats són fàcilment representables, ja que si s'emmagatzema el graf per separat, només és necessari aguardar un booleà per cada node indicant la seva pertinença a dintre el PIDS.

Un punt negatiu que es pot observar però és que els operadors que es podran aplicar segurament no seran capaços d'allunyar-nos de mínims o màxims locals. Això és així per la naturalesa del problema, que pot tenir diferents solucions properes a l'òptim, però amb elements al PIDS molt diferents, és a dir una relació de veïnatge feble. Per això és molt rellevant que hi hagi factors estocàstics per explorar més bé tot l'espai de solucions i tenir més possibilitats d'acostar-se a l'òptim.

## 4.3 Algorismes Utilitzats

Per resoldre el problema MIPDS s'ha implementat dos algorismes de cerca local: Hill Climbing i Simulated Annealing.

### 4.3.1 Hill Climbing

L'algorisme Hill Climbing es basa en una cerca local determinista, on es generen tots els possibles veïns de l'estat actual i s'escull el millor d'aquest, és a dir amb un valor heurístic més elevat, a cada pas. Tot i això, la funció generadora per l'estat inicial pot complementar l'algorisme amb una component estocàstica.

Es pot observar el pseudocodi de la versió implementada en aquest projecte en la figura següent:

**Algorithm 14** Algorisme de Hill Climbing

```
1:  Actual ← EstatInicial
2:  Millora ← false
3:  while Millora do
4:      Best ← Actual
5:      for totsElsVeins do
6:          Nou ← generarSuccessor(Actual)
7:          if heuristic(Nou) > heuristic(Best) then
8:              Best ← Nou
9:              Millora ← true
10:         end if
11:     end for
12:     if Millora then
13:         Actual ← Best
14:     end if
15: end while
```

Aquesta versió respecta d'altres possibles, es pot diferenciar en el fet que no es guarden tots els veïns de l'estat actual per posteriorment ordenar-los i escollir el millor, sinó que per cada veí generat només es guarda el millor fins al moment. D'aquesta manera s'estalvia un cost espacial considerable.

Pel que fa a costos, a nivell espacial en la implementació escollida no té importància significativa, ja que tindria cost asimptòtic del cost d'un estat. Tot i això, el cost temporal és un dels desavantatges de l'algorisme. Això és degut al fet que per cada iteració, es generen tots els possibles veïns amb els diferents operadors utilitzats. Considerant que el màxim d'iteracions és la mida de l'espai de solucions, el cost asimptòtic és $O(2^N)$ pel cost de l'operador amb cost més elevat.

També és interessant remarcar que Hill Climbing és un algorisme que té un comportament reminiscent al d'un golafre, ja que a cada iteració de la cerca, es queda amb el millor estat nou trobat, encara que això faci que a la llarga es quedi estancat en un mínim/màxim local.

### 4.3.2 Simulated Annealing

L'algorisme Simulated Annealing utilitza la cerca local simulant el procés de recuita de metalls, procediment que consisteix a escalfar i refredar lentament els materials. Això en l'algorisme pren forma com a millora i empitjorament de la solució actual que s'obté. Això s'aconsegueix tenint sempre una solució actual i per cada iteració generant només una solució veïna aleatòria que substituirà l'actual en cas de tenir una heurística millor o, en cas de ser pitjor segons una funció de probabilitat $e^{\Delta E/F(T)}$ on $\Delta E$ és la diferència de l'heurístic entre la solució actual i la nova generada i $F(T)$ és una funció respecte al paràmetre temperatura.

Com que les solucions veïnes generades són aleatòries, l'algorisme és estocàstic, volent simular l'aleatorietat en la col·locació dels àtoms en el procés de refredament dels metalls.

El pseudocodi es pot observar en la següent figura:

**Algorithm 15** Algorisme de Simulated Annealing

```
1: Actual ← EstatInicial
2: while Temp > 0 do
3:    for iterXTemp > 0 do
4:       Nou ← generarSuccessorAleatori(Actual)
5:       ΔE ← heuristic(Nou) − heuristic(Actual)
6:       if ΔE > 0 then
7:          Actual ← Nou
8:       else
9:          amb probabilitat de e^(δE / (k·e^(−λ·T))) : Actual ← Nou
10:      end if
11:      iterXTemp ← iterXTemp − 1
12:   end for
13:   Temp ← Temp − 1
14: end while
```

Com a paràmetres importants en la implementació escollida tenim els següents valors:

- $Temp$ - Simula la temperatura en el procés de recuita de metalls de la via real. En l'algorisme actua com a indicador de les iteracions i afecta la probabilitat d'acceptació dels canvis que empitjoren la solució. Decrement en un per cada conjunt d'iteracions realitzades.
- $iterXTemp$ - Indica el nombre d'iteracions que es realitzaran (nous estats veïns aleatoris generats) per cada valor de temperatura. Això garanteix que durant aquestes iteracions la probabilitat d'acceptar solucions pitjors sigui igual, ja que $Temp$ no canvia i la resta de variables són constants. També es pot veure que el nombre total d'iteracions serà $iterT = Temp \cdot iterXTemp$.
- $k$ - Constant que permet modificar el comportament de $F(Temp)$. Augmentar $k$ implica augmentar el nombre d'iteracions on s'accepten solucions pitjors.
- $\lambda$ - Constant que permet modificar el comportament de $F(Temp)$. Augmentar $\lambda$ implica disminuir el nombre d'iteracions que accepten solucions pitjors i també la velocitat a la qual aquesta probabilitat disminueix en funció a $Temp$.

Sobre el cost de l'algorisme es pot veure que no és molt elevat, ja que a cada iteració només es genera un nou estat. El cost espacial és asimptòticament el cost d'un estat. El cost temporal és $O(Temp \cdot iterXTemp)$ pel temps de generar un nou estat i fer les comprovacions necessàries, que en ser una generació aleatòria no té un cost en funció de $N$.

## 4.4 Implementació

La solució inicial, operadors i heurístiques que s'han implementat s'han usat per als dos algorismes.

### 4.4.1 Representació de l'Estat

Per la representació dels estats del problema s'utilitzen les estructures de dades mencionades a l'apartat 2.1. Per simplificar el codi s'utilitza també un *struct* de C++ per guardar tota la informació de l'estat:

```
typedef struct{
vector D = vector(N, false); //elements de D
vector NND = vector(N,0); //Veins de D de cada node
int ND = 0; //nombre de nodes de D
double h = 0; //valor de l'heuristic
} estat;
```

### 4.4.2 Funció Inicialitzadora

S'han implementat 3 funcions inicialitzadores diferents:

1. D ple: Es comença amb tots els nodes al set PIDS, per tant, l'estat inicial és solució i el cost de computació és constant.

2. Aleatòria: Es parteix d'un conjunt buit i es van afegint nodes fins que el conjunt sigui PIDS. El cost és $O(N^2 + M)$

3. Golafre Estocàstic: S'aprofita l'algorisme golafre comentat a l'apartat 3.5.

Totes les versions de la funció inicialitzen a part del PIDS, el vector NND per guardar els veins de cada node que pertanyen a D, i ND, el nombre d'elements de PIDS.

### 4.4.3 Operadors

S'han implementat 3 operadors diferents:

1. REMOVE(i): Elimina el vèrtex i del conjunt actual PIDS. És l'operador més intuïtiu, ja que ajuda a minimitzar la quantitat de nodes de PIDS. Té una condició d'aplicabilitat: que el vèrtex i estigui al conjunt PIDS. El cost de ramificació és de $O(N)$

2. ADD(i): Afegeix el vèrtex i al conjunt actual PIDS. Té una condició d'aplicabilitat: que el vèrtex i estigui al conjunt PIDS. El cost de ramificació és de $O(N)$

3. SWAP(i,j): És l'equivalent de realitzar un REMOVE(j) i un ADD(i) a la vegada. El nombre de vèrtex de PIDS es manté però canvien. Les condicions d'aplicabilitat són que el vèrtex i no sigui del PIDS i que el vèrtex j sí que hi sigui. El cost de ramificació és de $O(N^2)$.

L'espai de solucions es pot explorar completament utilitzant els dos operadors de REMOVE() i ADD(). L'operador de SWAP() s'afegeix per poder evitar limitacions d'haver d'aplicar seqüencialment els dos operadors anteriors, en casos on l'heurístic intenta minimitzar el nombre de nodes del conjunt PIDS (sobretot en el cas de Hill Climbing).

### 4.4.4 Heurística

Per tots els heurístics es descrten les no solucions retornant un valor negatiu considerable (es recorda que tant Hill Climbing com Simulated Annealing implementats funcionen maximitzant l'heurístic). Això ha resultat necessari per evitar que els algorismes retornin no solucions però que momentaniament es pugui sortir de l'espai de solucions i tornar-hi a entrar en el cas del Simulated Annealing.

S'han implementat 4 funcions heurístiques diferents:

1. H1: Es vol minimitzar la quantitat de vèrtex del conjunt PIDS actual.

2. H2: Es vol maximitzar la suma de les cardinalitats dels vèrtex del conjunt PIDS actual

3. H3: Es vol minimitzar la suma de les diferències per cada vèrtex del valor mínim de veïns que han de ser del conjunt PIDS i el nombre actual de veïns de PIDS (serà superior al mínim perquè sinó seria una no solució i es tracten de manera diferent com s'ha comentat abans)

## 4.5 Experimentació i Comparació dels Algorismes

### 4.5.1 Hill Climbing vs Simulated Annealing

Després d'implementar els dos algorismes es va observar el seu funcionament amb els diferents benchmarks amb valors de prova per les constants de Simulated Annealing i observant els diferents valors d'heurístics i solucions inicials.

Immediatament, es va poder comprovar que, com es podia veure amb l'anàlisi de costos dels dos algorismes, el Hill Climbing proposta té un problema greu amb el cost temporal. Això fa que no es puguin obtenir bons resultats amb qualsevol dels benchmarks amb nombre elevat de vèrtexs. Pel que fa als tests que es podien executar amb un temps raonable, no s'obtenien resultats propers als òptims. Això és degut al fet que la combinació d'heurístics i operadors no permetia desencallar aquestes situacions.

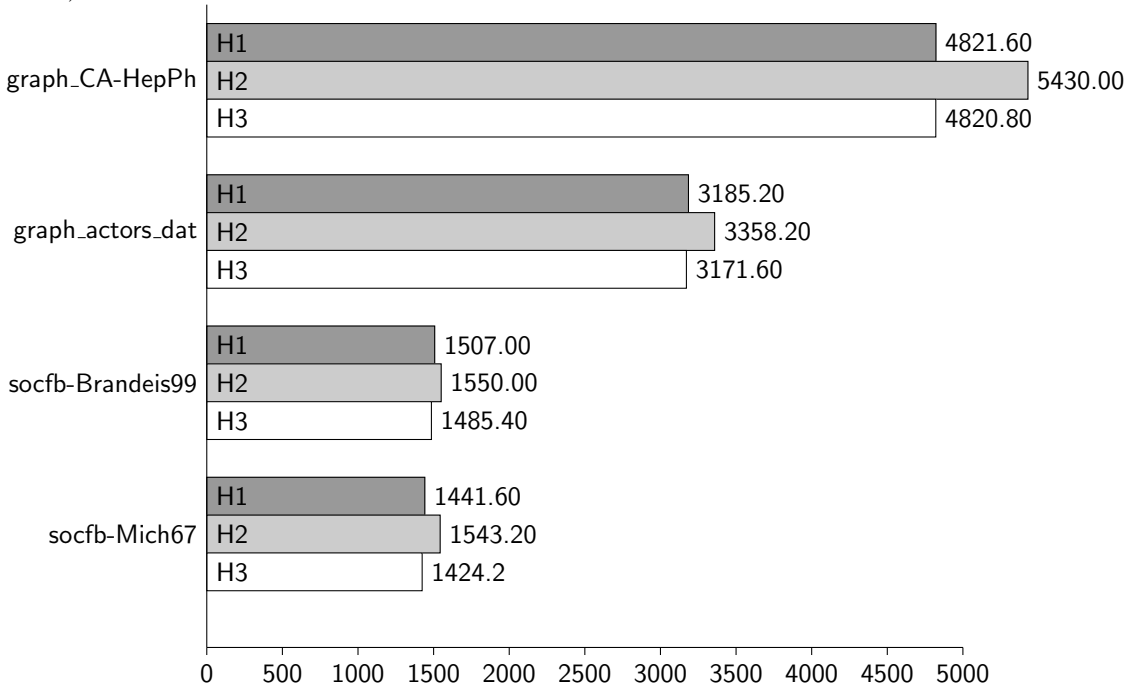
Simulated Anneling, en canvi, té un cost temporal correcte, tot i que no tan bo com el dels algorismes golafres, com es pot esperar per la seva naturalesa. Els resultats del Simulated Annealing també són considerablement bons, millorant els del greedy estocàstic quan es fa servir com a solució inicial.

És per això que per la resta d'experiments i resultats es decideix només treballar amb el Simulated Annealing proposat (s'utilitzarà també l'algorisme golafre estocàstic ja que és el que dona més bons resultats).

### 4.5.2 Escollir Heurístic

Per tal d'escollir el millor heurístic per utilitzar el Simulated Annealing, s'han fixat valors per defecte dels paràmetres $T$, $iT$, $k$ i $\lambda$, utilitzant l'algorisme golafre estocàstic per generar la solució. Per cada conjunt de paràmetres s'han fet 5 execucions per obtenir-ne les mitjanes dels valors, ja que l'algorisme no és determinista.

Els resultats obtinguts per diferents benchmarks són els següents (en nombre de vèrtexs al conjunt dominant):




Es pot comprovar que, encara que en alguns casos la diferència és petita, l'heurístic 3 explicat a l'apartat 4.4.4, sempre és millor que els altres dos. Es pot concloure doncs que aquest heurístic és el que genera més bons resultats de manera general i és el que es farà servir per a la resta d'experiments.

### 4.5.3 Escollir Paràmetres per Simulate Annealing

També s'ha realitzat un experiment per determinar els millors paràmetres $k$ i $\lambda$ per l'algorisme.

Es fixen els següents valors: 5 iteracions per execució, heurístic 3, $Temp = 10000$, $iterXTemp = 15$ i també es generarà la solució inicial mitjançant l'algorisme golafre estocàstic. S'utilitzen diferents benchmarks de mides variades per un millor anàlisis.

Amb els resultats obtinguts, que es troben a l'annex A, es pot veure que, tot i que la diferència és subtil, es pot triar $\lambda = 14$ i $k = 10$ per obtenir uns millors resultats. Es fixen aquests valors doncs per la resta d'execucions

# 5 Metaheurística

Els científics Kenneth Sörensen i Fred Glover defineixen la metaheurística com *"un marc algorísmic d'alt-nivell, independent d'un problema, que proporciona una sèrie de guíes o estratègies per desenvolupar algorismes d'optimització heurística"* [12]. Sovint, les estratègies metaheurístiques s'inspiren en fenòmens del món natural, com per exemple els algorismes Simulated Annealing, Optimització d'Eixams, Colònia de Formigues, Cerca Gravitacional, o Algorisme Genètic. Els algorismes metaheuristics no garanteixen trobar la solució òptima, i sovint busquen trobar una solució suficientment bona, però en certs problemes permeten arribar a aquesta solució bona en una fracció del temps d'altres tècniques més exhaustives.

## 5.1 Algorisme Genètic

En el cas del problema del MPIDS s'ha decidit implementar un Algorisme Genètic. Aquesta classe d'algorismes pretenen simular l'evolució de les espècies: els individus més aptes es reprodueixen i els seus descendents hereten característiques d'ambdós pares, amb una probabilitat que hi hagi una mutació. Aquests algorismes són un tipus d'algorismes de cerca local, és a dir, que exploren l'espai de solucions.

S'ha escollit l'Algorisme Genètic i no un altre estratègia metaheurística gràcies a la senzilla codificació binària que té el problema, com s'explicarà més endavant.

### 5.1.1 Requisits

Per realitzar un algorisme genètic cal complir els següents requisits:

- Codificar les solucions del problema. Generalment se sol fer com una cadena binària, la qual cosa ens permetrà combinar-les i mutar-les. Aquestes cadenes es diuen *Gens*, seguint la terminologia inspirada per la biologia.
- Disposar d'una funció que ens permeti avaluar la qualitat d'una solució. És equivalent a la funció heurística d'altres problemes de cerca local. A algorismes genètics se sol anomenar *Funció d'Adaptabilitat*.
- Establir operadors per moure's per l'estat de sol·lucions. Solen ser bastant estàndard, un operador d'*encreuament*, que representa la reproducció sexual de dos individus, i un operador de *mutació*, que representa les mutacions aleatòries a la naturalesa que poden donar avantatges evolutius.
- Decidir l'estratègia de selecció d'individus per a la reproducció. Això és important perquè hauria de valorar els individus amb major adaptabilitat, però alhora garantir la diversitat de la població, o altrament es pot convergir massa aviat i comprometre la qualitat de la nostra solució.
- Decidir com es generen la població de solucions inicials. Aquestes haurien de ser variades per permetre explorar l'espai de solucions, i també és favorable que s'apropin a una solució òptima.

### 5.1.2 Funcionament de l'Algorisme

El funcionament de l'algorisme és prou simple:

- Es genera la població de solucions inicials, de mida N.
- Durant un nombre determinat de generacions, o fins que l'adaptabilitat de la població no millora:
  - S'escullen N individus de la població seguint la funció de selecció i s'aparellen.
    - Amb una probabilitat determinada, s'encreuen i es generen dos fills. Altrament els pares passen a la nova generació.
    - Amb una probabilitat determinada, es muta un fill.
  - Se substitueix la població actual per la nova generació.

Per a una descripció més detallada de l'algorisme en pseudocodi el llibre *Artificial Intelligence: A Modern Approach* ens proporciona el següent pseudocodi:

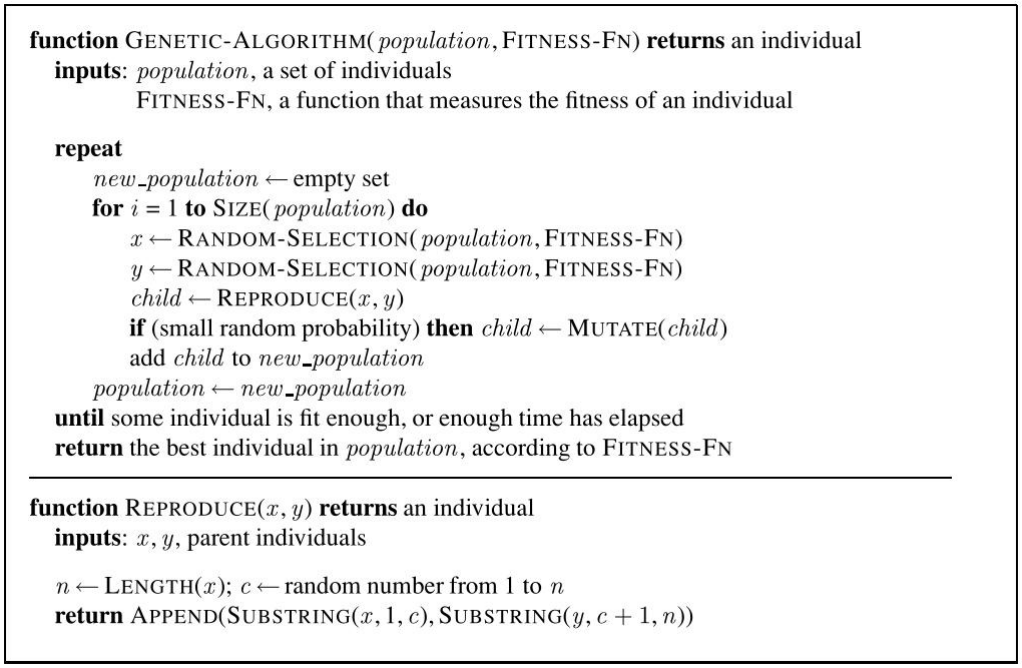



Figura 2: Pseudocodi de l'Algorisme Genètic [11]

### 5.1.3 Algorisme Genètic en el problema MPIDS

**Codificació dels Gens** La codificació dels Gens realitzada és prou senzilla: s'emmagatzema un vector de Booleans $Gen[]$ on cada element representa un vèrtex del graf: si $Gen[v]$ és 1, aleshores el vèrtex $v$ forma part del *Conjunt Dominant*. Aquesta codificació ens permet representar tots els conjunts possibles, i no ens permet representar cap no-conjunt (és a dir, en altres problemes genètics una codificació mal plantejada pot representar un estat impossible). És per aquest motiu que s'han escollit aquesta meta-heurística. Però, un problema que pot tenir és que és possible representar no-solucions (per exemple, un vèrtex té menys del 50% dels seus veïns al *Conjunt Dominant*). Això, però, es pot solucionar penalitzant les no-solucions a la *Funció d'Adaptabilitat*.

**Funció d'Adaptabilitat** La funció d'adaptabilitat ens mesura la qualitat d'una solució. Es vol una funció que penalitzi tenir més veïns dins del *Conjunt Dominant* que el 50% necessari, donat que això podria indicar que la solució no és mínima, però alhora s'han de de penalitzar molt més fortament tenir menys del 50% de veïns al *Conjunt Dominant*, donat que això voldria dir que s'ha sortit de l'espai de solucions. Així doncs, s'ha pensat la següent funció d'adaptabilitat:

$$f(V,E) = \sum_{v=v_0}^{v_n} f_v(v)$$

$$f_v(v) = \begin{cases} 0 & \text{si } a_D(v) = \lceil g(v)/2 \rceil \\ a_D(v) - \lceil g(v)/2 \rceil & \text{si } a_D(v) > \lceil g(v)/2 \rceil \\ (a_D(v) - \lceil g(v)/2 \rceil)^2 & \text{si } a_D(v) < \lceil g(v)/2 \rceil \end{cases}$$

**Operadors** Per l'operador d'encreuament, un cop s'ha seleccionat una parella d'individus i han superat la probabilitat d'encreuament, ens cal seleccionar un punt aleatori on encreuar-les. Visualment, siguin Ga i Gb dos Gens es generarien dos fills Ga' i Gb' així:

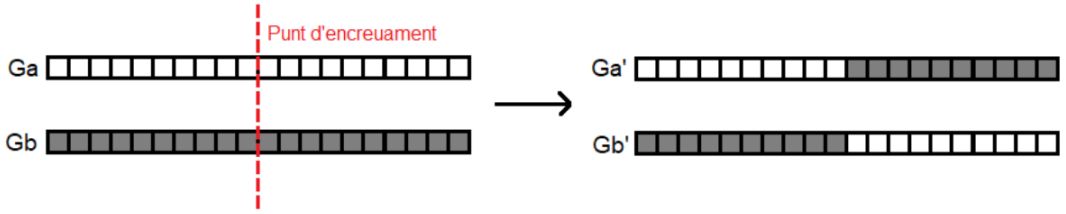



Figura 3: Encreuament de dos Gens

Aquest operador tindria cost asimptòtic $O(n)$ si implementa el Gen com a un vector de Booleans, donat que per a cada vèrtex del graf tenim un element. Un altre possibilitat seria implementar la cadena de booleans com una llista encadenada. Aleshores l'encreuament tindria cost $O(1)$, ja que només suposaria canviar els punters rellevants.

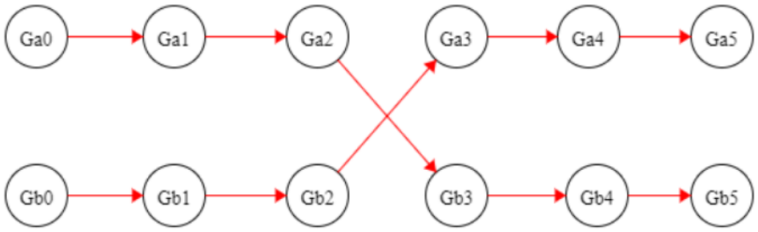



Figura 4: Encreuament de dos Gens amb llistes encadenades

Aquesta implementació és més complexa, i tot i que redueix considerablement el cost de l'encreuament augmenta el cost de les consultes. A més, la millora del cost de l'encreuament no suposaria una millora en el cost asimptòtic, com es demostrarà més endavant. Per tant, s'usaran vectors en la implementació final.

Per a l'operador d'encreuament, s'escollira un element aleatori del Gen i es canviarà de valor (de 0 a 1 i viceversa). Aquest operador té cost $O(1)$.

**Estratègia de Selecció** Es necessita seleccionar una quantitat d'individus per emparellar-los igual a la grandària de la població, valorant la seva adaptabilitat però mantenint la diversitat. Un gen pot ser seleccionat més d'una vegada o cap vegada. Una estratègia naïf seria donar a cada gen una probabilitat de ser seleccionat proporcional a la seva adaptabilitat. Això, però, no garanteix la diversitat i fa que la convergència sigui massa ràpida. La següent estratègia per generar els N gens seleccionats aconsegueix millor els objectius:

- Generar N tornejos entre parelles de gens seleccionades aleatòriament
- Per a cada parella, seleccionar el gen amb major adaptabilitat.

**Població Inicial** Per a generar la solució inicial s'ha fet servir l'Algorisme Golafre Estocàstic vist anteriorment. Abans d'això, però, es va provar una generació de poblacions inicials completament aleatòria. Aquesta versió generava resultats molt més pitjors i es va decidir descartar-la: la qualitat de les solucions inicials sembla impactar enormement en el bon funcionament de l'algorisme genètic.

### 5.1.4 Valors Constants

Per facilitar l'experimentació i aclarir la redacció del càlcul de cost s'han definit les següents constants:

- *POP_SIZE*: representa el nombre d'individus que conformen la població.
- *MAX_GEN*: representa el màxim nombre de generacions que es realitzaran.
- *PROB_CROSS*: representa la probabilitat d'encreuament.
- *PROB_MUTATE*: representa la probabilitat de mutació.

- *PROF*: representa l'"esforç" de l'algorisme golafre inicial. És directament proporcional a la qualitat però inversament al temps d'execució.

A més, tenim les següents variables que depenen de l'entrada

- $N$: representa el nombre de vèrtexs del graf.
- $M$: representa el nombre d'arestes del graf.

### 5.1.5 Anàlisi dels Costos

Per a analitzar asimptòticament el cost, assumim que ja tenim les solucions inicials generades i que no es realitza cap comprovació posterior a l'algorisme. També assumim que els operadors d'encreuament i mutació es realitzen sempre, i que es realitzen tantes generacions com el màxim possible (és a dir, la població no convergeix abans ni ens limita el temps d'execució).

Com s'ha vist previament, els cost de l'operador de mutació és $O(1)$ i el cost de l'operador d'encreuament és $O(N)$. La funció d'adaptabilitat recorre linealment els vèrtexs adjacents per a cada vèrtex del graf. Per tant, el seu cost és $O(N + M)$. Es realitzen *MAX_GEN* iteracions, i a cada una s'han de generar *POP_SIZE* nous individus.

El cost, doncs, és dins de l'ordre de $O(MAX\_GEN*POP\_SIZE*(1+N+(N+M))$. Asimptòticament es pot eliminar constants, i obtenir el cost $O(N + M)$. Aquest cost és lineal, però en aplicacions reals estarà limitat per les constants. Si tenim en compte el cost de generar la població inicial amb algorisme golafre, augmentaria l'ordre del cost asimptòtic. Es podria implementar amb una generació inicial de cost lineal, però com s'ha pogut comprovar la qualitat de les solucions inicials afecta enormement al rendiment de l'algorisme genètic.

## 5.2 Experiments i Resultats

Realitzant múltiples execucions del programa amb els inputs donats s'ha conclós que aquest eren els valors òptims per als paràmetres:

- *POP_SIZE*: 10.
- *MAX_GEN*: 1000.
- *MUTATE_PROB*: 0.10
- *CROSS_PROB*: 0.95.

Un valor màxim de generació alt permet que l'algorisme realitzi més iteracions i per tant aconsegueixi millorar més la població, mentre que una població gran no sembla tenir tant efecte positiu en els resultats. Amb la probabilitat de mutació s'ha trobat un equilibri delicat: una probabilitat més alta causava que l'adaptabilitat total de la població fos inestable i no millorés continuament, mentre que un valor més baix causava que la població es quedés estancada i li costés millorar l'adaptabilitat. La probabilitat d'encreuament s'ha trobat que amb un valor molt alt permet que l'adaptabilitat total de la població vagi millorant continuament.

El paràmetre de *PROF* (que determina la qualitat de les solucions inicials) ha sigut el paràmetre més difícil de decidir. Un valor baix de profunditat causava que l'algorisme realitzés millores substancials en les solucions respecte les inicials, però donava pitjors resultats finals que una profunditat més alta. En aquest últim cas, les contribuicions de l'algorisme genètic queden molt reduïdes degut al poc marge de millora que hi havia; el conjunt de solucions millors és bastant específic i difícil de troba0r amb un algorisme estocàstic com és l'algorisme genètic. Decidir quin valor de profunditat és millor podria dependre de la densitat d'arestes de l'entrada, per exemple en el fitxer de dades d'*actors.txt* l'impacte en el temps d'execució de valors alts de profunditat és molt menor que en un fitxer com *astro.txt*. Es creu que amb una funció d'adaptabilitat diferent o amb una generació inicial més diversa pot ser no hi hauria aquest problema (o potser implementant una metaheurística híbrida combinant l'algorimse genètic amb altres tècniques, com es menciona a [6]). Finalment, s'ha decidit prioritzar la qualitat de les sol·lucions finals i

s'ha escollit un valor de profunditat de 15, que és un valor bastant alt.

Un cop establerts els paràmetres s'han realitzat els experiments, els resultats dels quals es poden trobar a l'apartat 7.1 de conclusions.

# 6 Programació Linear Entera

## 6.1 Introducció a la Programació Lineal

La programació lineal (LP) és un mètode de resolució de problemes que pot ser usat quan un problema pot ser expressat com un una funció lineal d'una o més variables que es vol optimitzar i un conjunt de restriccions d'igualtats i desigualtats sobre les variables mencionades.

Sigui $x$ el vector de variables que es vol optimitzar del problema, es pot expressar un problema de la següent forma:

$$\textit{minimitzar/maximitzar: } \sum_{i=1}^{n} c_i x_i$$

*subjecte a:*

$$\textit{restriccions del tipus: } \sum_{i=1}^{n} A_i x_i \ \textit{op}\ b$$

on $c$ i $A$ són vectors de ponderacions, $b$ és un valor i op és un operador de comparació tal que $op \in \{<, >, \leq, \geq, =\}$. Els valors d'A, b i op poden ser diferents entre restriccions.

També pot haver-hi alguna restricció del tipus: $\forall i \in \{1, n\}, x_i \in \mathbb{Z}$. En aquest cas, el problema de programació lineal es pot especificar com un problema de programació lineal entera (ILP per les sigles en anglès).

Un problema és de programació lineal entera quan totes les variables estan restringides a ser enteres. Si només algunes de les variables són enteres, llavors es parla d'un problema de programació lineal entera mixta. Si totes les variables han de prendre valors binaris (només 0 o 1), llavors es pot parlar d'un problema de programació lineal 0-1. (Aquest cas es pot considerar també d'ILP, ja que seria un ILP normal afegint restriccions del tipus $\forall i \in \{1, n\}, x_i \geq 0 \wedge x_i \leq 1$).

Un problema es pot expressar en forma canònica de la següent manera:

$$\begin{aligned} &\textit{min } c^T x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

$$c \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ A \in \mathbb{R}^{n \times m},\ n \geq m,\ m = rang(A)$$

Com es tracta d'operacions amb vectors i matrius, la transposició del vector c és necessària per a poder fer la minimització d'un valor: el valor obtingut després de la multiplicació $c^T x$.

En la representació canònica, les variables s'acostumen a identificar com les columnes de la matriu, i les restriccions com les files.

La forma canònica no és la més convenient per a la resolució d'algorismes, però va bé per una representació clara.

### 6.1.1 Definicions bàsiques

- Qualsevol vector $x$ tal que $Ax = b$ és una **solució**.
- Una solució $x$ que satisfaci $x \geq 0$ és una solució factible.
- Si un LP conté solucions factibles, llavors és un problema **factible**. Sinó és un problema **inviable**.
- Una solució factible $x^*$ s'annomena òptima si per a tot solució factible $x$ es compleix que $c^T x^* \leq c^T x$
- Un LP factible sense solució òptima és **ilimitat**.

### 6.1.2 Geometria de la Programació Lineal

Els problemes de LP es poden representar geomètricament en un espai de $n$ dimensions, on $n$ és el nombre de variables del problema. En aquesta representació, el conjunt de solucions factibles és un políedre convex de $n$ dimensions: és a dir, és un políedre de $n$ dimensions on per a tota parella de punts pertanyent a les cares del políedre, una recta que passi pels 2 punts només tallarà com a molt dues vegades amb els límits del políedre.

Les solucions bàsiques i factibles del problema coincideixen amb els vèrtexs del políedre convex.

Per a entendre millor la geometria de la programació linal es proposa un exemple d'optimització de dues variables (la qual cosa resultarà en una geometria bidimensional). L'exemple està extret del llibre CLRS, capítol 29, i és el següent: [14]

$$
\begin{array}{ll}
\textit{maximize} & x_1 + x_2 \\
\textit{subject to} & \\
& 4x_1 - x_2 \leq 8 \\
& 2x_1 + x_2 \leq 10 \\
& 5x_1 - 2x_2 \geq -2 \\
& x_1, x_2 \geq 0
\end{array}
$$

Aquest problema de LP dóna resultat als gràfics següent:

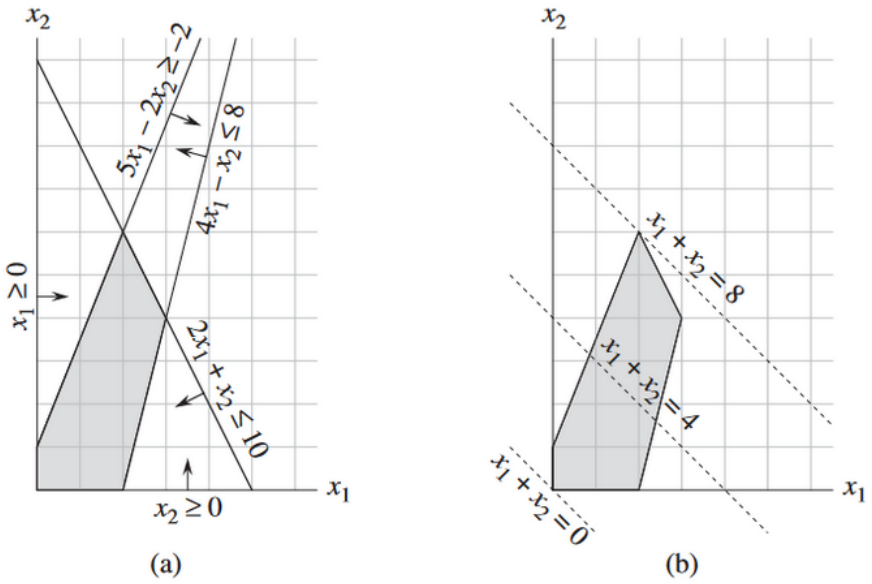



Figura 5: Gràfics que representen en un espai bidimensional les restriccions, l'espai de solucions factibles i la funció a optimitzar

En el *gràfic (a)* es mostren les restriccions en forma de rectes i unes fletxes en un sentit, indicant quina zona és l'acceptada. En gris es té la regió factible, que és la zona on intersecten les restriccions.

En el *gràfic (b)* es mostra la regió factible en gris igual que abans, aquest cop, però, mostrant només la zona de les restriccions que pertanyen al políedre. Les rectes amb guions representen la funció a optimitzar quan es pren com a valor de la recta l'indicat. En el cas d'aquest exemple, es pot veure que la intersecció màxima entre la recta i el políedre és quan la recta pren per valor 8 i és tangent al punt $x_1 = 2$, $x_2 = 6$.

### 6.1.3 Teorema fonamental de la Programació Lineal

El teorema fonamental de la programació lineal diu que tot LP factible que té un límit inferior té una solució òptima en una cara de zero dimensions (és a dir, un vèrtex) del políedre convex. [13]

Aquest teorema és útil per al funcionament d'alguns mètodes de resolució de problemes de LP, i s'usarà en apratats posteriors.

### 6.1.4 Mètodes de programació lineal

En aquest apartat s'expliquen breument diferents mètodes per a resoldre un problema de LP. Més endavant ja s'especificarà quin s'ha triat per a la resolució del problema MPIDS, però és simportant tenir una noció bàsique d'altres opcions disponibles.

#### 6.1.4.1 Mètode Simplex

El teorema fonamental de la programació lineal ens assegura que és suficient explorar les solucions bàsiques factibles per a trobar l'òptim en un problema limitat i factible. El mètode simplex es va movent entre solucions factibles bàsiques que no empitjorin la funció objectiu (la funció a optimitzar) fins que o bé es troba la optimalitat o bé es detecta que no és limitat.

#### 6.1.4.2 Mètode Simplex Revisat

És una optimització del mètode Simplex que guanya espai d'emmagatzematge i temps emprat, i conté un cert control sobre els errors de precisió de les operacions de coma flotant.

#### 6.1.4.3 Mètode Simplex Dual

A diferència del mètode Simplex, aquest no es va movent entre solucions factibles, mirant si aquestes compleixen la optimalitat (les condicions de la funció a optimitzar), sinó que es va movent entre solucions que compleixen la optimalitat i va comprovant si aquestes són factibles.

## 6.2 CPLEX

L'eina usada en aquest apartat de la pràctica és CPLEX, una eina d'IBM per a la resolució de problemes d'optimització lineal (problemes de LP). Té implementats diferents mètodes de resolució de problemes, entre els que es troben alguns dels comentats anteriorment.

Per a la realització de la pràctica, en una primera proposta s'ha optat per a usar un mètode Simplex Dual, ja que per a programació amb ILP és teòricament més eficient que el mètode Simplex (també annomenat mètode Simplex Primal). [4] Després de buscar informació sobre CPLEX però, es va veure que per defecte et tria l'algorisme que considera el més convenient. Per tant, s'ha decidit deixar el paràmetre per defecte.

## 6.3 ILP aplicat a MPIDS

Un cop ja es té prou informació sobre ILP i CPLEX es pos passar a la resolució del problema. Per al problema del Minimum Positive Influence Dominating Set (MPIDS) s'ha escollit una representació 0-1 LP, amb un vector de n elements com a variable (on n és el nombre de vèrtexs del graf) i on cada element del vector pot prendre valors binaris. Com a restriccions s'ha escollit fer una restricció per cada un dels nodes, de tal manera que la suma dels veins d'un noed sigui superior a la meitat del nombre de veïns. Aquesta representació mencionada és pot expressar formalment de la següent manera:

$$
\begin{aligned}
&\textit{minimitzar:} \quad \sum_{i=1}^{n} x_i \\
&\textit{subjecte a:} \\
&\qquad \forall v \in V, \sum_{i \in Neig(v)} x_i \geq \left\lceil \frac{deg(v)}{2} \right\rceil \\
&\qquad \forall i \in [1,n], x_i \in \{0,1\}
\end{aligned}
$$

$Neig(v) \equiv$ veins del node $v$
$deg(v) \equiv$ grau del node $v$ (és a dir, quants veïns té)

Amb aquesta representació i la configuració de l'algorsime de resolució de CPLEX mencionat anteriorment, s'ha passat la representació a codi en c++ i s'ha executat l'algorisme.

## 6.4 Experimentació

Per a provar la representació feta en ILP del problema de MPIDS s'ha executat amb tots els jocs de proves proporcionats en l'enunciat. A diferència de les dades de referència però, s'ha executat l'algorisme amb un temps màxim de 10 minuts (en comptes de les dues hores que s'usaven en els experiments de

referència). Tot i així, s'ha pogut veure que en 10 minuts es pot aconseguir una solució molt semblant al de la solució (en cas que es passés els 10 minuts fent càlculs), i el mateix resultat quan ha acabat abans, assegurant una solucó òptima.

Els resultats dels experiments es troben en l'apartat de resultats 7.1 de les conclusions.

# 7 Conclusions

## 7.1 Resultats

Finalment, per a tots els problemes hem obtingut els següents resultats:

| Resultats | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark | Greedy | | Local Search | | Metaheuristic | | ILP | |
| nº elements a PIDS i s | Val | t | Val(avg) | t(avg) | Val(avg) | t(avg) | Val | t |
| soc-gplus | 8341 | 3.13 | 8347.90 | 30.73 | **8309.20** | 143.11 | **8244** | 0.11 |
| graph_CA-CondMat | 9847 | 26.25 | 9779.40 | 28.15 | **9769.40** | 96.36 | 9584 | 600 |
| graph_CA-AstroPh | 7083 | 46.05 | 6961.10 | 75.90 | **6957.80** | 490.91 | 6674 | 600 |
| graph_CA-HepPh | 4880 | 15.03 | **4816.30** | 27.09 | 4821.40 | 158.57 | **4718** | 89.98 |
| graph_actors_dat | 3196 | 12.86 | **3169.30** | 22.54 | 3172.60 | 127.51 | 3092 | 600 |
| ego-facebook | 1978 | 4.71 | **1975.60** | 5.69 | **1975.60** | 28.35 | **1973** | 1.01 |
| socfb-Brandeis99 | 1533 | 9.74 | **1482.40** | 29.04 | 1483.50 | 241.32 | 1401 | 600 |
| socfb-Mich67 | 1491 | 4.47 | 1421.50 | 18.16 | **1419.90** | 106.66 | 1332 | 600 |
| graph_jazz | **81** | 0.00 | 83.20 | 0.44 | 81.60 | 0.14 | **79** | 0.19 |
| graph_football | 68 | 0.00 | **67.10** | 0.36 | 69.30 | 0.03 | **63** | 7.99 |

Per a l'algorisme Golafre s'ha fet servir l'algorisme *Golafre Ingenu*, donat que dona els millors resultats. Per als algorismes de *Cerca Local* i *Metaheurístic* els resultats donats són la mitjana de 10 iteracions del programa per un dataset. En tots els programes s'ha establert un límit temporal de 600 segons.

Per la columna de ILP, estan marcats en negreta quins dels resultats obtinguts tenen l'optimalitat assegurada. Respecte les altres 3 columnes, en cada fila es marca en negreta el valor més petit obtinguts entre els 3 algoritmes.

Podem observar com el millor resultat es troba disputat entre els algorismes de Cerca Local i la Metaheurística, amb la Metaheurística generant els resultats lleugerament millors en els benchmarks amb major nombre de nodes. Però, si el temps de computació és una preocupació, la Cerca Local podria ser preferible donat que genera valors comparables en temps molt menors. També s'hauria de tenir en compte que l'algorisme golafre no implementa cap tècnica de reducció a conjunt minimal o cap preprocessat de les dades (com és el cas de l'algorisme plantejat a [6]), donat que això faria que el nostre algorisme no fos estrictament golafre. Combinar l'estratègia golafre amb alguna estratègia de minimalització potser generaria resultats competitius. Sobre el cas de programació lineal, creiem que és molt variable si troba l'òptim o no en un temps raonable. Però, s'ha pogut observar que genera bons resultats en temps comparables als altres algorismes.

## 7.2 Aprenentatges

Realitzar aquest treball ens ha donat una renovada perspectiva sobre els Algorismes Golafres. A l'assignatura d'Algorísmia s'analitzen teòricament, i creiem que haver-ne implementat ens ha permès consolidar els nostres coneixements sobre els Algorismes Golafres. Hem pogut comprovar que els Algorismes Golafres són eines poderoses, i ens veiem segurs de concloure que aquest tipus d'algorismes són una aposta segura com a primer intent d'abordar problemes d'optimització combinatòria.

La realització de la Cerca Local ens ha permès aprendre sobre les tècniques de Hill Climbing i Simulated Annealing i la seva utilitat en problemes d'optimització. També hem après les limitacions de Hill Climbing en problemes amb un gran nombre de mínims locals, com és aquest problema, i hem vist com l'algorisme Simulated Annealing aconsegueix solventar aquesta limitació gràcies a la seva estocasticitat. Hem après els efectes de cada paràmetre de Simulated Annealing i sobre com optimitzar-los mitjançant l'experimentació.

També hem pogut comprendre el funcionament de l'Algorisme Genètic i la seva relativa simplicitat d'implementació. Hem vist com el seu funcionament simula l'evolució de les espècies. Hem pogut observar el gran paper que juga la generació de població inicial en l'Algorisme Genètic, i com s'ha de buscar tant la relativa optimalitat de les solucions inicials com la seva diversitat. Però també hem vist les seves

limitacions. Tot i que el problema del MPIDS té una fàcil codificació en binari que afavoreix a l'Algorisme Genètic, creiem que aquest problema busca un equilibri molt delicat que és difícil de trobar amb aquest algorisme. El llibre *Artificial Intelligence: a Modern Approach* il·lustra molt bé aquest problema: *"Actualment, no està clar si l'atractiu dels algorismes genètics sorgeix del seu rendiment o dels seus orígens estèticament agradables en la teoria de l'evolució. Queda molta feina per fer per identificar les condicions en què funcionen bé els algorismes genètics"*.

En el cas de *ILP* hem pogut aprendre la metodologia de programació lineal i la seva aplicació en l'eina *CPLEX* de *IBM*. Hem vist com és una eina capaç de generar molt bones solucions (o fins i tot les òptimes) en un temps relativament baix. Però, tot això està subjecte a ser capaços de representar les restriccions del problema en termes de programació lineal. En general, estem sorpresos pel funcionament de la programació lineal i creiem que, donat un problema que compleixi les restriccions, la programació lineal seria una molt bona candidata per la seva resolució.

Finalment, després de realitzar aquest treball considerem que les nostres capacitats de cerca, lectura i referenciat de *papers* científics professionals han millorat considerablement. Aquest treball també ens ha donat l'oportunitat d'aprendre a utilitzar el programari LaTeX. Creiem que aquests dos coneixements ens serviran durant la resta de la nostra trajectòria acadèmica i professional.

# 8 Referències


[1] IBM. (n.d.). WHAT IS CPLEX? Retrieved November 25, 2021. https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-what-is.

[2] *Integer Programming.* MIT - Optimization Methods in Business Analytics. (n.d.). Retrieved November 24, 2021, from. https://web.mit.edu/15.053/www/AMP-Chapter-09.pdf.

[3] profjoshemman. (2016, April 30). *Integer linear programming - graphical method - optimal solution, mixed, rounding, relaxation.* YouTube. Retrieved November 25, 2021. https://www.youtube.com/watch?v=RhHhy-8sz-4.

[4] UPC. (n.d.). Combinatorial Problem Solving (CPS). CPS - Master in innovation and research in informatics (MIRI). Retrieved November 25, 2021. https://www.cs.upc.edu/~erodri/webpage/cps/cps.html.

[5] Traian Marius Truta Alina Campan and Matthew Beckerich. Fast dominating set algorithms for social networks. *RWTH Aachen University*, 2015.

[6] Salim Bouamama and Christian Blum. An improved greedy heuristic for the minimum positive influence dominating set problem in social networks. *Algorithms*, pages 14–79, 2021.

[7] Akshaye Dhawan and Matthew Rink. Positive influence dominating set generation in social networks. *ResearchGate*, 2015.

[8] Thomas Stützle Holger H. Hoos. *Stochastic Local Search: Foundations and Applications.* Elsevier, 2004.

[9] Jon Kleinberg and Éva Tardo. *Algorithm Design*, chapter 12. Pearson/Addison-Wesley, 2006.

[10] Jiehui Pan and Tian-Ming Bu. A fast greedy algorithm for finding minimum positive influence dominating sets in social networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 360–364, 2019.

[11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Hoboken, New Jersey, 1995.

[12] Kenneth Sörensen and Fred Glover. Metaheuristics. *Encyclopedia of Operations Research and Management Science*, pages 960–970, 2013.

[13] F. Tardella. The fundamental theorem of linear programming: extensions and applications, Optimization, 60:1-2, 283-301, DOI: 10.1080/02331934.2010.506535.

[14] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms, 3rd Edition. Chp 29: Linear Programming.* MIT Press, 2009.

[15] Feng Wang, Erika Camacho, and Kuai Xu. Positive influence dominating set in online social networks. *Springer*, pages 313–321, 2009.

[16] Feng Wang, Hongwei Du, Erika Camacho, Kuai Xu, Wonjun Lee, Yan Shi, and Shan Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, pages 265–269, 2011.

[17] Lidong Wu Weidong Chen, Hao Zhong and Ding-Zhu Du. A general greedy approximation algorithm for finding minimum positive influence dominating sets in social networks. *Springer*, 2021.

# A Resultats de l'Experiment per la Tria de Paràmetres per Simulated Annealing

| K | L | MIN | AVG | VSD | TIME | TSD |
|---|---|---|---|---|---|---|
| data/soc-gplus.txt | | | | | | |
| 2 | 0.1 | 8333 | 8339.20 | 5.11 | 17.85 | 0.57 |
| 2 | 0.075 | 8325 | 8334.60 | 5.54 | 17.63 | 0.34 |
| 2 | 0.05 | 8334 | 8342.00 | 7.48 | 17.48 | 0.82 |
| 2 | 0.025 | 8331 | 8336.00 | 3.85 | 16.77 | 0.38 |
| 6 | 0.1 | 8325 | 8335.60 | 6.68 | 18.00 | 0.59 |
| 6 | 0.075 | 8325 | 8333.00 | 5.40 | 18.27 | 0.79 |
| 6 | 0.05 | 8331 | 8334.00 | 2.97 | 17.76 | 0.19 |
| 6 | 0.025 | 8328 | 8338.40 | 7.50 | 18.42 | 0.38 |
| 10 | 0.1 | 8330 | 8338.20 | 5.08 | 17.52 | 0.56 |
| 10 | 0.075 | 8329 | 8334.40 | 4.18 | 17.77 | 0.47 |
| 10 | 0.05 | 8324 | 8335.80 | 8.80 | 17.33 | 0.11 |
| 10 | 0.025 | 8321 | 8335.20 | 8.11 | 17.60 | 0.79 |
| 14 | 0.1 | 8319 | 8335.20 | 9.30 | 18.09 | 0.56 |
| 14 | 0.075 | 8318 | 8331.40 | 9.29 | 17.21 | 0.26 |
| 14 | 0.05 | 8328 | 8335.20 | 4.53 | 17.26 | 0.49 |
| 14 | 0.025 | 8327 | 8333.20 | 4.92 | 17.21 | 0.39 |
| data/graph_actors_dat.txt | | | | | | |
| 2 | 0.1 | 3239 | 3246.80 | 6.55 | 9.22 | 0.32 |
| 2 | 0.075 | 3225 | 3238.20 | 7.83 | 8.61 | 0.14 |
| 2 | 0.05 | 3232 | 3239.00 | 4.38 | 9.26 | 0.46 |
| 2 | 0.025 | 3239 | 3241.20 | 1.72 | 9.60 | 0.51 |
| 6 | 0.1 | 3229 | 3243.60 | 10.87 | 9.10 | 0.19 |
| 6 | 0.075 | 3232 | 3245.80 | 8.59 | 9.48 | 0.19 |
| 6 | 0.05 | 3233 | 3245.00 | 10.94 | 9.88 | 0.38 |
| 6 | 0.025 | 3227 | 3238.20 | 6.76 | 9.30 | 0.27 |
| 10 | 0.1 | 3236 | 3239.60 | 4.03 | 9.43 | 0.26 |
| 10 | 0.075 | 3231 | 3236.20 | 5.19 | 9.27 | 0.24 |
| 10 | 0.05 | 3226 | 3239.80 | 8.95 | 9.28 | 0.19 |
| 10 | 0.025 | 3225 | 3236.80 | 6.71 | 9.24 | 0.09 |
| 14 | 0.1 | 3232 | 3242.20 | 7.78 | 9.51 | 0.44 |
| 14 | 0.075 | 3225 | 3244.00 | 10.66 | 9.50 | 0.13 |
| 14 | 0.05 | 3236 | 3240.20 | 2.71 | 9.35 | 0.13 |
| 14 | 0.025 | 3231 | 3241.20 | 5.38 | 9.12 | 0.12 |
| data/socfb-Brandeis99.txt | | | | | | |
| 2 | 0.1 | 1483 | 1489.40 | 5.68 | 16.66 | 0.58 |
| 2 | 0.075 | 1480 | 1487.00 | 5.55 | 16.24 | 0.20 |
| 2 | 0.05 | 1488 | 1493.80 | 3.76 | 16.58 | 0.20 |
| 2 | 0.025 | 1481 | 1487.80 | 6.05 | 16.43 | 0.80 |
| 6 | 0.1 | 1480 | 1487.20 | 5.19 | 16.03 | 0.36 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 0.075 | 1480 | 1486.00 | 6.72 | 16.26 | 0.81 |
| 6 | 0.05 | 1481 | 1487.60 | 5.95 | 15.85 | 0.30 |
| 6 | 0.025 | 1484 | 1488.80 | 2.71 | 16.36 | 0.68 |
| | | | | | | |
| 10 | 0.1 | 1483 | 1488.00 | 4.34 | 16.52 | 0.38 |
| 10 | 0.075 | 1484 | 1489.00 | 5.87 | 15.95 | 0.15 |
| 10 | 0.05 | 1482 | 1485.80 | 2.40 | 15.33 | 0.34 |
| 10 | 0.025 | 1484 | 1489.60 | 3.01 | 15.79 | 0.26 |
| | | | | | | |
| 14 | 0.1 | 1480 | 1487.40 | 4.32 | 14.90 | 0.16 |
| 14 | 0.075 | 1484 | 1488.20 | 3.06 | 15.41 | 0.46 |
| 14 | 0.05 | 1486 | 1488.80 | 2.71 | 15.45 | 0.17 |
| 14 | 0.025 | 1478 | 1487.60 | 5.08 | 15.98 | 0.13 |