Pràctica 4: Optimització en SAT

Lògica en la Informàtica

FIB

Antoni Lozano

Q2 2023–2024

Objectius

Aquesta pràctica té com a objectiu:

- Fer servir *SAT solvers* per optimitzar problemes combinatoris.
- Seguint l'exemple de **minColoring**, resoldre **towers** i **gangsters**.

Referències

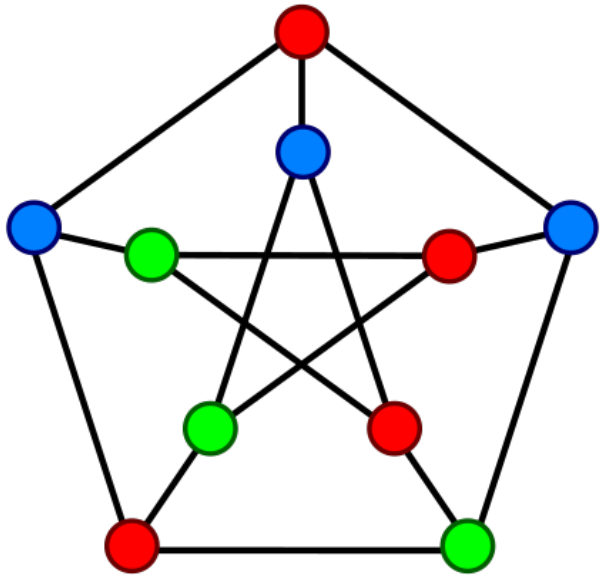
Com a guia d'estudi teniu

- l'exemple del **minColoring**
- aquestes transparències

Min Coloring

Coloració i nombre cromàtic

Una coloració (dels vèrtexs) d'un graf G és una assignació d'etiquetes de colors a cada vèrtex de G tal que cap aresta connecta dos vèrtexs amb el mateix color.
Una coloració que minimitza el nombre de colors necessaris per acolorir un graf G se'n diu coloració mínima de G. El nombre mínim de colors necessaris per acolorir un graf G se'n diu nombre cromàtic de G i es representa amb $\chi(G)$.



Min Coloring

L'objectiu és trobar el nombre cromàtic d'un graf donat en el format:

```
%%%%%%% Begin example input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numNodes(15).
adjacency(1, [ 2,3,4,5,6,    9,10,11,12,13,14,15]).
adjacency(2, [1,  3,4,5,6,7,  9,   11,12,      15]).
adjacency(3, [1,2,  4,5,6,7,8,9,10,   12,13,14  ]).
adjacency(4, [1,2,3,  5,6,7,  9,10,11,12,13,   15]).
adjacency(5, [1,2,3,4,    7,8,9,      12,13,   15]).
adjacency(6, [1,2,3,4,      8,  10,11,          15]).
adjacency(7, [ 2,3,4,5,    8,9,10,11,      14  ]).
adjacency(8, [    3,  5,6,7,  9,10,      13,14,15]).
adjacency(9, [1,2,3,4,5,  7,8,    11,          15]).
adjacency(10,[1,  3,4,  6,7,8,    11,12,   14,15]).
adjacency(11,[1,2,  4,  6,7,  9,10,   12,13,14  ]).
adjacency(12,[1,2,3,4,5,        10,11,   13,14,15]).
adjacency(13,[1,  3,4,5,    8,    11,12,   14,15]).
adjacency(14,[1,  3,      7,8,  10,11,12,13,   15]).
adjacency(15,[1,2,  4,5,6,  8,9,10,   12,13,14  ]).

%%%%%%% End example input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Esquema general

El nostre esquema per resoldre problemes d'optimització amb un *SAT solver* genera les clàusules, crida al *SAT solver*, mostra la solució i calcula el seu cost.

Només cal especificar:

1. Les variables SAT (`satVariable`)
2. Les clàusules que descriuen el problema (`writeClauses`)
3. El format de la solució (`displaySol`)
4. El càlcul del cost (`costOfThisSolution`)

Min Coloring

```
%%%%%% Some helpful definitions to make the code cleaner: ================

node(I):-   numNodes(N), between(1,N,I).
edge(I,J):- adjacency(I,L), member(J,L).
color(C):-  numNodes(N), between(1,N,C).

%%%%%% End helpful definitions ========================================


%%%%%%  1. Declare SAT variables to be used: ============================

% x(I,C)    meaning  "node I has color C"
satVariable( x(I,C) ):- node(I), color(C).
```

Min Coloring

```
%%%%%%  2. Clause generation for the SAT solver: ========================================

% This predicate writeClauses(MaxCost) generates the clauses that guarantee that
% a solution with cost at most MaxCost is found

writeClauses(infinite):- !, numNodes(N), writeClauses(N),!.
writeClauses(MaxColors):-
    eachNodeExactlyOnecolor(MaxColors),
    noAdjacentNodesWithSameColor(MaxColors),
    true,!.
writeClauses(_):- told, nl, write('writeClauses failed!'), nl,nl, halt.

eachNodeExactlyOnecolor(MaxColors):-
    node(I), findall(x(I,C), between(1,MaxColors,C), Lits ), exactly(1,Lits), fail.
eachNodeExactlyOnecolor(_).

noAdjacentNodesWithSameColor(MaxColors):-
    edge(I,J), between(1,MaxColors,C), writeOneClause([ -x(I,C), -x(J,C) ]), fail.
noAdjacentNodesWithSameColor(_).
```

Min Coloring

```
%%%%%%% 3. DisplaySol: this predicate displays a given solution M: ============================

% displaySol(M):- nl, write(M), nl, nl, fail.
displaySol(M):- node(I), member(x(I,C),M), write(I-C), write(' '), fail.
displaySol(_):- nl,!.


%%%%%%% 4. This predicate computes the cost of a given solution M: ============================

% Here the sort predicate is used to remove repeated elements of the list:
costOfThisSolution(M,Cost):- findall(C,member(x(_,C),M),L), sort(L,L1), length(L1,Cost), !.
```

Min Coloring

```
?- [minColoring].
true.

?- main(minColoringInput1).
Looking for initial solution with arbitrary cost...
Generated 3840 clauses over 225 variables.
Launching kissat...

Solution found with cost 7

1-11 2-13 3-15 4-9 5-10 6-14 7-11 8-12 9-14 10-13 11-15 12-12 13-13 14-14 15-15


Now looking for solution with cost 6...
Generated 1140 clauses over 90 variables.
Launching kissat...

Solution found with cost 6

1-2 2-4 3-6 4-1 5-5 6-3 7-2 8-1 9-3 10-4 11-6 12-3 13-4 14-5 15-6


Now looking for solution with cost 5...
Generated 915 clauses over 75 variables.
Launching kissat...

Unsatisfiable. So the optimal solution was this one with cost 6:
1-2 2-4 3-6 4-1 5-5 6-3 7-2 8-1 9-3 10-4 11-6 12-3 13-4 14-5 15-6
```

Towers

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% We are given a map with NxM positions. On the map are a number of square villages.
%% On each position inside a village we are allowed to place a tower, at most one tower per
%% village.  These towers can be used to watch horizontally and vertically (like the towers
%% or castles in the game of chess).
%% The aim is to determine where to position the towers such that each village (has at least
%% one position that) is watched by at least one tower (for example, if the village has a
%% tower itself), and MINIMIZE the total number of towers.
%% There is a list of "significant" villages that MUST have a tower.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Condicions no explícites:

1. Les torres només es poden posar als pobles, no fora.
2. El nombre màxim de torres és `MaxNumTowers`
   (la variable apareix al predicat `writeClauses(MaxNumTowers)`)
3. Les que serveixin per relacionar diferents *SAT variables* entre si.

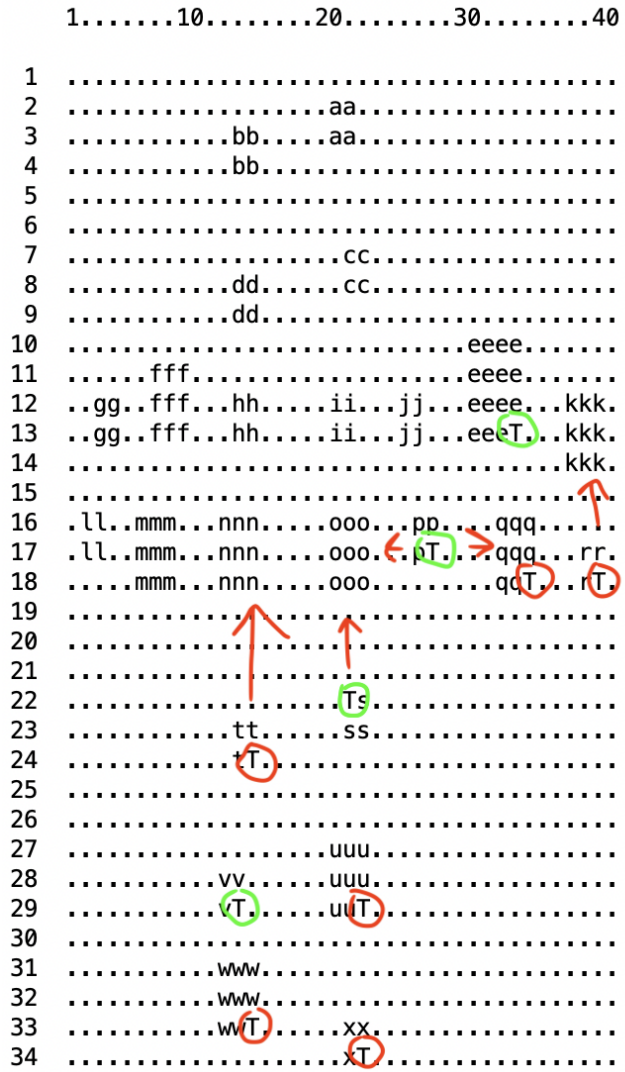
Towers

Inici de l'entrada `towersInput1`:

```
%%%%%%% begin input example towersInput1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% upperLimitTowers(10).         % maxim number of towers allowed
%% significantVillages([e,q,r]). % list of villages where a tower is mandatory
%% map_size(34,40).              % numrows, numcols; the left upper corner of the map has coordinates (1,1)
%%
%% village(a, 2,20,2).           % village(ident,row,col,size):  row,col of left upper corner, and size of the square
%% village(b, 3,13,2).
%% village(c, 7,21,2).
%% village(d, 8,13,2).
%% village(e,10,30,4).
%% village(f,11, 7,3).
%% village(g,12, 3,2).
%% village(h,12,13,2).
%% village(i,12,20,2).
%% village(j,12,25,2).
%% village(k,12,37,3).
%% village(l,16, 2,2).
%% village(m,16, 6,3).
%% village(n,16,12,3).
%% village(o,16,20,3).
%% village(p,16,26,2).
```

Towers

Solució subòptima (cost 10):



Towers

```
%%%%%%% Some helpful definitions to make the code cleaner: ==================================

row(I):- map_size(N,_), between(1,N,I).                          % I is a row in the map
col(J):- map_size(_,M), between(1,M,J).                          % J is a col in the map
position(I,J):- row(I), col(J).                                  % I-J is a position in the map
village(V):- village(V,_,_,_).                                   % V is a village identifier
rowVillage(V,I):- village(V,I1,_,S), I2 is I1+S-1, between(I1,I2,I).  % I is a row of the village V
colVillage(V,J):- village(V,_,J1,S), J2 is J1+S-1, between(J1,J2,J).  % J is a col of the village V
posVillage(V,I,J):- rowVillage(V,I), colVillage(V,J).            % I-J is one position in village V
significantVillage(V):- significantVillages(L), member(V,L).     % V is a significant village
posWatchesVillage(V,I,J):- position(I,J), rowVillage(V,I).       % position (I,J) watches village V
posWatchesVillage(V,I,J):- position(I,J), colVillage(V,J).       %                 "

%%%%%%% End helpful definitions ============================================================
```

Towers

```
%%%%%%  1. Declare SAT variables to be used: ==========================================

satVariable( towerPos(I,J) ):- row(I), col(J).  % means "there is a tower at position I-J"
% YOU MAY WANT TO INTRODUCE SOME OTHER VARIABLE FOR MAKING THE CARDINALITY CONSTRAINTS SMALLER


%%%%%%  2. Clause generation for the SAT solver: ======================================

% This predicate writeClauses(MaxCost) generates the clauses that guarantee that
% a solution with cost at most MaxCost is found

writeClauses(infinite):- !, upperLimitTowers(N), writeClauses(N),!.
writeClauses(MaxNumTowers):-
    ...
    true,!.                    % this way you can comment out ANY previous line of writeClauses
writeClauses(_):- told, nl, write('writeClauses failed!'), nl,nl, halt.
```

Towers

```
%%%%%%% 3. DisplaySol: this predicate displays a given solution M: ============================

displaySol(M):- write('    1.......10........20........30........40'), nl,
                row(I), nl, write2(I), write('  '), col(J), writePos(M,I,J), fail.
displaySol(_):- nl,nl.

writePos(M,I,J):- member(towerPos(I,J),M), write('T'), !.
writePos(_,I,J):- posVillage(V,I,J), write(V), !.
writePos(_,_,_):- write('.'), !.
write2(N):- N < 10, !, write(' '), write(N),!.
write2(N):- write(N),!.


%%%%%%% 4. This predicate computes the cost of a given solution M: ============================

costOfThisSolution(M,Cost):- ...
```

Gangsters

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The mafia has a lot of gangsters for doing different tasks.
%% These tasks are planned every 3 days (72h), according to a forecast
%% of the tasks to be done every hour.
%% No gangster can do two different tasks during the same hour or on two consecutive hours.
%% Some gangsters are not available on certain hours.
%% We want to plan all tasks (which gangster does what task when) and
%% we want to find the minimal K such that no gangster works more than
%% K consecutive hours.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gangsters

```
%%%%%%% begin input example gangstersInput1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% % example: 4 gangsters are needed for killing on hour 1, one gangster on hour 2, two gangsters
%% gangstersNeeded( killing,       [4,1,2,4,2,1,1,4,1,1,3,2,4,2,1,2,1,3,2,3,4,1,3,1,2,3,1,3,4,3,2
%% gangstersNeeded( countingMoney, [1,2,1,3,1,4,3,1,3,1,4,3,2,2,1,2,1,2,1,1,2,1,2,1,1,3,1,2,2,4,3
%% gangstersNeeded( politics,      [2,4,2,1,1,1,4,1,1,4,1,3,2,4,1,1,4,1,4,3,1,3,2,4,4,2,4,2,1,1,4
%%
```

72 hores

```
%% gangsters([g01,g02,g03,g04,g05,g06,g07,g08,g09,g10,g11,g12]).
%%
%% notAvailable(g01,[6,13,14,16,21,35,37,41,59]).
%% notAvailable(g02,[14,34,40,45,48,52,58,65,70,72]).
%% notAvailable(g03,[8,11,13,27,30,38,50,51,70]).
%% notAvailable(g04,[4,12,16,17,26,30,42,45,48,55,71]).

%%%%%%% end input example gangstersInput1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% EXAMPLE OUTPUT:
%% 
%%                       10        20        30        40        50        60        70  
%%               123456789012345678901234567890123456789012345678901234567890123456789012
%% 
%% gangster g01: -----------------------------p-------p-----p----p----p--------------p--
%% gangster g02: ------p--p-p--------------p--ppp-----pp---pp-----p----p---p----p--------
%% gangster g03: -p----p--p-p-p--p-------p-----pp-pp-p-pp-pppp--pp--p-pp-pppp--p-------p-
%% gangster g04: -pp---p--p--pp----pp-p-pp-p-p-pppppp-cc-p--c----pppppp-p-c-p--ppp--p-p--
%% gangster g05: pppppppppppppp--p-pp-p-ppppp-c--cc-p-cc-c-c-p--ppppppp-c-c-pppppppppppp-
%% gangster g06: pp-c--c-c-cc-pp-p-pppppppppp-c--ccc-ccc-c-c-ppppp-cc-p-c-c-pppppp-cc-ppp
%% gangster g07: -c-c-cc-c-cccc--ppp-c-pp-c-cccccccc-ccccccccc--c--ccccccccc-c--cc-cccc-p
%% gangster g08: ccccccccc-cccc-p-c-cccc--c-ccccccccc-k-ccccccc-cccccccc-k-p-c-ccc-ccccc-
%% gangster g09: k--k-c-k-cc-k--cccc-k-ccccc-k-c-k-c-kk-k-c-k-cccc-kk-cc-kk--cccccccccccc
%% gangster g10: k--k-c-k--k-k-cc-kkkk-k--k-kkk-kk-k-kk-k-c-kkkk--kkk-kkkkk-c-kkkk--k--k-
%% gangster g11: k-kkk--k--kkkk-k-kkkk-k-kk-kkkkkkkk-kk-kkk-kkkkkkkkk-kkkkk--kkkkkk-k--kk
%% gangster g12: kkkkkkkkkkkkkkkkkk-kkkkkkkkkkkkkkkkkk-kkkkkkkkkkkkkkkkkk-kkkkkkkkkkkkkkk
```

Gangsters

```
%%%%%%% Some helpful definitions to make the code cleaner: ====================================

task(T):-          gangstersNeeded(T,_).
needed(T,H,N):-    gangstersNeeded(T,L), nth1(H,L,N).
gangster(G):-      gangsters(L), member(G,L).
hour(H):-          between(1,72,H).
blocked(G,H):-     notAvailable(G,L), member(H,L).
available(G,H):-   hour(H), gangster(G), \+blocked(G,H).

%%%%%%% End helpful definitions ============================================================

%%%%%%%  1. Declare SAT variables to be used: ============================================

satVariable( does(G,T,H) ) :- ...   % means: "gangster G does task T at hour H"     (MANDATORY)
```

es requereixen N gàngsters per la tasca T en l'hora H

definir i afegir més SAT vars. si cal

Gangsters

```
%%%%%%%  2. Clause generation for the SAT solver: ========================================

% This predicate writeClauses(MaxCost) generates the clauses that guarantee that
% a solution with cost at most MaxCost is found

writeClauses(infinite) :- !, writeClauses(72),!.
writeClauses(MaxConsecutiveHours) :-
    ...
    true,!.
writeClauses(_) :- told, nl, write('writeClauses failed!'), nl,nl, halt.
```

será l'argument del predicat que asseguri que cap gàngster treballa més de MaxConsecutiveHours hores consecutives

Gangsters

```
%%%%%%  4. This predicate computes the cost of a given solution M: ============================

costOfThisSolution(M,Cost) :-
    between(0,71,N), Cost is 72-N,
    gangster(G),
    
    
    
    
    

%%%%%% ========================================================================================
```

← per comprovar si el cost és 72, 71, 70, …