Pràctica 4: Optimització en SAT

Lògica en la Informàtica

FIB

Antoni Lozano

Q1 2024–2025

Objectius

Aquesta pràctica té com a objectiu:

- Fer servir *SAT solvers* per optimitzar problemes combinatoris.
- Concretament, seguint l'exemple de **minColoring**, resoldre **basket** i **factory**. Per a **factory**, s'adjunten tres exemples d'entrada. Dels que tenen *maybe* al nom, no en coneixem l'òptim.

Referències

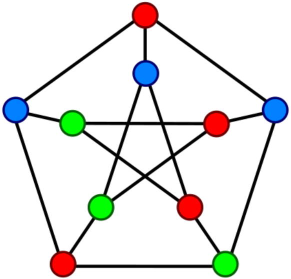
Com a guia d'estudi teniu

- 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
```

Factory

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% We have a factory of concrete products (beams, walls, roofs) that
%% works permanently (168h/week).  Every week we plan our production
%% tasks for the following week.  For example, one task may be to produce
%% a concrete beam of a certain type, which takes 10 hours and requires
%% (always one single unit of) the following resources: platform, crane,
%% truck, mechanic, driver.  But there are only a limited amount of units
%% of each resource available. For example, we may have only 3 trucks.  We
%% have 168 hours (numbered from 1 to 168) for all tasks, but we want to
%% finish all tasks as soon as possible.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Factory

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

task(T) :-              task(T,_,_).
duration(T,D) :-        task(T,D,_).
usesResource(T,R) :-    task(T,_,L), member(R,L).
hour(H) :-              maxHour(M), between(1,M,H).

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


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

satVariable( start(T,H) ) :- task(T), hour(H).   % "task T starts at hour H"     (MANDATORY)
% more variables will be needed....
```

Factory

File **easy152.pl**:

```
maxHour(168).

%% task( taskID, Duration, ListOFResourcesUsed ).
task(1,19,[1,2]).
task(2,52,[1,2]).
task(3,16,[1,3]).
task(4,52,[1,3]).
task(5,16,[2,3]).
task(6,20,[2,3]).
task(7,45,[2,3]).

%% resourceUnits( resourceID, NumUnitsAvailable ).
resourceUnits(1,2).
resourceUnits(2,1).
resourceUnits(3,2).
```

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

% displaySol(M):- nl, write(M), nl, nl, fail.
% No need to modify displaySol:
displaySol(_):- nl,nl,   write('    '),
    write('        10        20        30        40        50        60        70        80'),
    write('        90       100       110       120       130       140       150       160'),nl, write('    '),
    write('12345678901234567890123456789012345678901234567890123456789012345678901234567890'),
    write('1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678'),nl,fail.
displaySol(M):- task(T), writeNum2(T), member(start(T,H),M), duration(T,D),
		B is H-1, writeX(' ',B), writeX('x',D), nl, fail.
displaySol(M):- findall(T-H,member(start(T,H),M),L), sort(L,L1), write(startTimes(L1)), write('.'),  nl,nl,!.
displaySol(_).

writeX(_,0):-!.
writeX(X,N):- write(X), N1 is N-1, writeX(X,N1),!.

writeNum2(T):-T<10, write(' '), write(T), write(': '), !.
writeNum2(T):-                 write(T), write(': '), !.
```

Factory

El cost és l'hora màxima de finalització d'alguna tasca.

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

costOfThisSolution(M,Cost) :-
    % 'Cost' is the maximum task completion hour of this solution/model M
    ...


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

Basket

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Extend this Prolog source to design a Basketball League with N
%% teams, with N-1 rounds (playing days), where every two teams play  1
%% against each other on exactly one round, one team at home and the other team
%% away, and on each round each team has exactly one match (home or away).  2
%% Moreover, we say that a team has a "double" on round R if it plays
%% at home on rounds R-1 and on round R, or if it plays away on R-1 and on R.
%% No "triples" are allowed: no three consecutive homes, nor three aways.
%% Minimize the number of doubles of the team with the largest number of doubles.
%%
%% Additional constraints (see the input example below):
%% 1. No doubles on certain rounds
%% 2. Movistar has bought the tv rights for Sunday 8pm for all
%%    matches among a group of teams (the so-called tv Teams) and wants
%%    on every round at least one match between two tv Teams.
%% 3. On certain rounds certain teams cannot play at home.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Basket

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

numTeams(14).               % This number is always even.
noDoubles([2,8,13]).        % No team has a double on any of these rounds.
tvTeams([1,2,3,4,5,6]).     % The list of tv teams.
notHome( 1, [2,5,7,9,10]).  % Team 1 cannot play at home on round 2, also not on round 5, etc.
notHome( 2, [4,6,8,10]).
notHome( 3, [2,3,5,7,9,10]).
notHome( 4, [4,6,8,12]).
notHome( 5, [1,3,12]).
notHome( 6, [1,3,5,7,10]).
notHome( 7, [1,3,5,7,9]).
notHome( 8, [1,3,5,7,9]).
notHome( 9, [1,4,8,10]).
notHome(10, [2,4,8,9,11]).
notHome(11, [2,4,8,12]).
notHome(12, [6]).
notHome(13, [6,10,11,13]).
notHome(14, [2,4]).

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

Basket

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

team(T):- numTeams(N), between(1,N,T).
difTeams(S,T):- team(S), team(T), S\=T.
round(R):- numTeams(N), N1 is N-1, between(1,N1,R).
tvMatch(S,T):- tvTeams(TV), member(S,TV), member(T,TV), S\=T.
away(T,R):- notHome(T,L), member(R,L).

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


%%%%%%%  1. Declare SAT variables to be used: =================================================
%%          It is mandatory to use these variables!
satVariable( match(S,T,R) ):- team(S), team(T), round(R). % "on round R there is a match S-T at home of S"
satVariable( home(S,R)    ):- team(S),          round(R). % "team S plays at home on round R"
satVariable( double(S,R)  ):- team(S),          round(R). % "team S has a double on round R"
```

Basket

```
%%%%%%% 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(MaxCost):-
    eachTeamEachRoundExactlyOneMatch,
    ...
    maxCost(MaxCost),
    true,!.
writeClauses(_):- told, nl, write('writeClauses failed!'), nl,nl, halt.

eachTeamEachRoundExactlyOneMatch:- team(T), round(R),
    findall( match(S,T,R), difTeams(S,T), LitsH ),
    findall( match(T,S,R), difTeams(S,T), LitsA ), append(LitsH,LitsA,Lits), exactly(1,Lits), fail.
eachTeamEachRoundExactlyOneMatch.
```

el cost MaxCost només es té en compte
en el predicat maxCost

cada equip té <= Max dobles

```
maxCost(infinite):-!.
maxCost(Max):- team(T), findall(double(T,R), round(R), Lits ), atMost(Max,Lits), fail.
maxCost(_).
```

Basket

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

costOfThisSolution(M,Cost) :-
    numTeams(N), N1 is N-1, between(0,N1,I), Cost is N1-I,
    ...  % Some team has 'Cost' doubles in this solution/model

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