clique and colour

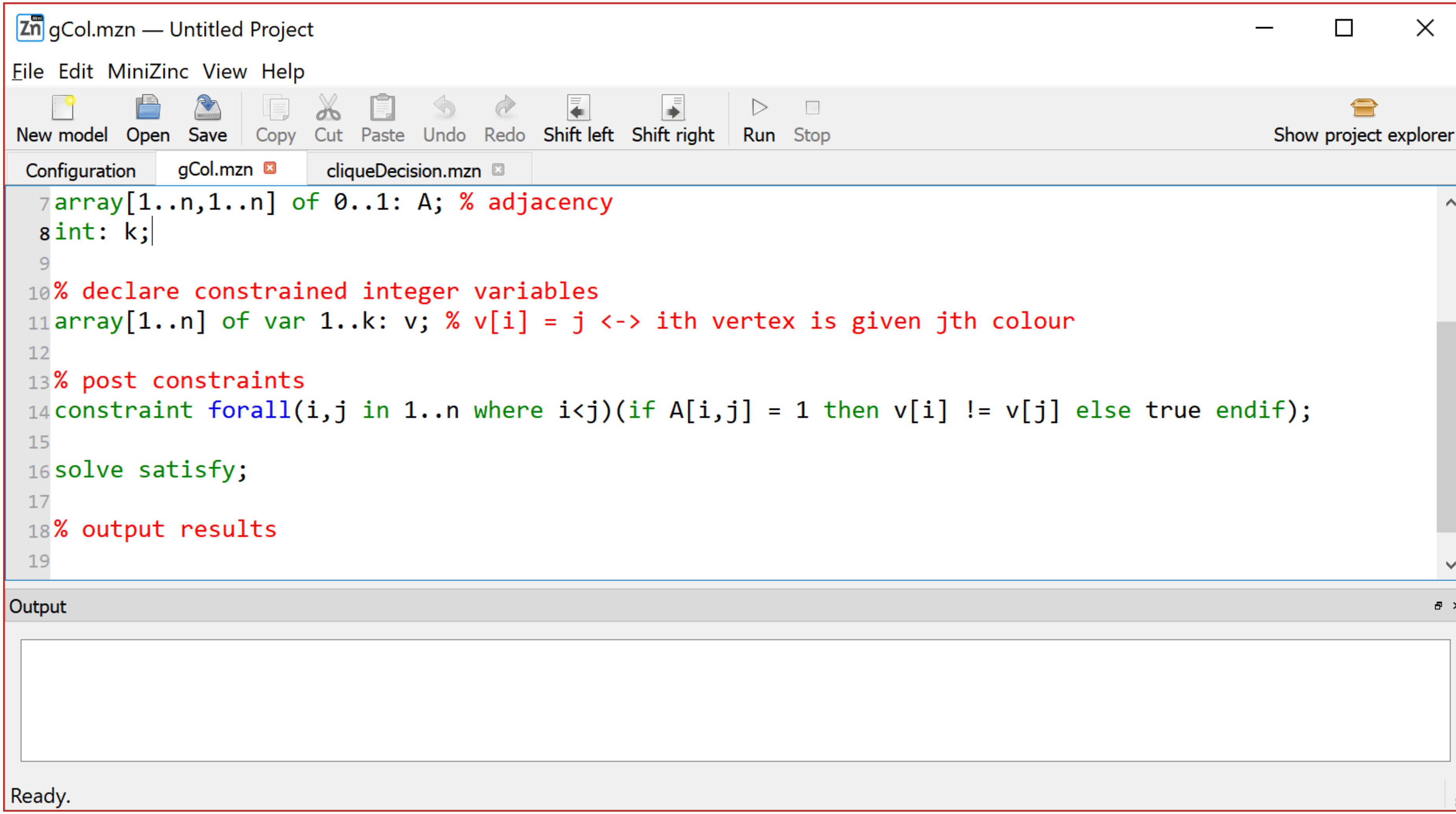
Part 1: introduction to two core problems
Part 2: a study of branch and bound

Part 1: colour & clique

gCol

gCol: Given a graph G = (V,E) and an integer k, can the graph be coloured with k colours such that adjacent vertices take different colours?

Graph Colouring



g10

g10.dzn — Untitled Project

File Edit MiniZinc View Help

New model Open Save Copy Cut Paste Undo Redo

Configuration g10.dzn

```
n = 10;
A = [|0,1,0,0,1,0,0,0,0,1
     |1,0,0,1,1,1,0,0,0,0
     |0,0,0,1,0,1,1,0,0,0
     |0,1,1,0,1,0,1,0,1,1
     |1,1,0,1,0,1,1,0,1,0
     |0,1,1,0,1,0,0,1,0,0
     |0,0,1,1,1,0,0,0,0,0
     |0,0,0,0,0,1,0,0,1,0
     |0,0,0,1,1,0,0,1,0,0
     |1,0,0,1,0,0,0,0,0,0|];
```

Output

Ready.

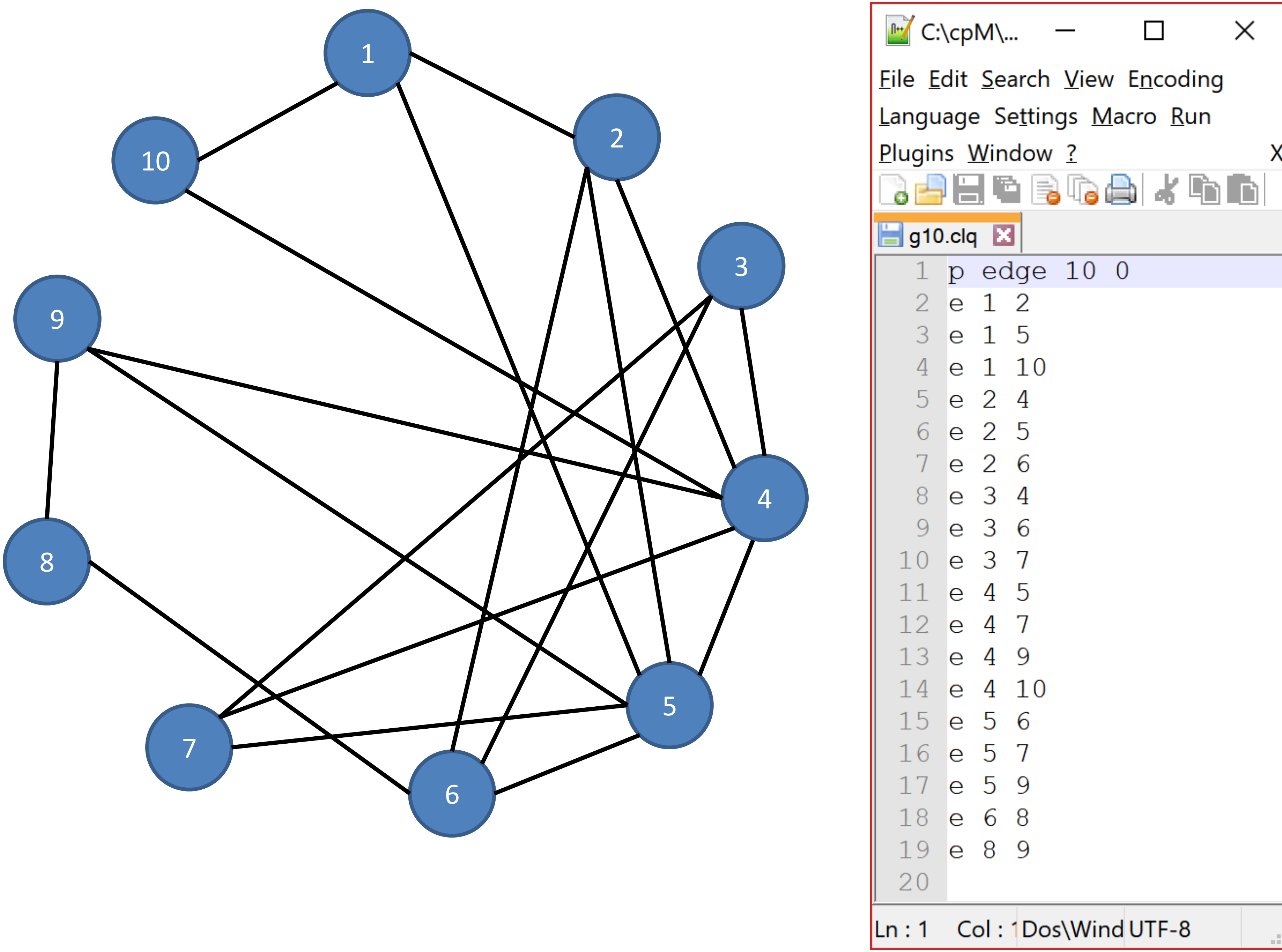
C:\cpM\...

File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X

g10.clq

```
p edge 10 0
e 1 2
e 1 5
e 1 10
e 2 4
e 2 5
e 2 6
e 3 4
e 3 6
e 3 7
e 4 5
e 4 7
e 4 9
e 4 10
e 5 6
e 5 7
e 5 9
e 6 8
e 8 9
```

Ln : 1 Col : 1 Dos\Wind UTF-8


1
2
3
4
5
6
7
8
9
10
C:\cpM\...
File Edit Search View Encoding
Language Settings Macro Run
Plugins Window ?
X
g10.clq
p edge 10 0
e 1 2
e 1 5
e 1 10
e 2 4
e 2 5
e 2 6
e 3 4
e 3 6
e 3 7
e 4 5
e 4 7
e 4 9
e 4 10
e 5 6
e 5 7
e 5 9
e 6 8
e 8 9
Ln : 1 Col : 1 Dos\Wind UTF-8

Graph Colouring

Command Prompt

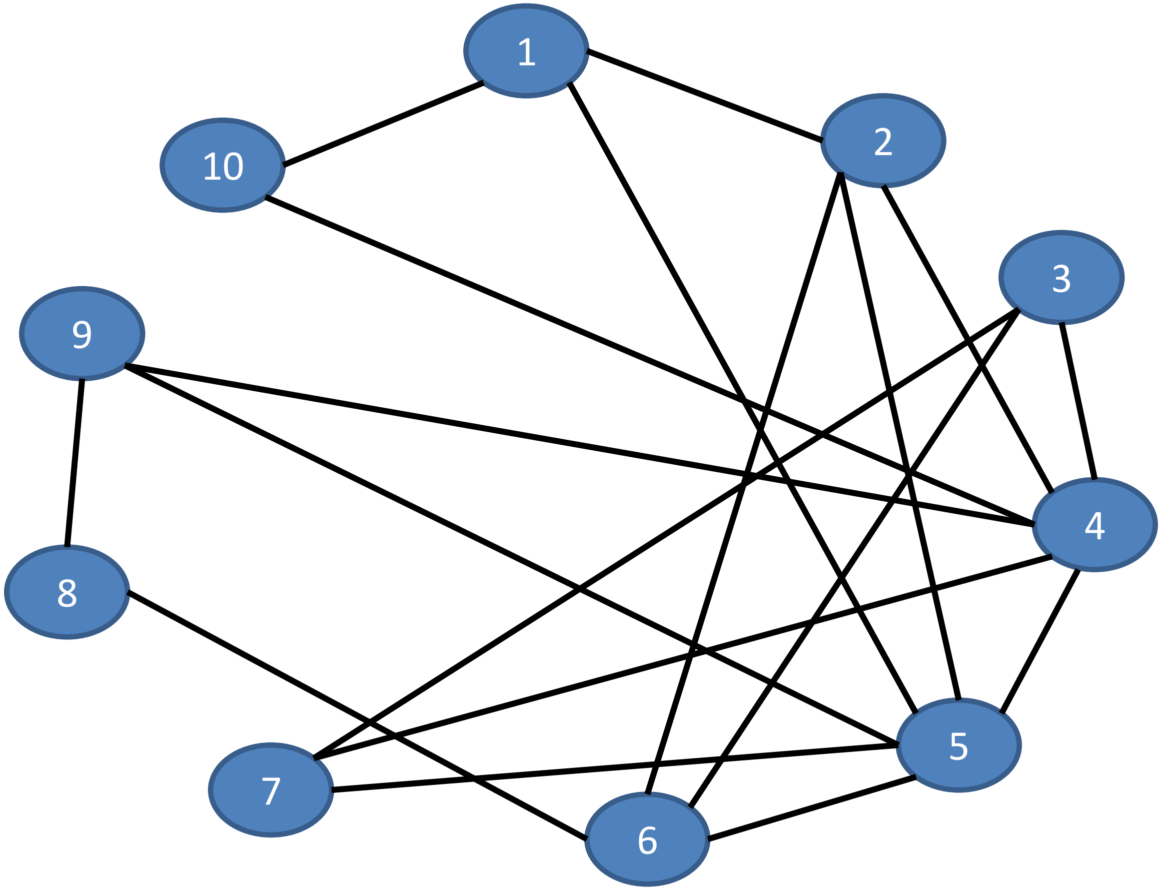
```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=4"
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
----------

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
----------

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=2"
=====UNSATISFIABLE=====

C:\cpM\minizincCPM\cliqueAndColour>
```

Graph Colouring

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=3" -a
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
----------
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 3]);
----------
v = array1d(1..10 ,[3, 2, 1, 3, 1, 3, 2, 1, 2, 1]);
----------
v = array1d(1..10 ,[3, 2, 1, 3, 1, 3, 2, 1, 2, 2]);
----------
v = array1d(1..10 ,[1, 3, 2, 1, 2, 1, 3, 2, 3, 2]);
----------
v = array1d(1..10 ,[1, 3, 2, 1, 2, 1, 3, 2, 3, 3]);
----------
v = array1d(1..10 ,[3, 1, 2, 3, 2, 3, 1, 2, 1, 1]);
----------
v = array1d(1..10 ,[3, 1, 2, 3, 2, 3, 1, 2, 1, 2]);
----------
v = array1d(1..10 ,[1, 2, 3, 1, 3, 1, 2, 3, 2, 2]);
----------
v = array1d(1..10 ,[1, 2, 3, 1, 3, 1, 2, 3, 2, 3]);
----------
v = array1d(1..10 ,[2, 1, 3, 2, 3, 2, 1, 3, 1, 1]);
----------
v = array1d(1..10 ,[2, 1, 3, 2, 3, 2, 1, 3, 1, 3]);
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

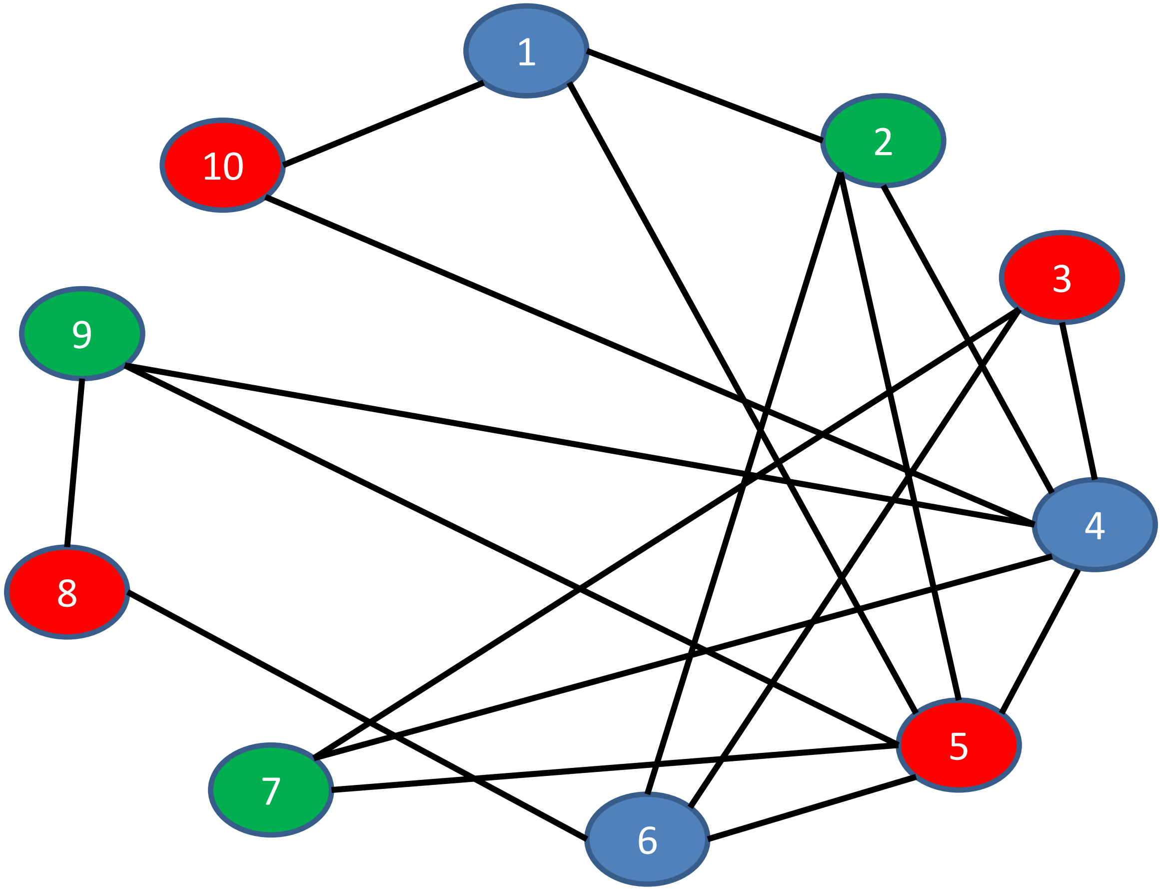
g10

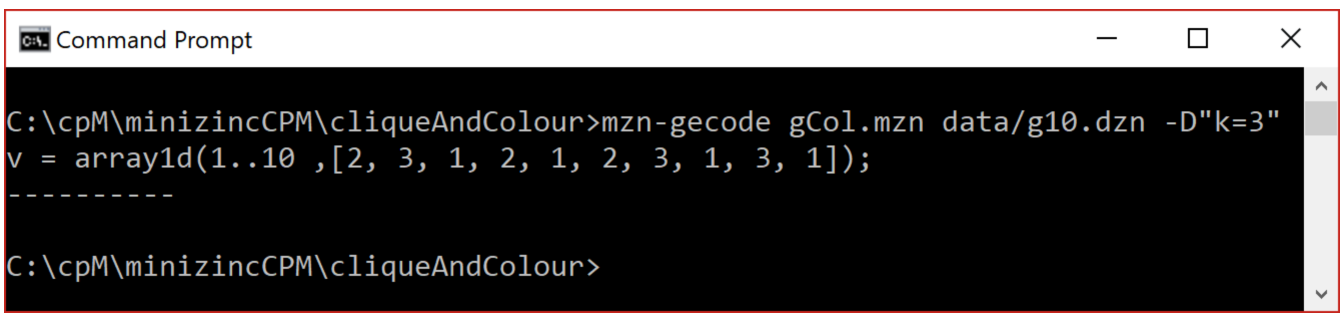


Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
----------

C:\cpM\minizincCPM\cliqueAndColour>
```

g10






```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode gCol.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
----------

C:\cpM\minizincCPM\cliqueAndColour>
```

optimisation

chromatic number (chi)

colOpt.mzn — Untitled Project

```
%
% colOpt: comput chromatic number (minimum number of colours required)
%

int: n;
array[1..n,1..n] of 0..1: A; % adjacency

% declare constrained integer variables
array[1..n] of var 1..n: v; % v[i] = j <-> ith vertex is given jth colour

% post constraints
constraint forall(i,j in 1..n where i<j)(if A[i,j] = 1 then v[i] != v[j] else true endif);

var 1..n: chi; % the chromatic number
constraint chi = max(v);

solve minimize chi;
```

chromatic number (chi)

```
var 1..n: chi; % the chromatic number
constraint chi = max(v);

solve minimize chi;
```

chromatic number (chi)

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g10.dzn
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
chi = 3;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

chromatic number (chi)

Command Prompt

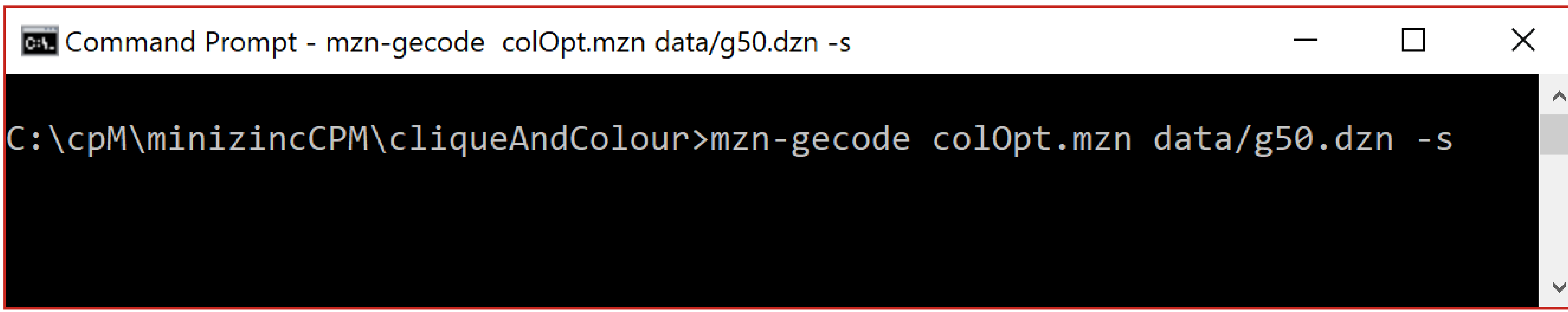
```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g10.dzn -a -s
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
chi = 3;
----------
==========
%%  runtime:       0.006 (6.000 ms)
%%  solvetime:     0.000 (0.000 ms)
%%  solutions:     1
%%  variables:     11
%%  propagators:   19
%%  propagations:  54
%%  nodes:         13
%%  failures:      5
%%  restarts:      0
%%  peak depth:    10

C:\cpM\minizincCPM\cliqueAndColour>
```

g40.dzn

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g40.dzn -s
v = array1d(1..40 ,[5, 6, 7, 4, 4, 8, 6, 3, 1, 5, 2, 7, 6, 3, 1, 8, 6, 1, 4, 2, 5, 3, 4, 4, 6, 1, 3,
 2, 5, 7, 2, 8, 3, 8, 5, 5, 5, 8, 7, 4]);
chi = 8;
----------
==========
%%  runtime:       9.886 (9886.000 ms)
%%  solvetime:     9.878 (9878.000 ms)
%%  solutions:     3
%%  variables:     41
%%  propagators:   377
%%  propagations:  12956547
%%  nodes:         280452
%%  failures:      140217
%%  restarts:      0
%%  peak depth:    40

C:\cpM\minizincCPM\cliqueAndColour>
```

g50.dzn



Don’t hold your breath

Think on this … will all gCol instances be hard? What will affect difficulty?

- Number of vertices (the order of the graph)?
- Number of edges (the size of the graph)?
- The number of colours allowed?

Think on this … are there things we can do to help tackle gCol?

- Symmetry breaking?
- The order we choose to colour vertices?
- Problem reductions?

Isolation Ordering of Vertices

```
int: n;
array[1..n,1..n] of 0..1: A; % adjacency
array[1..n] of 0..n-1: degree = [sum(j in 1..n)(1-A[i,j]) | i in 1..n];

% declare constrained integer variables
array[1..n] of var 1..n: v; % v[i] = j <-> ith vertex is given jth colour

% post constraints
constraint forall(i,j in 1..n where i<j)(if A[i,j] = 1 then v[i] != v[j] else true endif);

var 1..n: chi; % the chromatic number
constraint chi = max(v);

%solve minimize chi;
%solve::int_search(sort_by(v,degree),smallest,indomain_max,complete) minimize chi;
solve::int_search(v,smallest,indomain_max,complete) minimize chi;
```

G40 isolation ordering

Command Prompt

```
chi = 16;
----------
v = array1d(1..40 ,[15, 14, 15, 13, 15, 14, 14, 12, 14, 13, 11, 12, 12, 13, 10, 11, 14, 10, 11, 9, 15, 12,
 7, 6, 7, 13, 13, 9, 6]);
chi = 15;
----------
v = array1d(1..40 ,[14, 13, 14, 12, 14, 13, 13, 11, 13, 12, 10, 11, 11, 12, 9, 10, 13, 9, 10, 8, 14, 11, 9
5, 6, 12, 12, 8, 5]);
chi = 14;
----------
v = array1d(1..40 ,[13, 12, 13, 11, 13, 12, 12, 10, 12, 11, 9, 10, 10, 11, 8, 9, 12, 8, 9, 7, 13, 10, 8, 9
, 11, 11, 7, 4]);
chi = 13;
----------
v = array1d(1..40 ,[12, 11, 12, 10, 12, 11, 11, 9, 11, 10, 8, 9, 9, 10, 7, 8, 11, 7, 8, 6, 12, 9, 7, 8, 11
 10, 6, 3]);
chi = 12;
----------
v = array1d(1..40 ,[11, 10, 11, 9, 11, 10, 10, 8, 10, 9, 7, 8, 8, 9, 6, 7, 10, 6, 7, 5, 11, 8, 6, 7, 10, 6
5, 2]);
chi = 11;
----------
v = array1d(1..40 ,[10, 9, 10, 8, 10, 9, 9, 7, 9, 8, 6, 7, 7, 8, 5, 6, 9, 5, 6, 4, 10, 7, 5, 6, 9, 5, 4, 3
;
chi = 10;
----------
v = array1d(1..40 ,[9, 8, 9, 7, 9, 8, 8, 6, 8, 7, 5, 6, 6, 7, 4, 5, 8, 4, 5, 3, 2, 6, 4, 1, 8, 4, 9, 3, 7,
chi = 9;
----------
v = array1d(1..40 ,[8, 7, 8, 6, 8, 7, 5, 4, 3, 7, 2, 3, 5, 4, 3, 2, 5, 3, 2, 6, 8, 4, 1, 2, 5, 7, 4, 5, 8,
chi = 8;
----------
==========
%%  runtime:       5:45:22.792 (20722792.000 ms)
%%  solvetime:     5:45:22.777 (20722777.000 ms)
%%  solutions:     33
%%  variables:     41
%%  propagators:   377
%%  propagations:  3541249153
%%  nodes:         4106979452
%%  failures:      2053489286
%%  restarts:      0
%%  peak depth:    63

Z:\public_html\cpM\minizincCPM\cliqueAndColour>_
```

Warning: graph colouring can take over your life.

clique

Clique

clique: Given a graph $G = (V,E)$ and an integer k, is there a subset of the vertices, of size k, such that all vertices in that set are pair-wise adjacent

Clique

cliqueDecision.mzn — Untitled Project*

```
%
% Clique decision
%

int: n;
array[1..n,1..n] of 0..1: A; % adjacency
int: k;

% declare constrained integer variables
array[1..n] of var 0..1: v; % v[i] = 1 <-> ith vertex is in the clique

% declare constraints
constraint forall(i,j in 1..n where i<j)(if A[i,j] = 0 then v[i] + v[j] < 2 else true endif);
constraint sum(v) >= k;

solve satisfy;

```

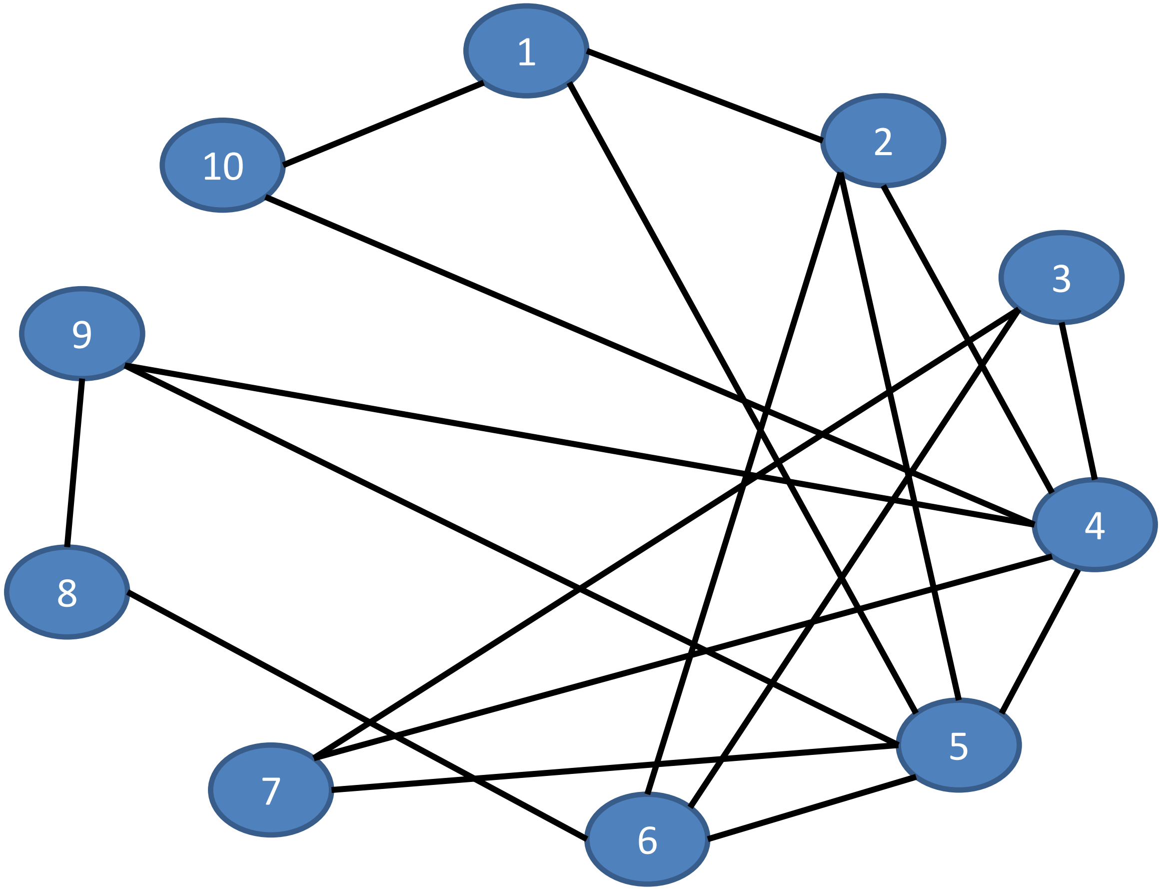
Output

Clique

g10.dzn

Command Prompt

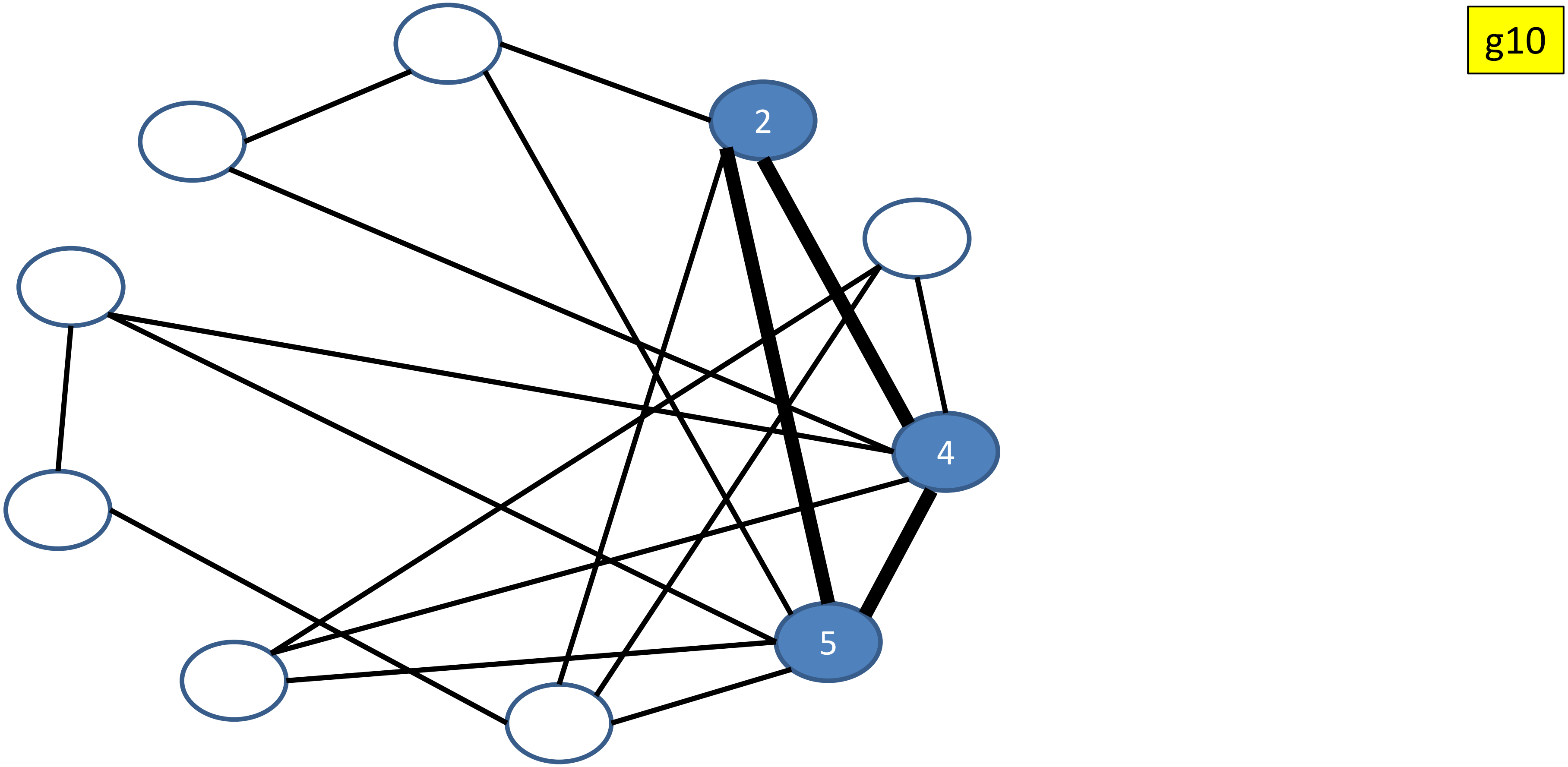
```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=5"
=====UNSATISFIABLE=====

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=4"
=====UNSATISFIABLE=====

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);
----------

C:\cpM\minizincCPM\cliqueAndColour>_
```



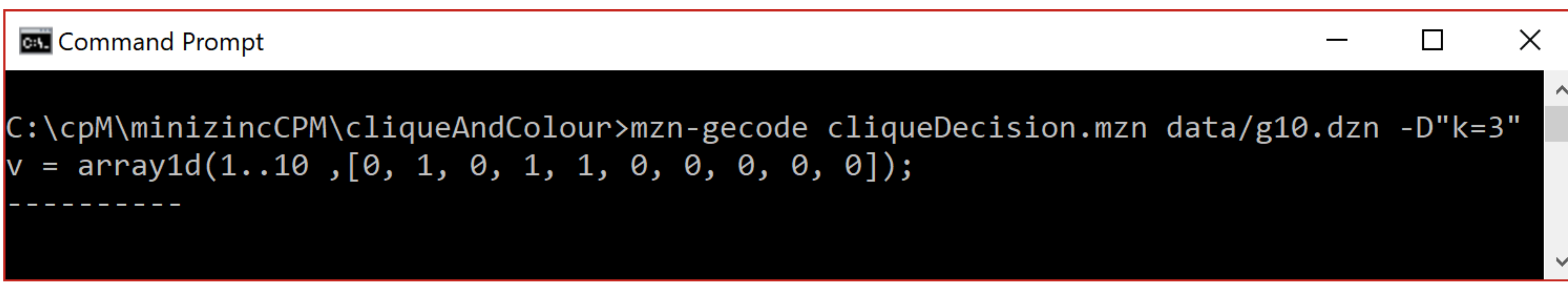
g10

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);
----------
```



g10
2
4
5



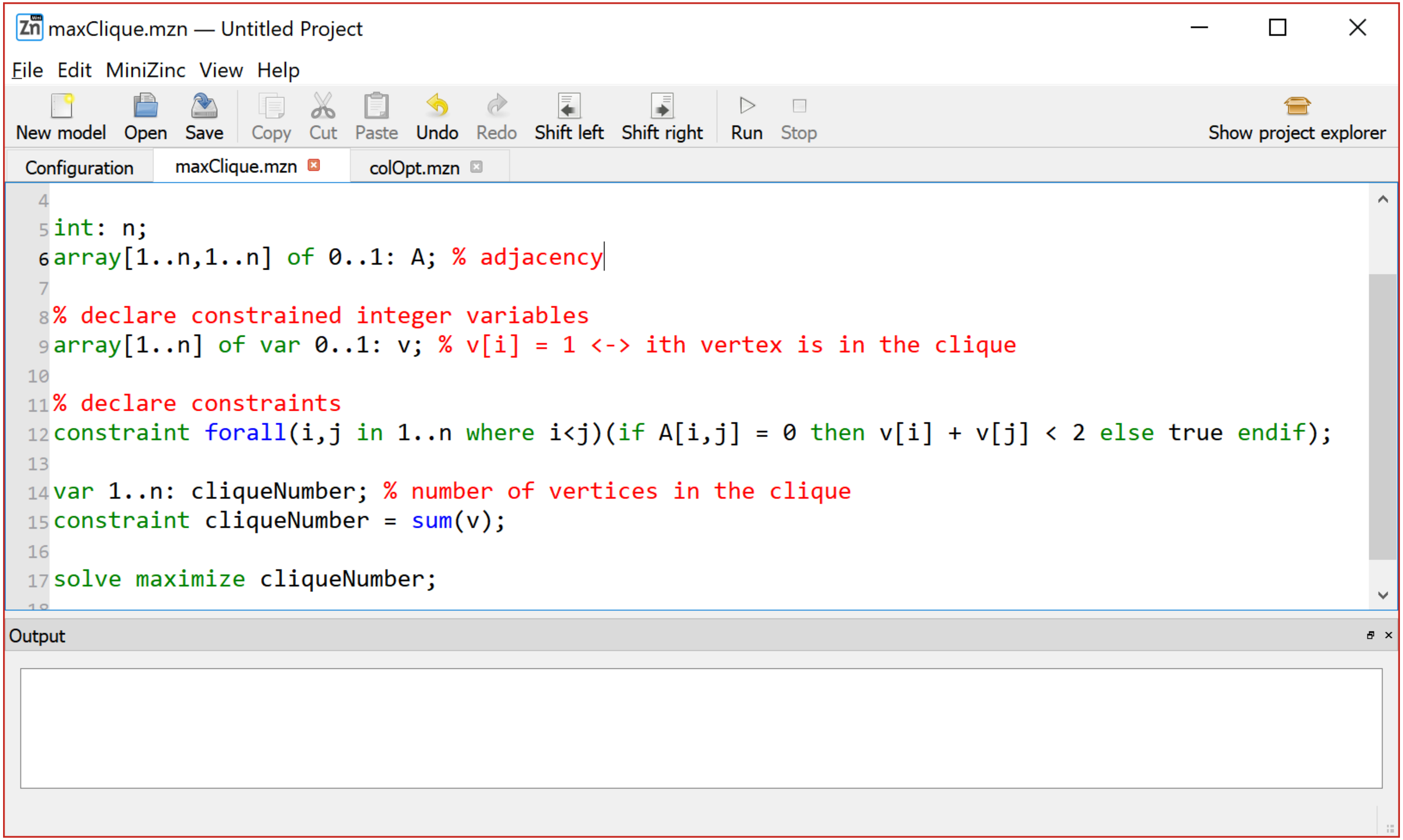
Command Prompt
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=3"
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);

Solution is not unique

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode cliqueDecision.mzn data/g10.dzn -D"k=3" -a
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);
----------
v = array1d(1..10 ,[0, 1, 0, 0, 1, 1, 0, 0, 0, 0]);
----------
v = array1d(1..10 ,[0, 0, 1, 1, 0, 0, 1, 0, 0, 0]);
----------
v = array1d(1..10 ,[0, 0, 0, 1, 1, 0, 1, 0, 0, 0]);
----------
v = array1d(1..10 ,[1, 1, 0, 0, 1, 0, 0, 0, 0, 0]);
----------
v = array1d(1..10 ,[0, 0, 0, 1, 1, 0, 0, 0, 1, 0]);
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

optimisation


maxClique.mzn — Untitled Project
File Edit MiniZinc View Help
New model Open Save Copy Cut Paste Undo Redo Shift left Shift right Run Stop
Show project explorer
Configuration
maxClique.mzn
colOpt.mzn
int: n;
array[1..n,1..n] of 0..1: A; % adjacency
% declare constrained integer variables
array[1..n] of var 0..1: v; % v[i] = 1 <-> ith vertex is in the clique
% declare constraints
constraint forall(i,j in 1..n where i<j)(if A[i,j] = 0 then v[i] + v[j] < 2 else true endif);
var 1..n: cliqueNumber; % number of vertices in the clique
constraint cliqueNumber = sum(v);
solve maximize cliqueNumber;
Output

```
var 1..n: cliqueNumber; % number of vertices in the clique
constraint cliqueNumber = sum(v);

solve maximize cliqueNumber;
```

g10.dzn

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g10.dzn
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);
cliqueNumber = 3;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

g40.dzn

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g40.dzn
v = array1d(1..40 ,[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]);
cliqueNumber = 7;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

g100.dzn

Command Prompt

```
C
:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g100.dzn  -s
v = array1d(1..100 ,[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0
, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
;
cliqueNumber = 9;
----------
==========
%%  runtime:       0.238 (238.000 ms)
%%  solvetime:     0.214 (214.000 ms)
%%  solutions:     9
%%  variables:     101
%%  propagators:   2428
%%  propagations:  780716
%%  nodes:         22563
%%  failures:      11273
%%  restarts:      0
```

g200.dzn

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g200.dzn -s
v = array1d(1..200 ,[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]);
cliqueNumber = 11;
----------
==========
%%  runtime:       9.010 (9010.000 ms)
%%  solvetime:     8.937 (8937.000 ms)
%%  solutions:     11
%%  variables:     201
%%  propagators:   9986
%%  propagations:  26431102
%%  nodes:         591891
%%  failures:      295935
%%  restarts:      0
%%  peak depth:    199

C:\cpM\minizincCPM\cliqueAndColour>
```

brock200_2.dzn

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/brock200_2.dzn -s
v = array1d(1..200 ,[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]);
cliqueNumber = 12;
----------
==========
%%  runtime:       8.615 (8615.000 ms)
%%  solvetime:     8.466 (8466.000 ms)
%%  solutions:     12
%%  variables:     201
%%  propagators:   10025
%%  propagations:  25233256
%%  nodes:         546573
%%  failures:      273275
%%  restarts:      0
%%  peak depth:    199

C:\cpM\minizincCPM\cliqueAndColour>
```

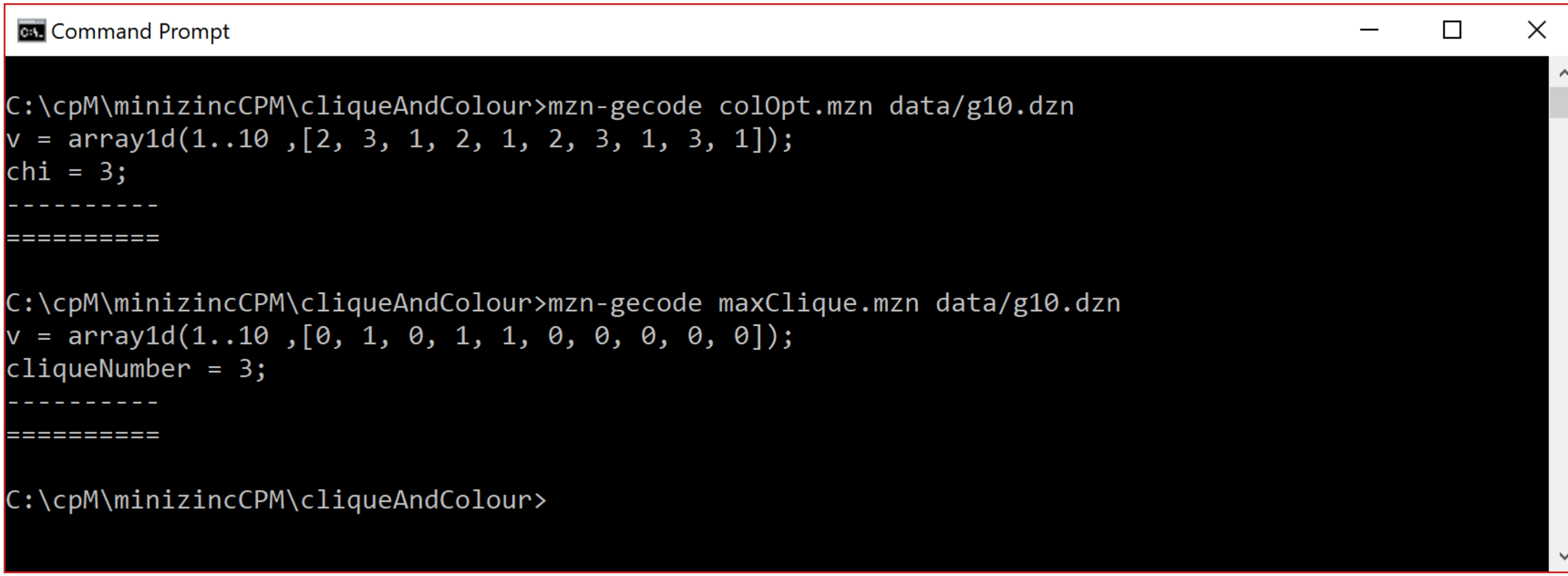
graph colouring and clique … are they related?

g05

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g05.dzn
v = array1d(1..5 ,[1, 2, 1, 3, 3]);
chi = 3;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g05.dzn
v = array1d(1..5 ,[0, 1, 1, 1, 0]);
cliqueNumber = 3;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

g10


Command Prompt
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g10.dzn
v = array1d(1..10 ,[2, 3, 1, 2, 1, 2, 3, 1, 3, 1]);
chi = 3;

==========
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g10.dzn
v = array1d(1..10 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0]);
cliqueNumber = 3;

==========
C:\cpM\minizincCPM\cliqueAndColour>

g20

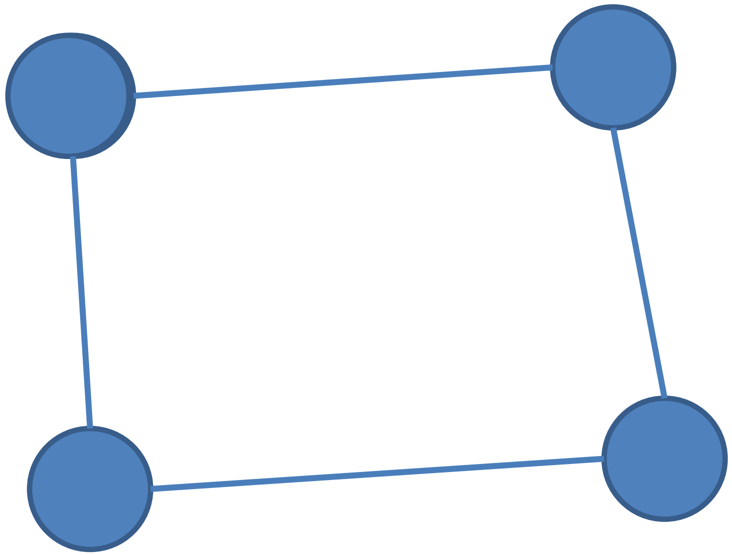
Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g20.dzn
v = array1d(1..20 ,[5, 2, 1, 3, 1, 3, 1, 5, 3, 4, 2, 3, 2, 4, 2, 4, 4, 5, 5, 4]);
chi = 5;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g20.dzn
v = array1d(1..20 ,[0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]);
cliqueNumber = 5;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```

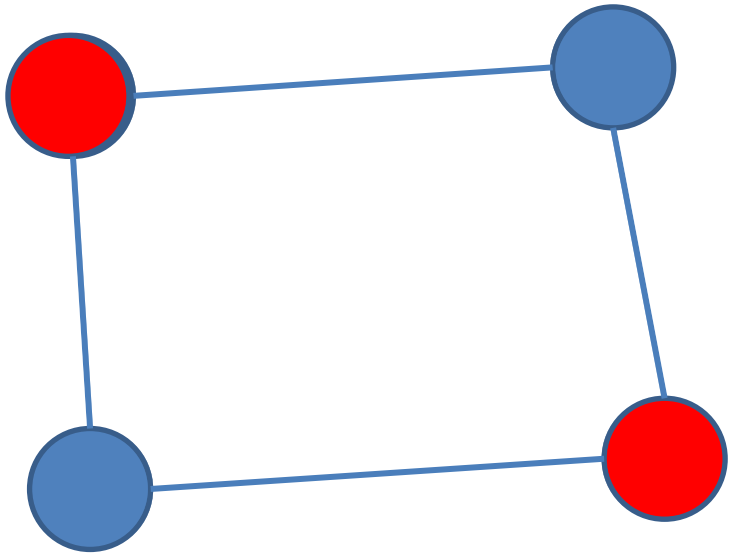
g30

Command Prompt

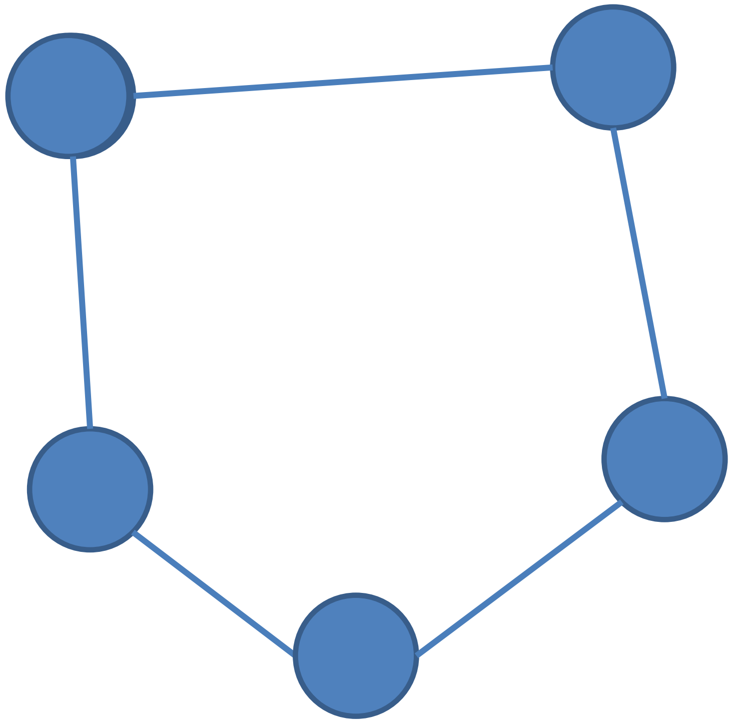
```
C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode colOpt.mzn data/g30.dzn
v = array1d(1..30 ,[1, 5, 3, 5, 7, 7, 6, 4, 4, 2, 2, 3, 6, 6, 6, 1, 4, 6, 5, 5, 3, 1, 2, 3, 2, 2, 7
, 7, 5, 1]);
chi = 7;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/g30.dzn
v = array1d(1..30 ,[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1
, 0, 0, 0]);
cliqueNumber = 6;
----------
==========

C:\cpM\minizincCPM\cliqueAndColour>
```



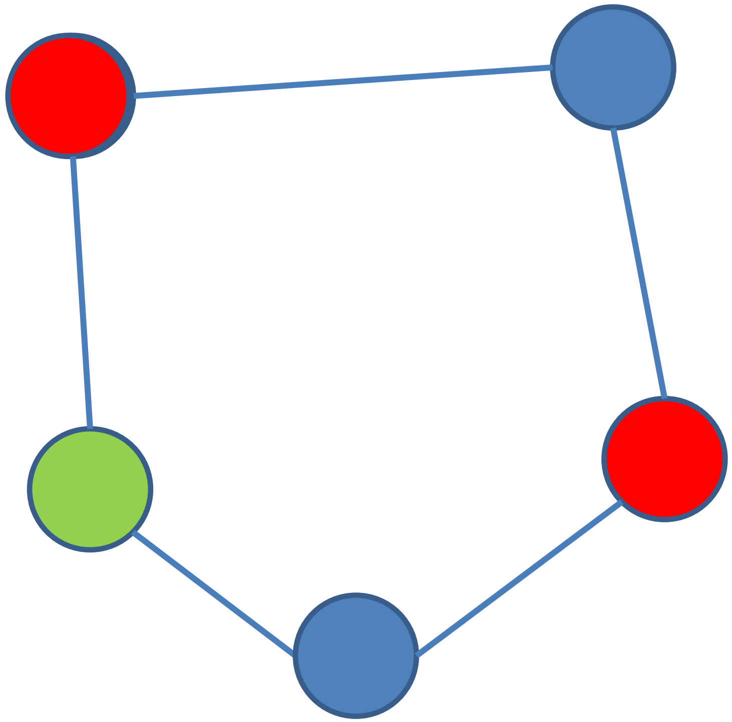
What is clique number and what is colour number?



What is clique number and what is colour number?



What is clique number and what is colour number?



What is clique number and what is colour number?

Part 2: branch and bound (BnB) using max clique (MC) as motivation

A sample of what's to come

MiniZinc

brock200_4

```
Command Prompt

C:\cpM\minizincCPM\cliqueAndColour>mzn-gecode maxClique.mzn data/brock200_4.dzn -s
v = array1d(1..200 ,[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]);
cliqueNumber = 17;
----------
==========
%%  runtime:       4:35.056 (275056.000 ms)
%%  solvetime:     4:34.919 (274919.000 ms)
%%  solutions:     17
%%  variables:     201
%%  propagators:   6812
%%  propagations:  516927031
%%  nodes:         16008643
%%  failures:      8004305
%%  restarts:      0
%%  peak depth:    199

C:\cpM\minizincCPM\cliqueAndColour>
```

274,919 milliseconds
16,008,643 nodes

A sample of what's to come

BnB

brock200_4

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC0 ../data/brock200_4.clq
17 633542730 107632
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC1 ../data/brock200_4.clq
17 14318331 8120
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MCSa1 ../data/brock200_4.clq
17 58730 711
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique BBMC1 ../data/brock200_4.clq
17 58730 229
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>_
```

mzn: 274,919 ms
java: 229 ms
speedup: 1,200

Small scale statistics

How did we do that?

algorithmic sketch
staring with MC0
heading to BBMC

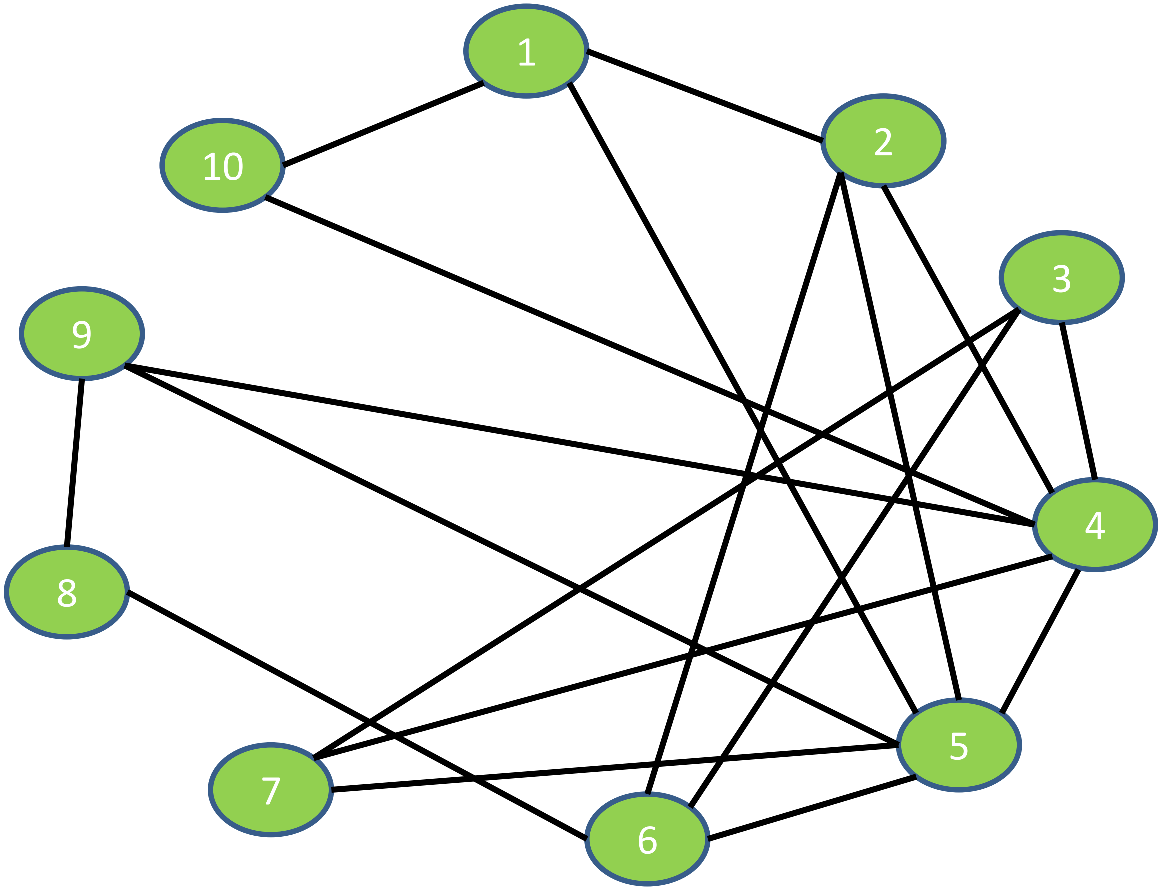
We have two sets …

C: is the growing clique, a set of vertices

P: is the candidate set, i.e. set of vertices that we ***might*** add to C

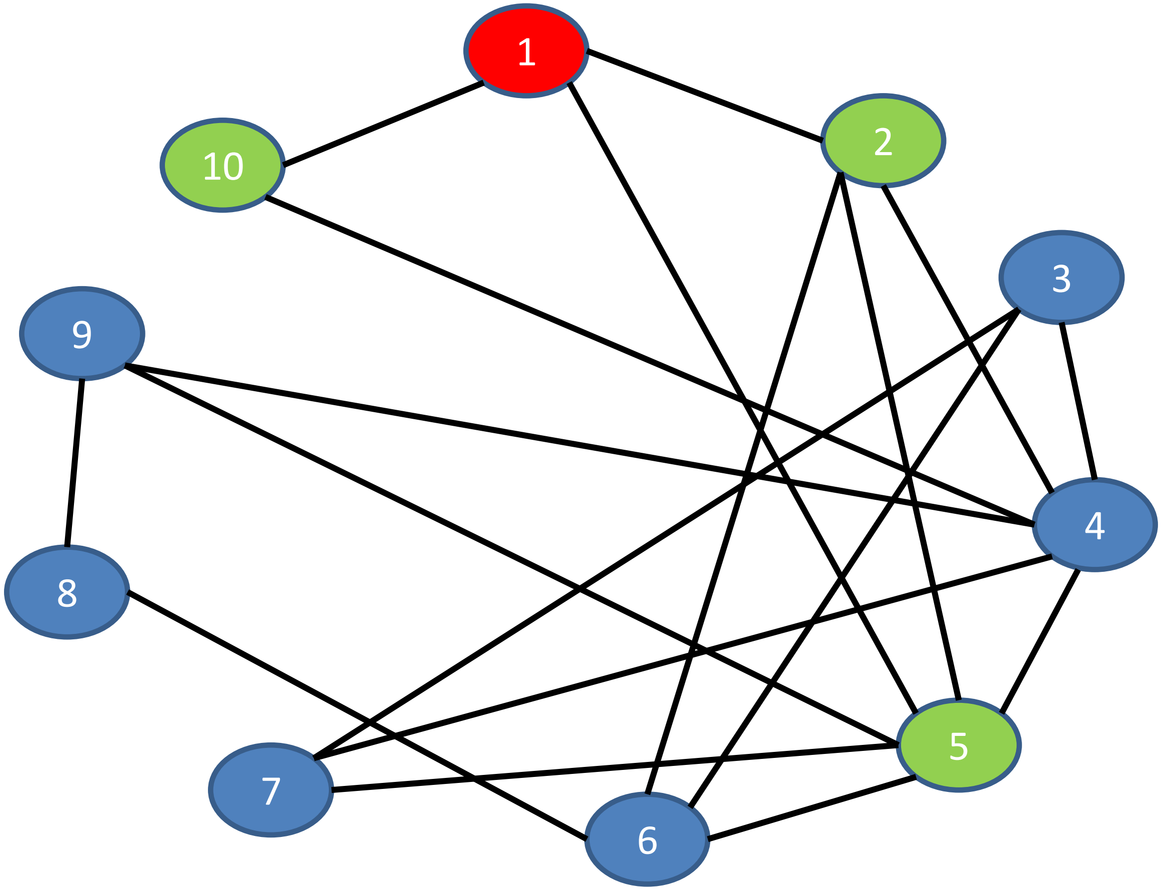
The largest clique found so far is called the ***incumbent*** and has a size ***maxSize***

MC0



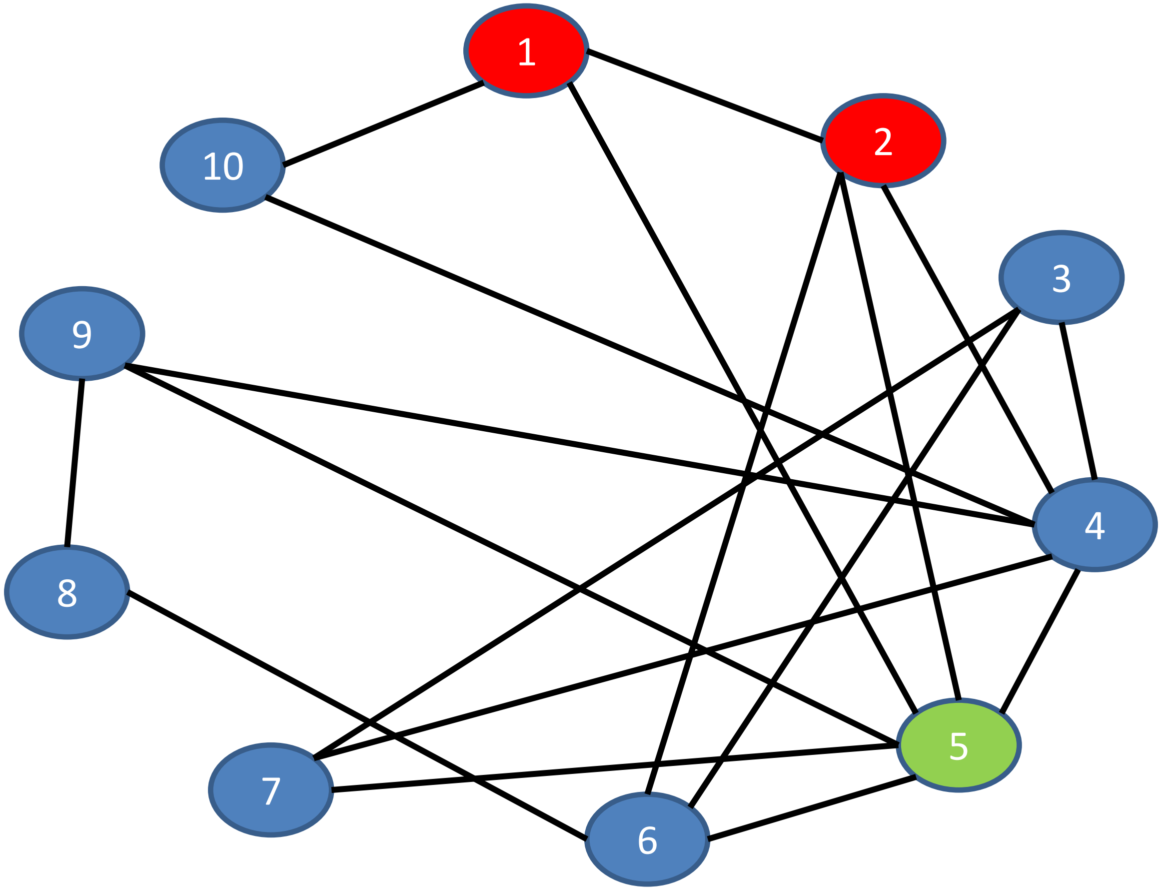
C = {}
P = {1,2,3,4,5,6,7,8,9,10}
maxSize = 0
incumbent = {}

Top of search



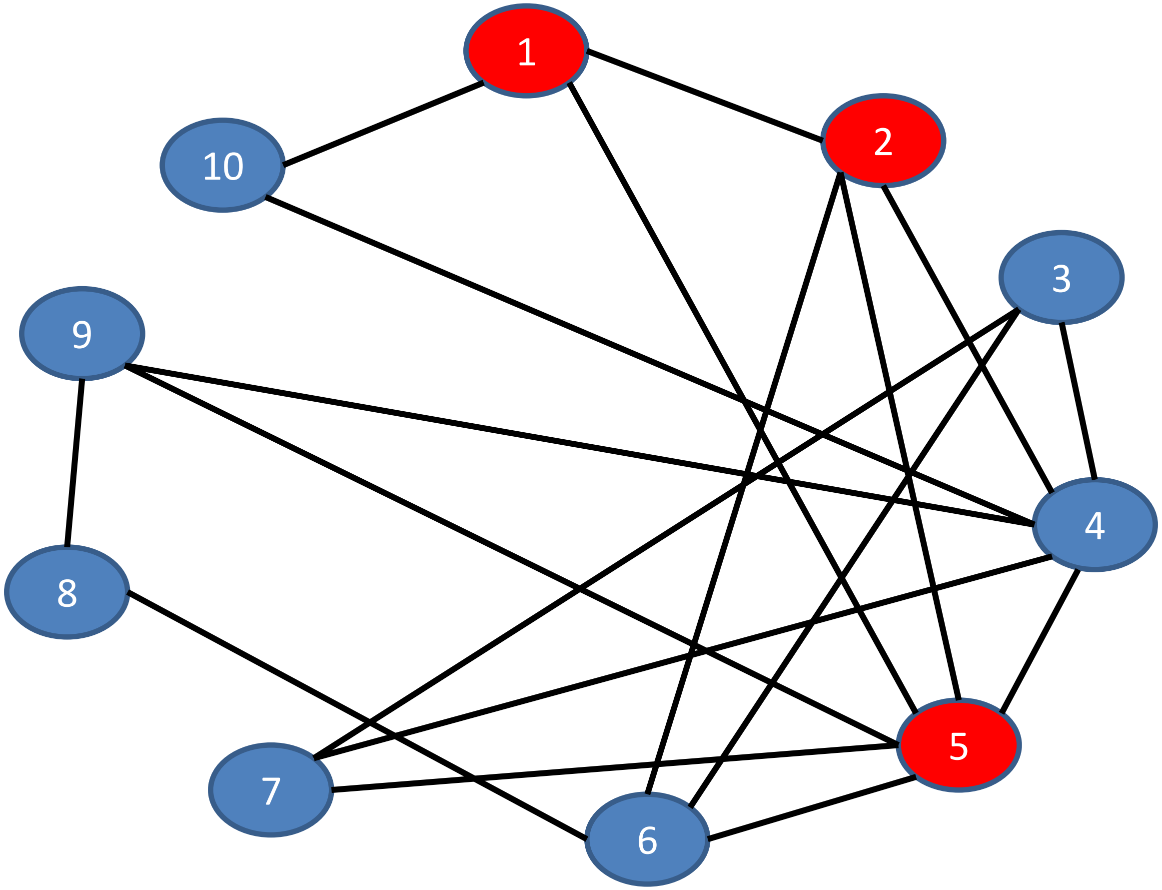
C = {1}
P = {2,5,10}
maxSize = 0
incumbent = {}

Select vertex 1



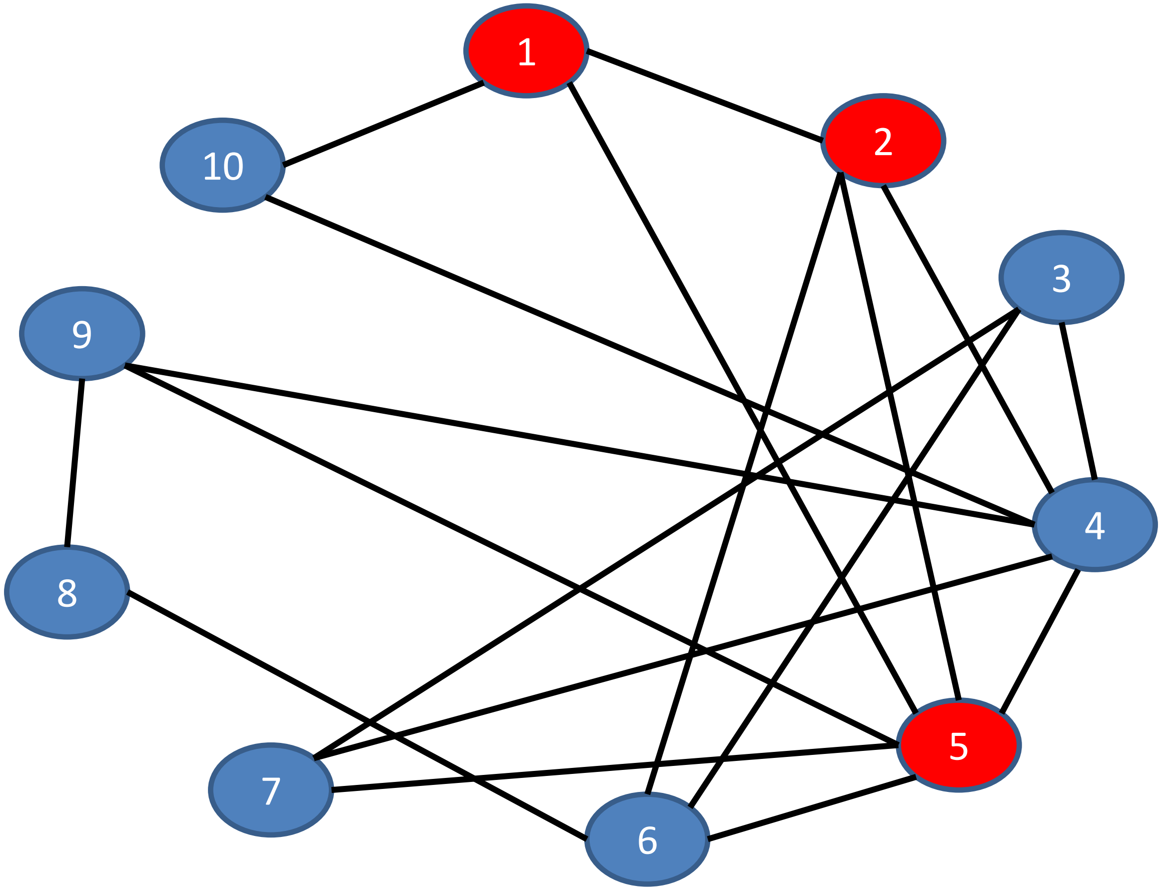
C = {1,2}
P = {5}
maxSize = 0
incumbent = {}

Select vertex 2



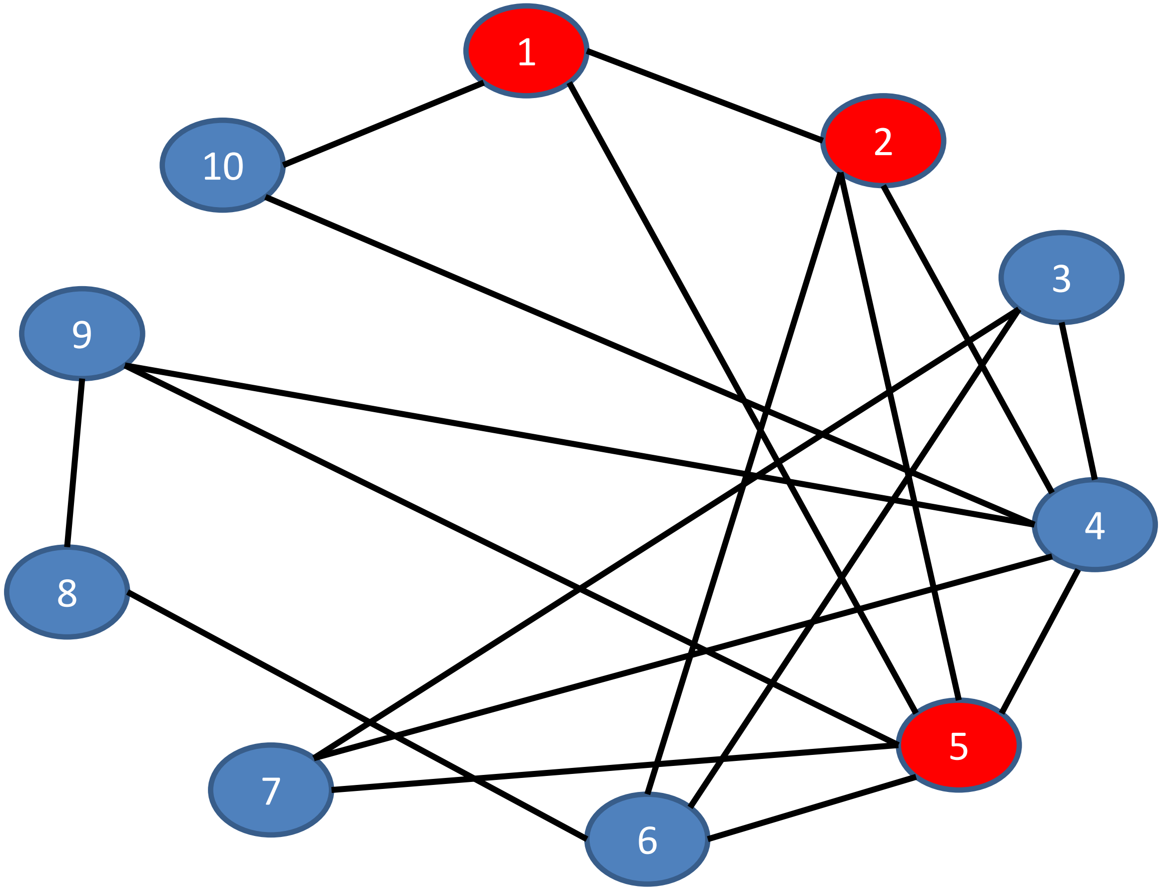
C = {1,2,5}
P = {}
maxSize = 0
incumbent = {}

Select vertex 5



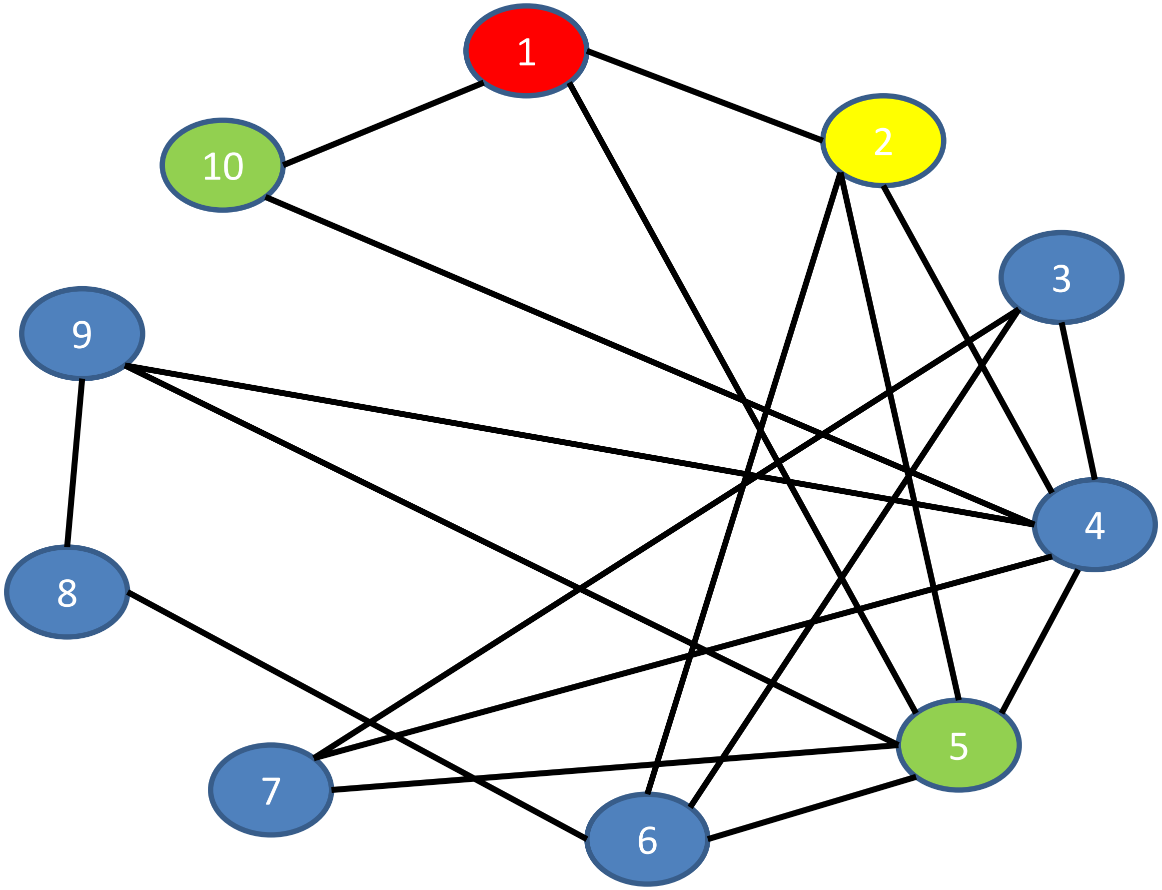
C = {1,2,5}
P = {}
maxSize = 3
incumbent = {1,2,5}

Save as incumbent



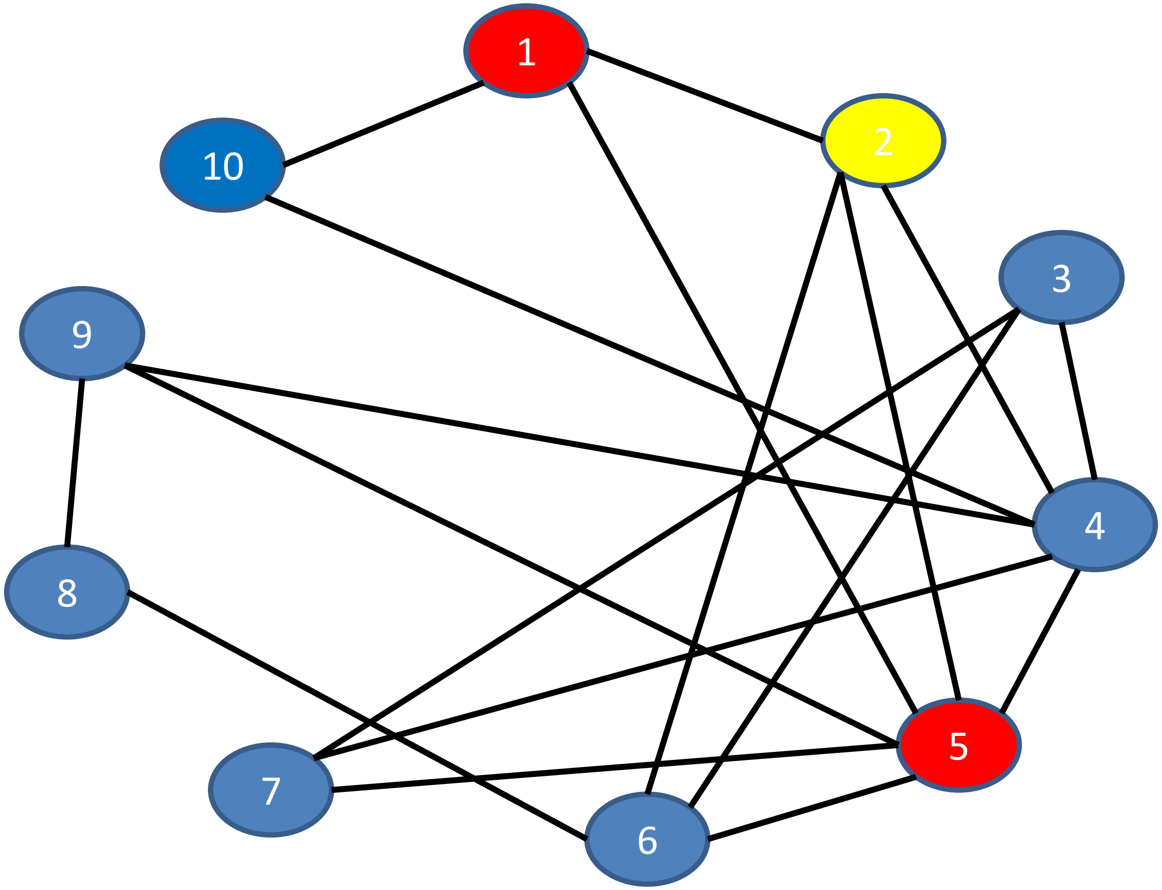
C = {1,2,5}
P = {}
maxSize = 3
incumbent = {1,2,5}

backtrack



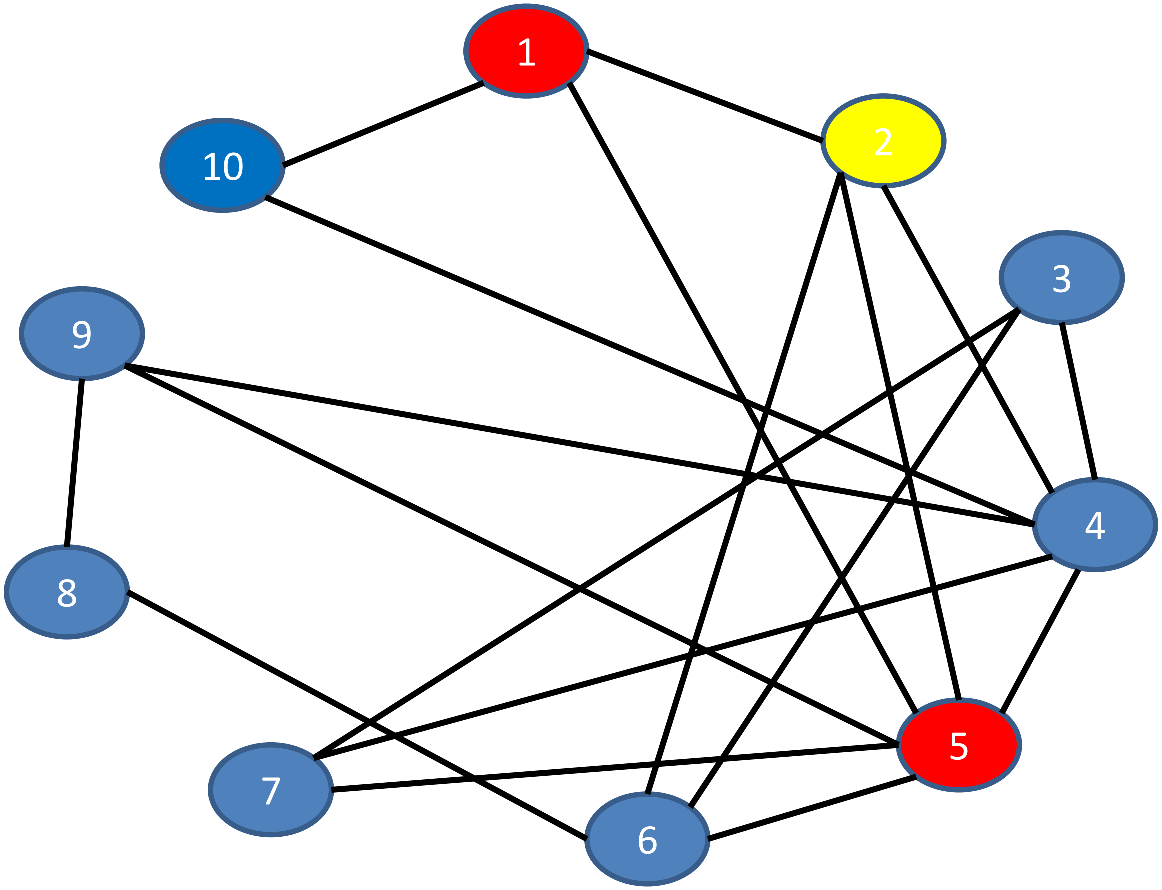
C = {1}
P = {5,10}
maxSize = 3
incumbent = {1,2,5}

Remove 2 from P



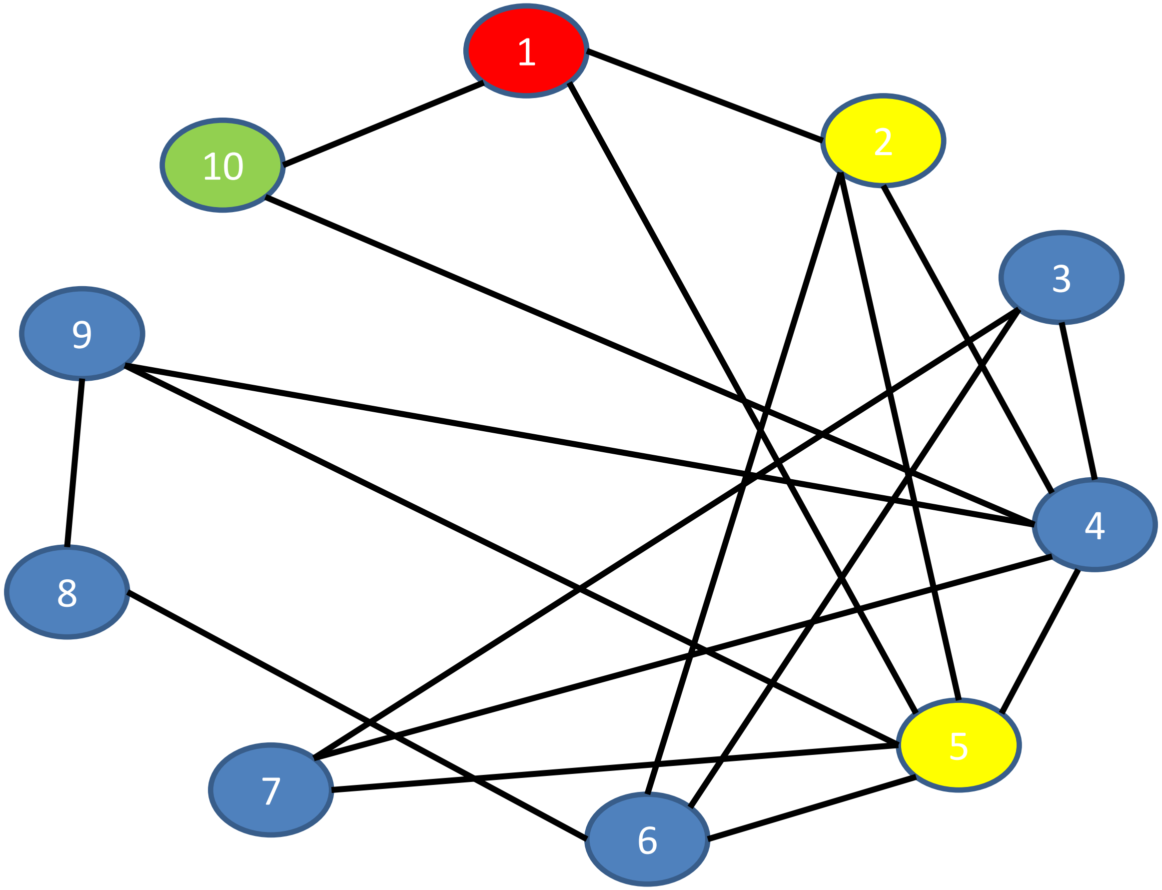
C = {1,5}
P = {}
maxSize = 3
incumbent = {1,2,5}

Select vertex 5



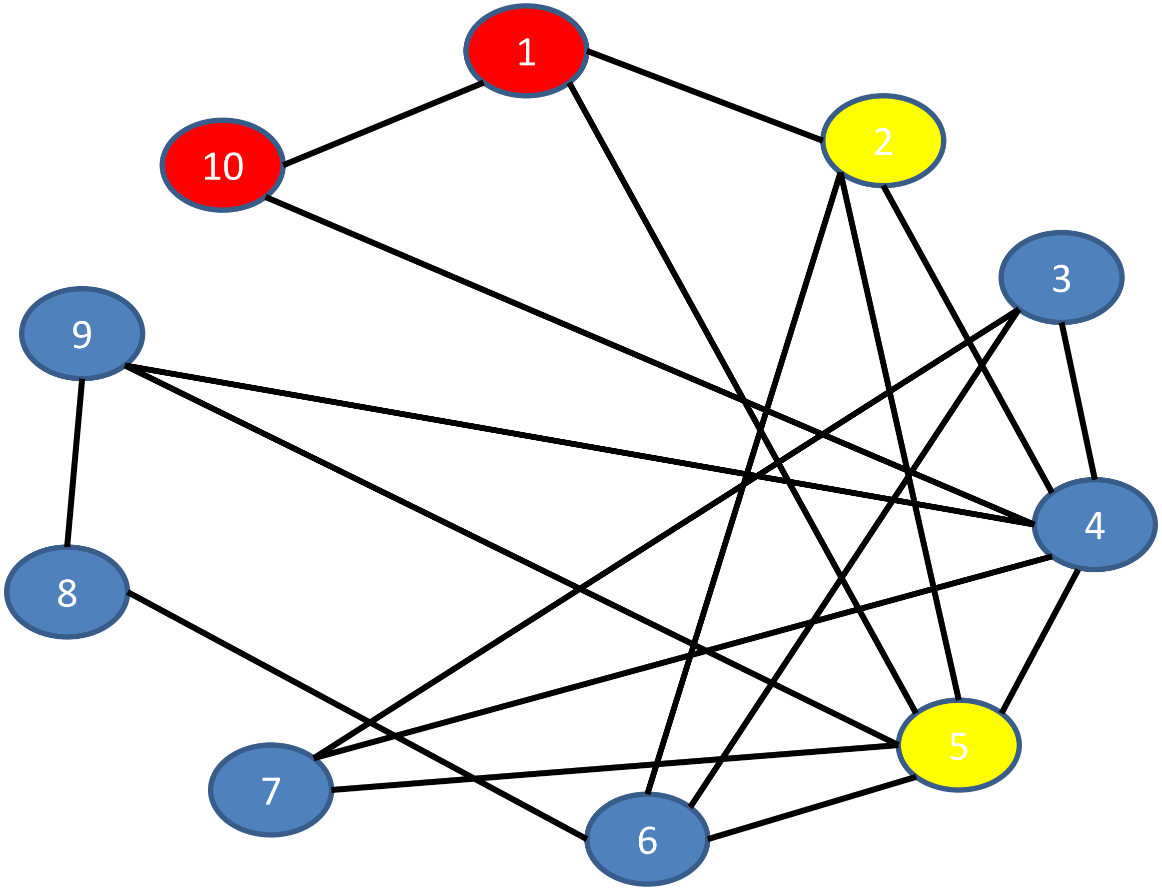
C = {1,5}
P = {}
maxSize = 3
incumbent = {1,2,5}

backtrack



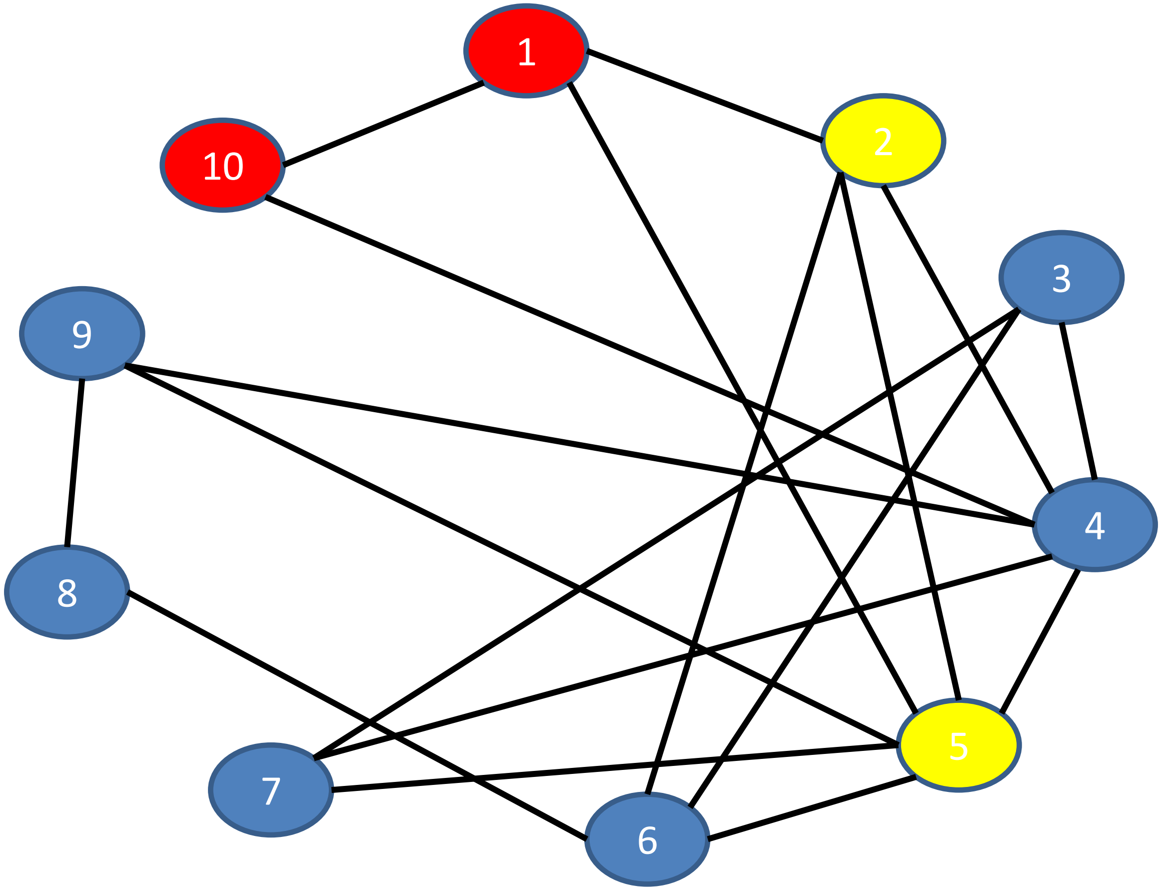
C = {1}
P = {10}
maxSize = 3
incumbent = {1,2,5}

Remove 5 from P



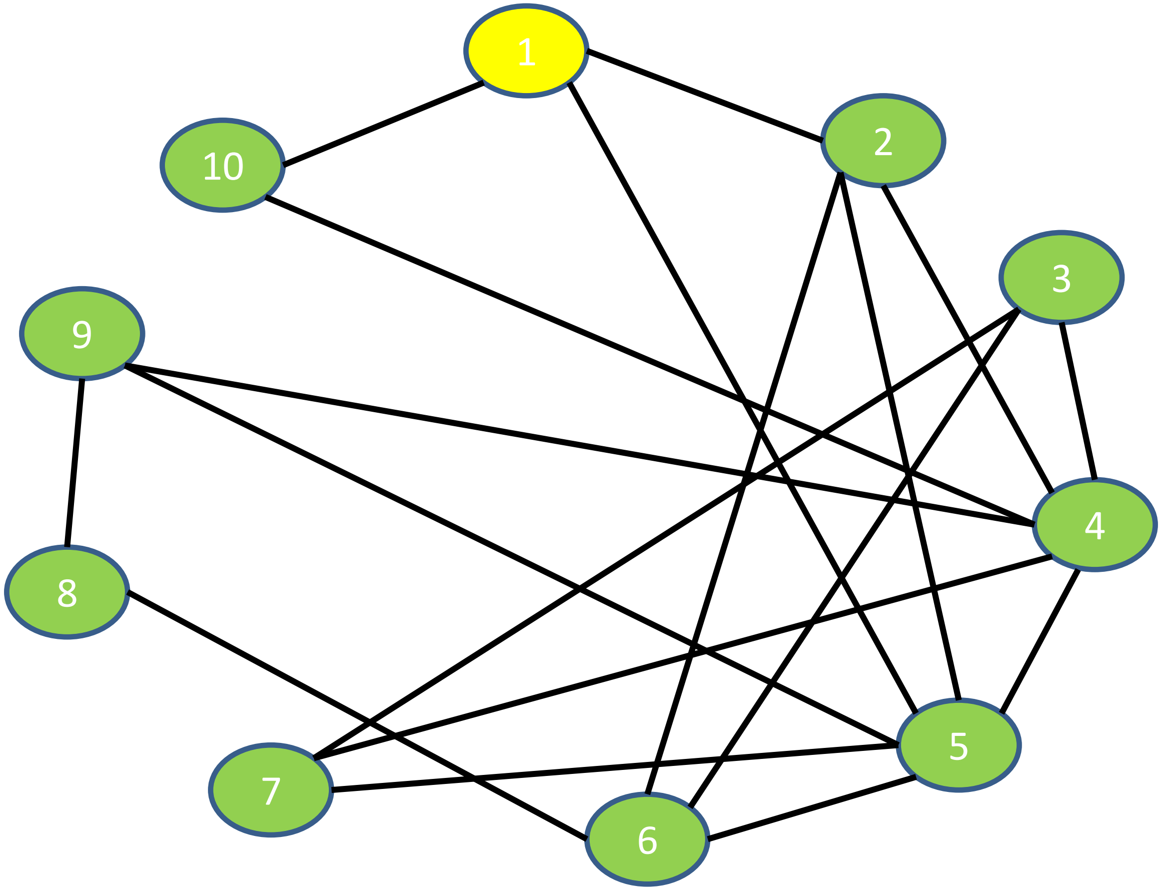
C = {1,10}
P = {}
maxSize = 3
incumbent = {1,2,5}

Select vertex 10



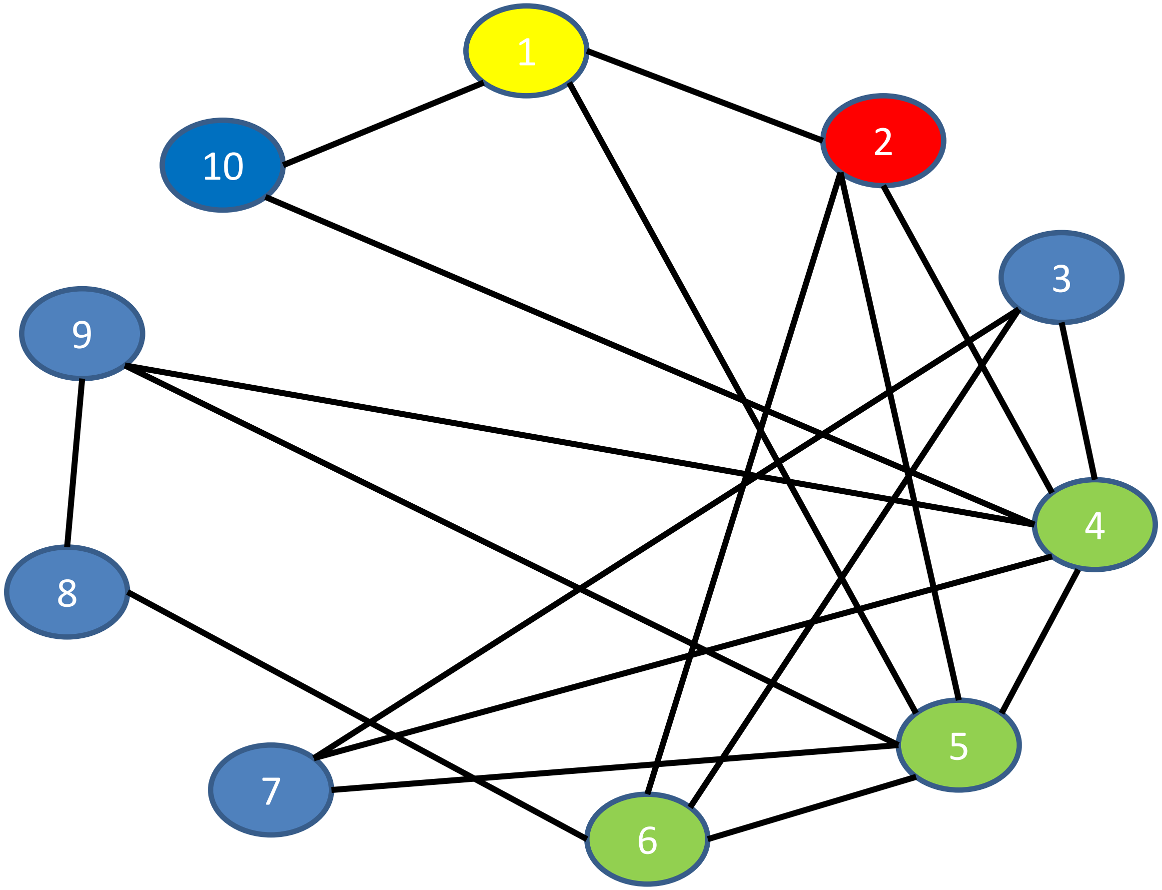
C = {1,10}
P = {}
maxSize = 3
incumbent = {1,2,5}

backtrack



C = {}
P = {2,3,4,5,6,7,8,9,10}
maxSize = 3
incumbent = {1,2,5}

Remove 1 from P



C = {2}
P = {4,5,6}
maxSize = 3
incumbent = {1,2,5}

Select vertex 2

… and so on

MC0 - Notepad

File Edit Format View Help

```
import java.util.*;

public class MC0 extends MC {

    MC0 (int n,int[][]A,int[] degree) {super(n,A,degree);}

    void expand(ArrayList<Integer> C,ArrayList<Integer> P){
        if (timeLimit > 0 && System.currentTimeMillis() - cpuTime >= timeLimit) return;
        nodes++;
        for (int i=P.size()-1;i>=0;i--){
            int v = P.get(i);
            C.add(v);
            ArrayList<Integer> newP = new ArrayList<Integer>(i);
            for (int j=0;j<=i;j++){
                int w = P.get(j);
                if (A[v][w] == 1) newP.add(w);
            }
            if (newP.isEmpty() && C.size() > maxSize) saveSolution(C);
            if (!newP.isEmpty()) expand(C,newP);
            C.remove(C.size()-1);
            P.remove(i);
        }
```

MC0 - Notepad

File Edit Format View Help

```java
import java.util.*;

public class MC0 extends MC {

    MC0 (int n,int[][]A,int[] degree) {super(n,A,degree);}

    void expand(ArrayList<Integer> C,ArrayList<Integer> P){
        if (timeLimit > 0 && System.currentTimeMillis() - cpuTime >= timeLimit) return;
        nodes++;
        for (int i=P.size()-1;i>=0;i--){
            int v = P.get(i);
            C.add(v);
            ArrayList<Integer> newP = new ArrayList<Integer>(i);
            for (int j=0;j<=i;j++){
                int w = P.get(j);
                if (A[v][w] == 1) newP.add(w);
            }
            if (newP.isEmpty() && C.size() > maxSize) saveSolution(C);
            if (!newP.isEmpty()) expand(C,newP);
            C.remove(C.size()-1);
            P.remove(i);
        }
```

MC0 blindly moves forwards whenever newP is not empty

Assume we have an incumbent and that $|P| + |C| \leq maxSize$

Assume we have an incumbent and that $|P| + |C| \leq maxSize$

We can cut off search and backtrack

Assume we have an incumbent and that $|P| + |C| \leq maxSize$

We can cut off search and backtrack

$|P| + |C|$ is a bound and when $|P| + |C| \leq maxSize$ we can abandon search in this branch

MC1 - Notepad

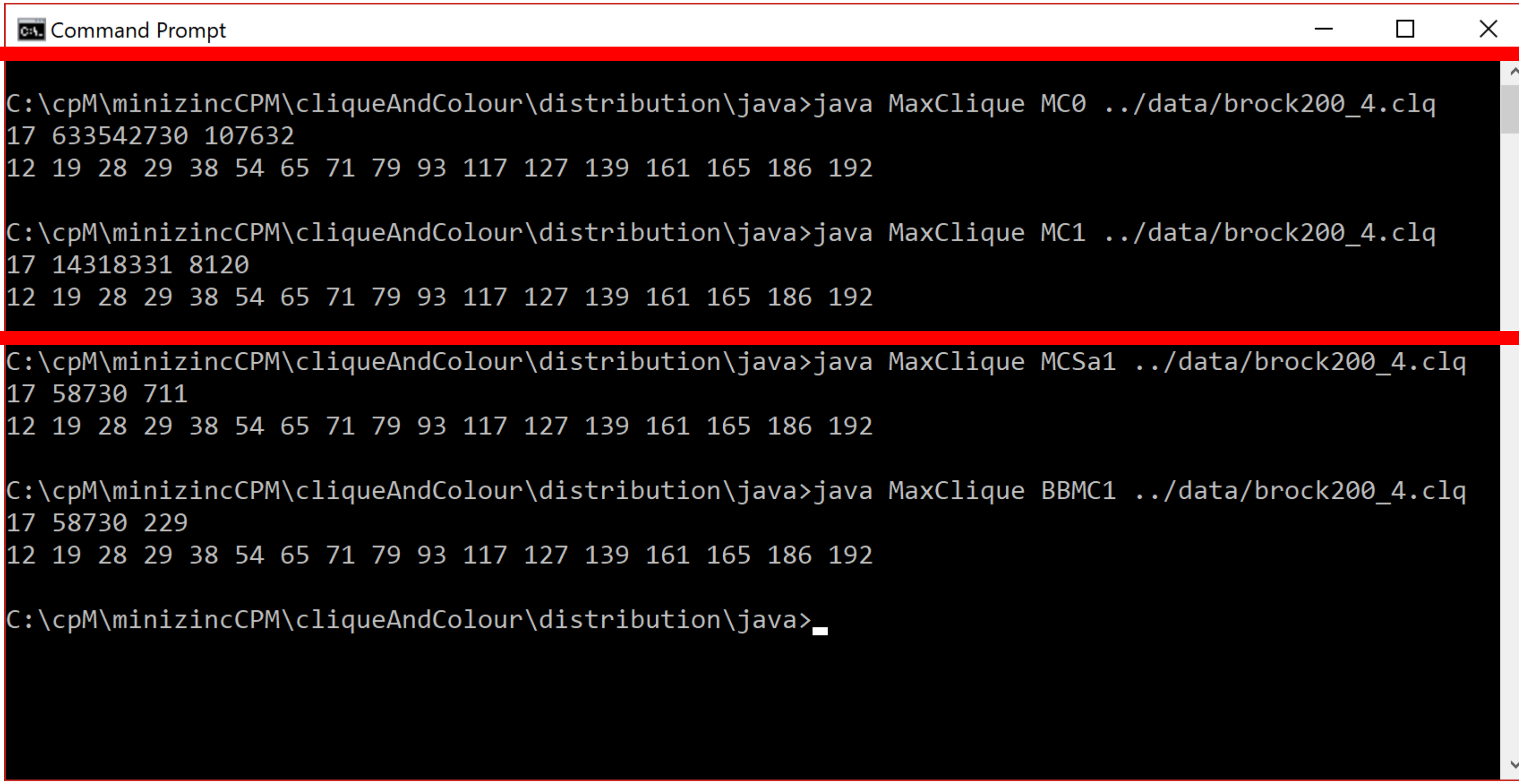
File Edit Format View Help

```
import java.util.*;

public class MC1 extends MC {

    MC1 (int n,int[][]A,int[] degree) {super(n,A,degree);}

    void expand(ArrayList<Integer> C,ArrayList<Integer> P){
        if (timeLimit > 0 && System.currentTimeMillis() - cpuTime >= timeLimit) return;
        nodes++;
        for (int i=P.size()-1;i>=0;i--){
            if (C.size() + P.size() <= maxSize) return;
            int v = P.get(i);
            C.add(v);
            ArrayList<Integer> newP = new ArrayList<Integer>(i);
            for (int j=0;j<=i;j++){
                int w = P.get(j);
                if (A[v][w] == 1) newP.add(w);
            }
            if (newP.isEmpty() && C.size() > maxSize) saveSolution(C);
            if (!newP.isEmpty()) expand(C,newP);
            C.remove(C.size()-1);
            P.remove(i);
        }
    }
```

Small scale statistics

MC0 v MC1

brock200_4



MC0: 107,632 ms
MC1: 8,120 ms
speedup: 13

Our first bound

Assume we have an incumbent and that $|P| + |C| \leq maxSize$

We can cut off search and backtrack

$|P| + |C|$ is a bound and when $|P| + |C| \leq maxSize$ we can abandon search in this branch

Can we think of other bounds that we might compute?

Can we think of other bounds that we might compute?

- Degree of vertices in P
  - All vertices in P are adjacent to all vertices in C
  - The degree between vertices in P may be an measure
    - Is this costly to compute?
    - Does it pay off in runtime (i.e. economical)?
  - If in P we have k vertices with degree ≥ k-1 … ?
  - See Torsten Fahle for menu of filters and bounds

Can we think of other bounds that we might compute?

- The chromatic number of P?
  - The chromatic number is greater than or equal to the clique number
  - Therefore it is a safe bound (***does not excessively prune search***)
  - But that's NP-hard to compute!
  - Is it economical?
  - Can we make good use of colour?

Assume we have a set of vertices P, we can ***greedely*** place these in ***colour classes***

Assume we have a set of vertices P, we can ***greedely*** place these in ***colour classes***

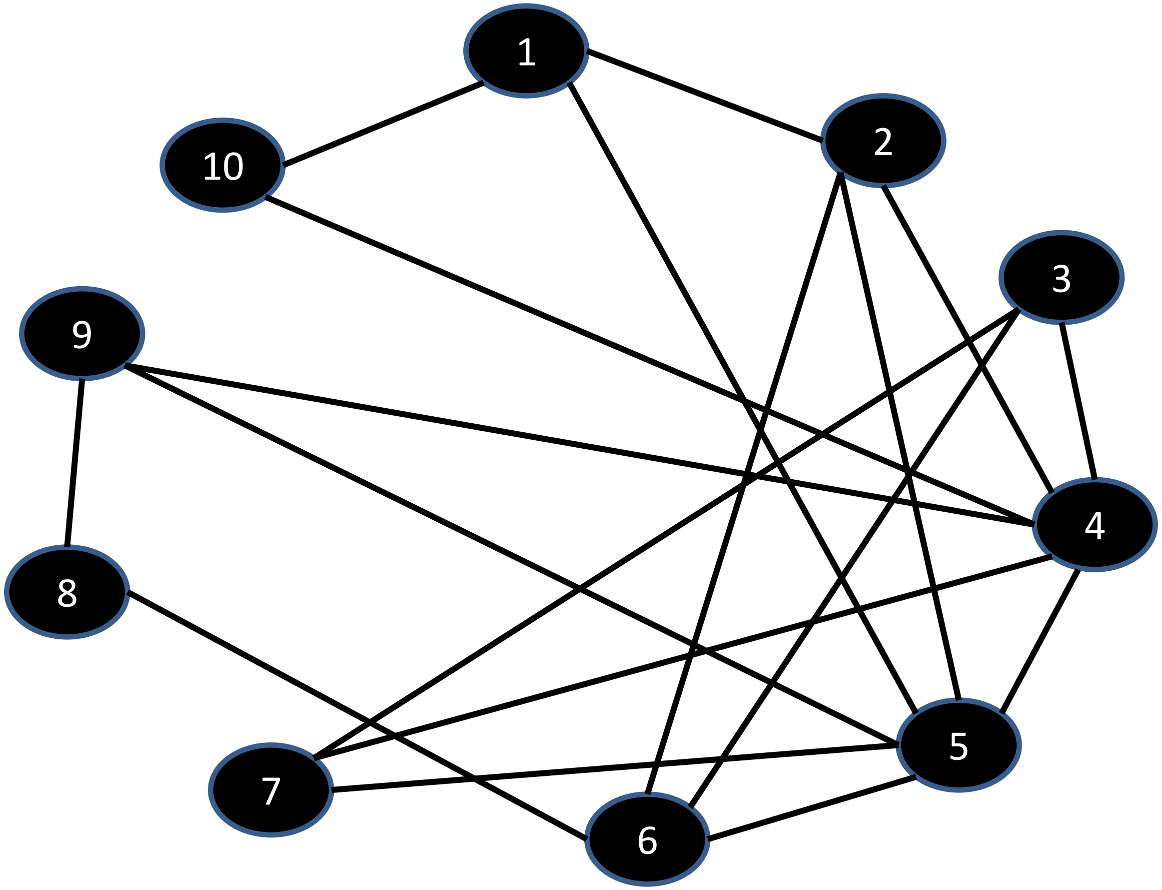
1. k = 1
2. Create an empty colour class C_k
3. Select and remove a vertex v from P and add v to C_k
4. For all vertices w in P
    - 4.1 if w is not adjacent to all vertices in C_k
    - 4.2 then add w to C_k and remove w from P
5. If P is not empty then increment k and go to 2
6. k is an upper bound of the chromatic number of P

Assume we have a set of vertices P, we can ***greedly*** place these in ***colour classes***

1. k = 1
2. Create an empty colour class C_k
3. Select and remove a vertex v from P and add v to C_k
4. For all vertices w in P
    4.1 if w is not adjacent to all vertices in C_k
    4.2 then add w to C_k and remove w from P
5. If P is not empty then increment k and go to 2
6. k is an upper bound of the chromatic number of P

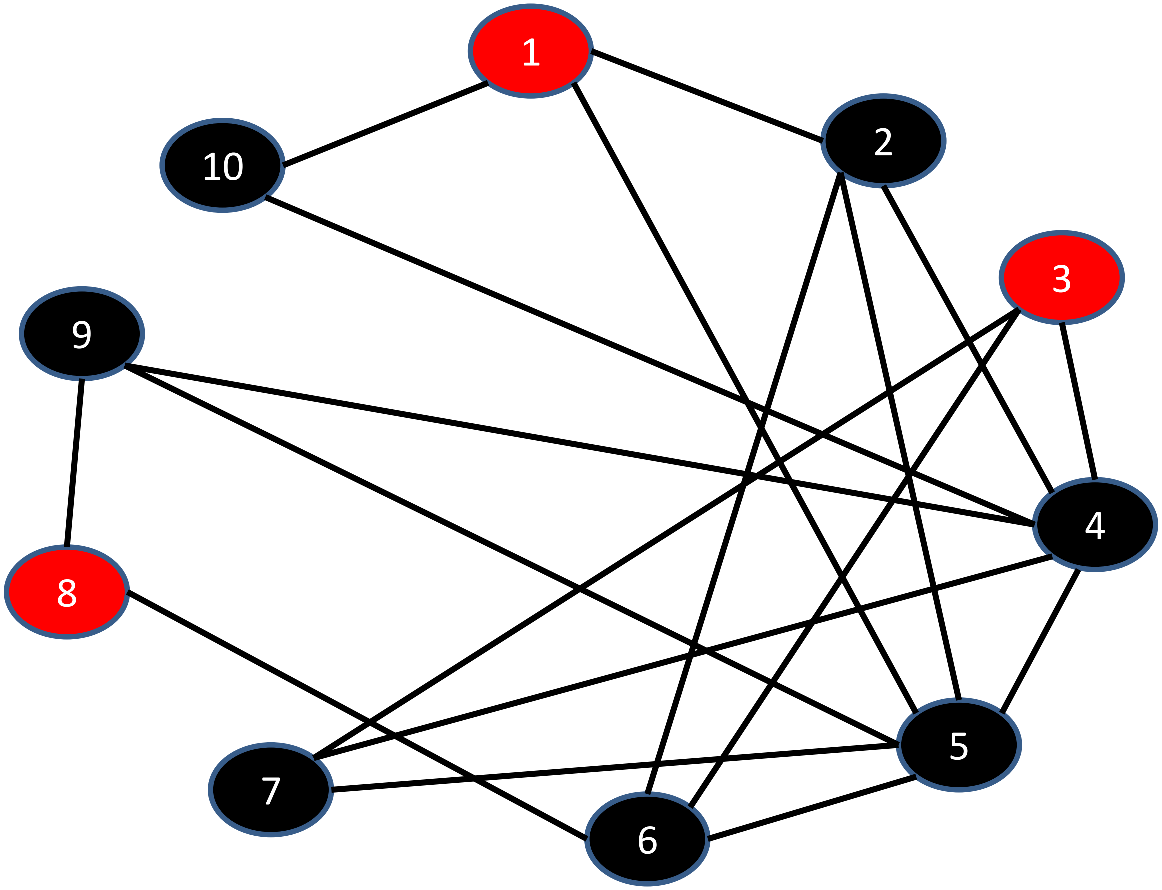
NOTE:
- colour classes are independent sets
- We can select at most one vertex from each colour class

Greedy colouring



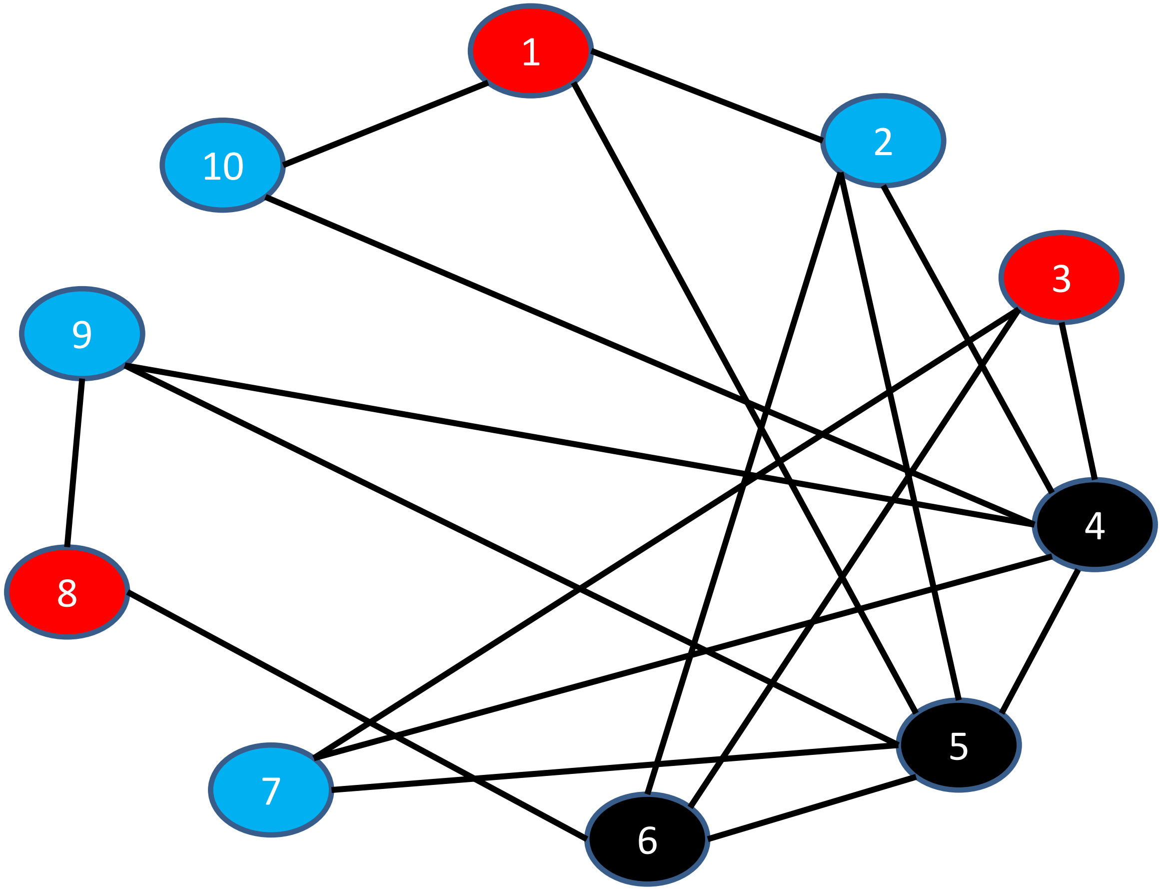
P = {1,2,3,4,5,6,7,8,9,10}

Greedy colouring



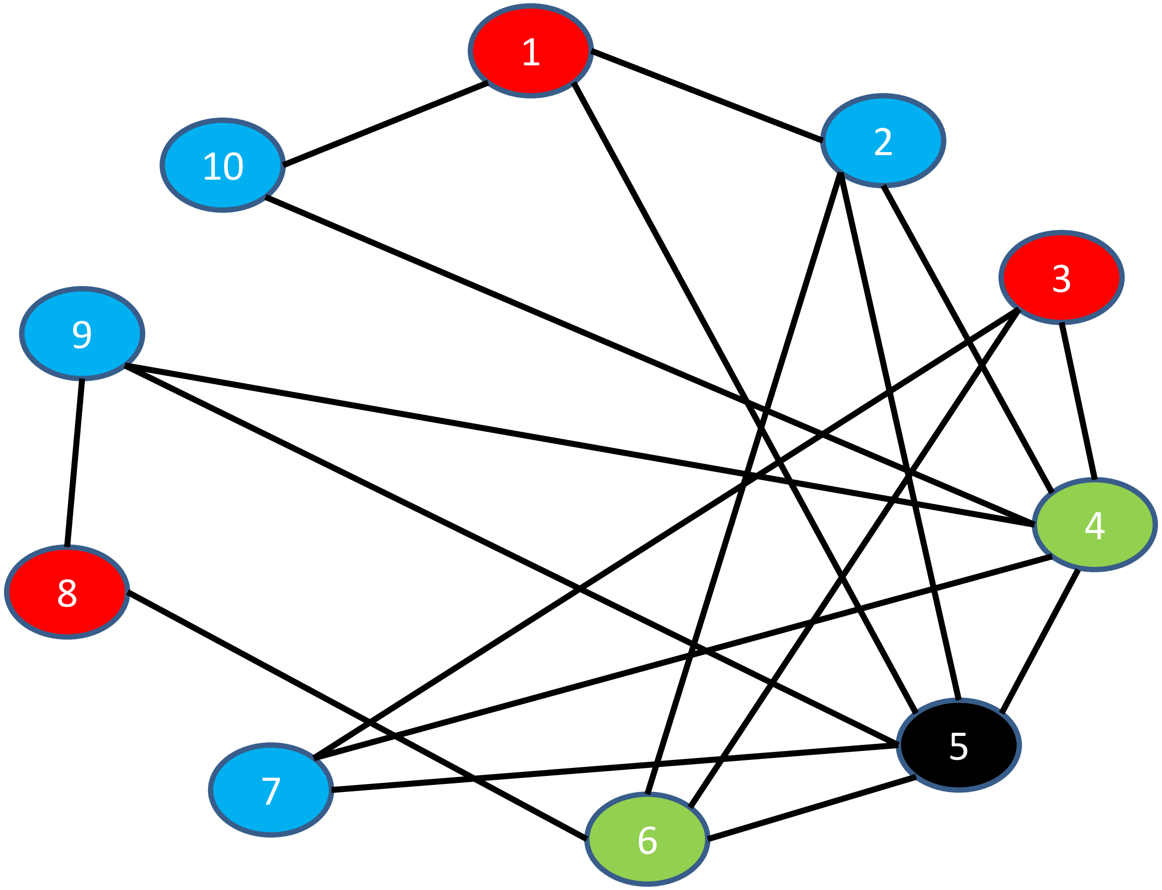
P = {1,2,3,4,5,6,7,8,9,10}
C_1 = {1,3,8}

Greedy colouring



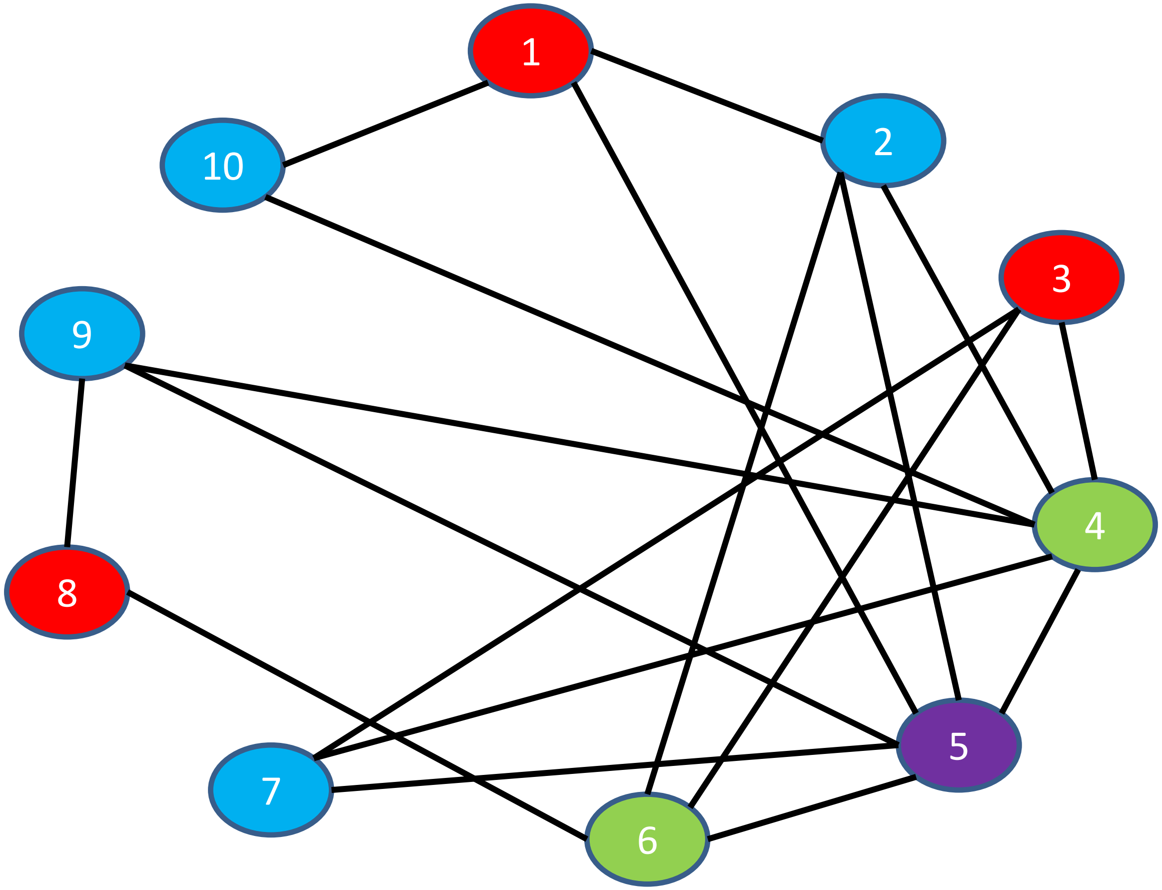
P = {2,4,5,6,7,9,10}
C_1 = {1,3,8}
C_2 = {2,7,9,10}

Greedy colouring



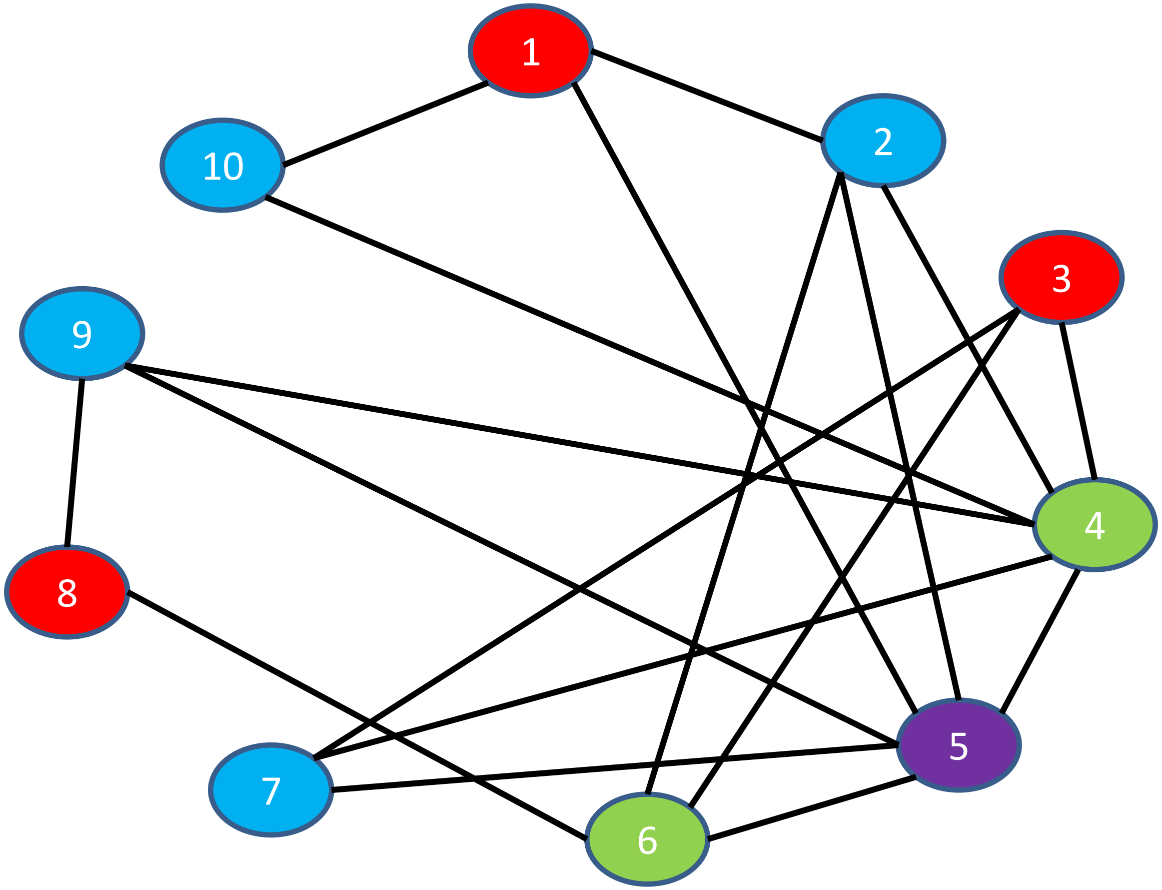
P = {4,5,6}
C_1 = {1,3,8}
C_2 = {2,7,9,10}
C_3 = {4,6}

Greedy colouring



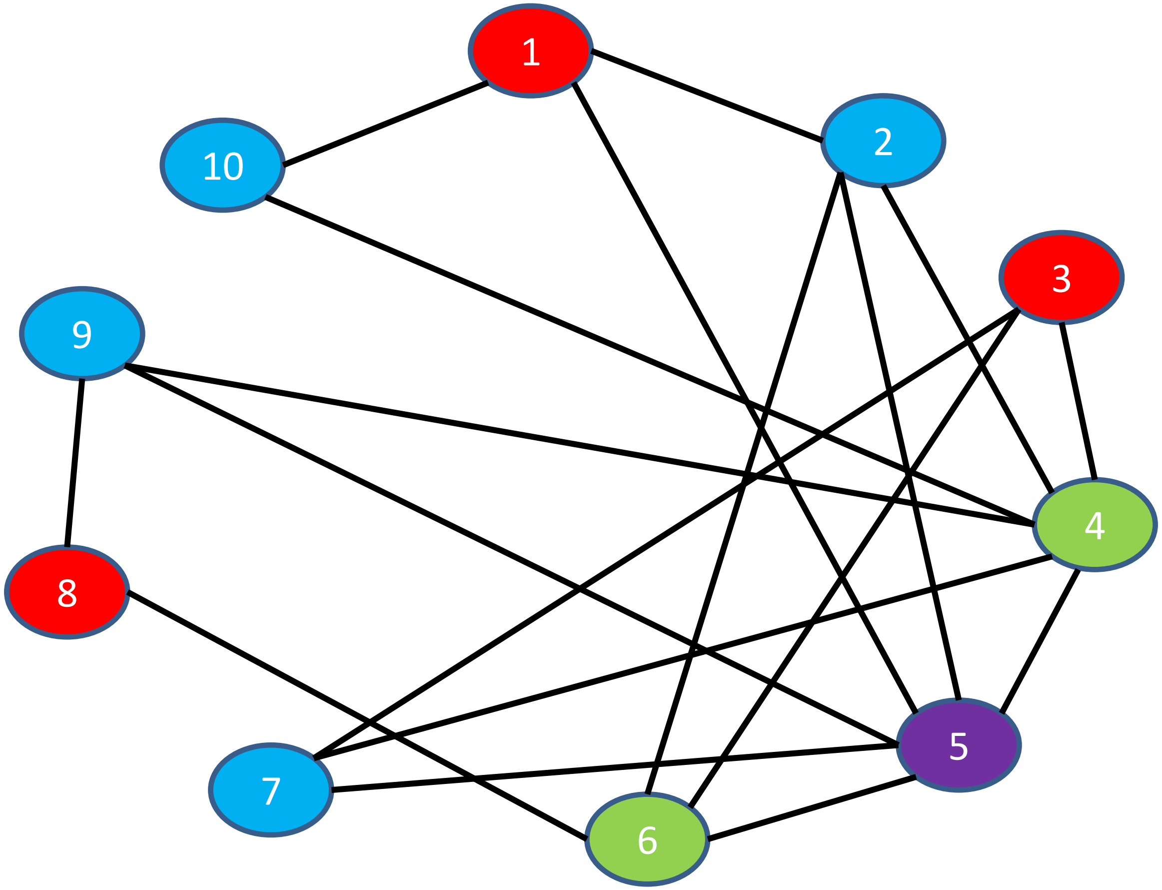
P = {5}
C_1 = {1,3,8}
C_2 = {2,7,9,10}
C_3 = {4,6}
C_4 = {5}

Greedy colouring



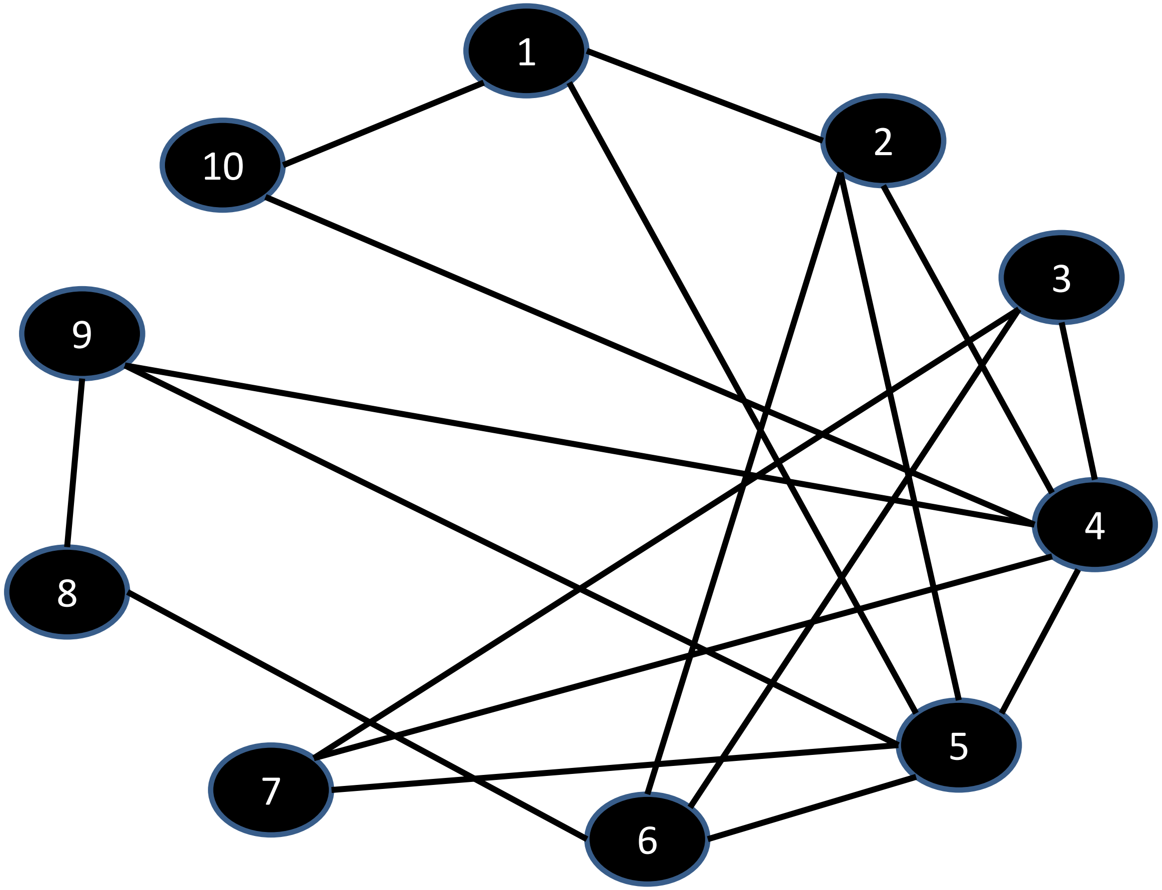
- We get a greedy estimate (k) of chromatic number of candidate set P
- We can do this in every call to expand …
- If |C| + k ≤ maxSize we can abandon current branch

Greedy colouring



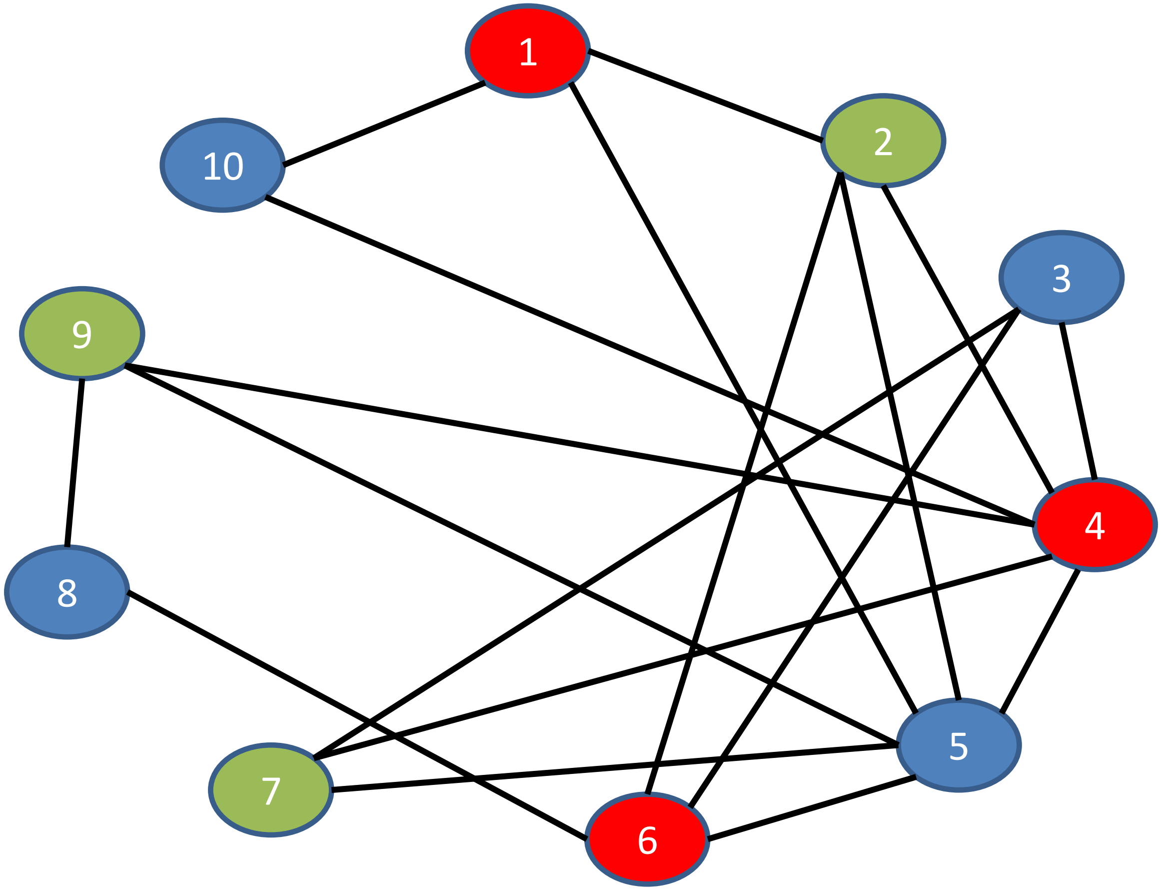
- We get a greedy estimate (k) of chromatic number of candidate set P
- We can do this in every call to expand …
- If |C| + k ≤ maxSize we can abandon current branch
- ***And we can get smart!***

Order vertices for colouring

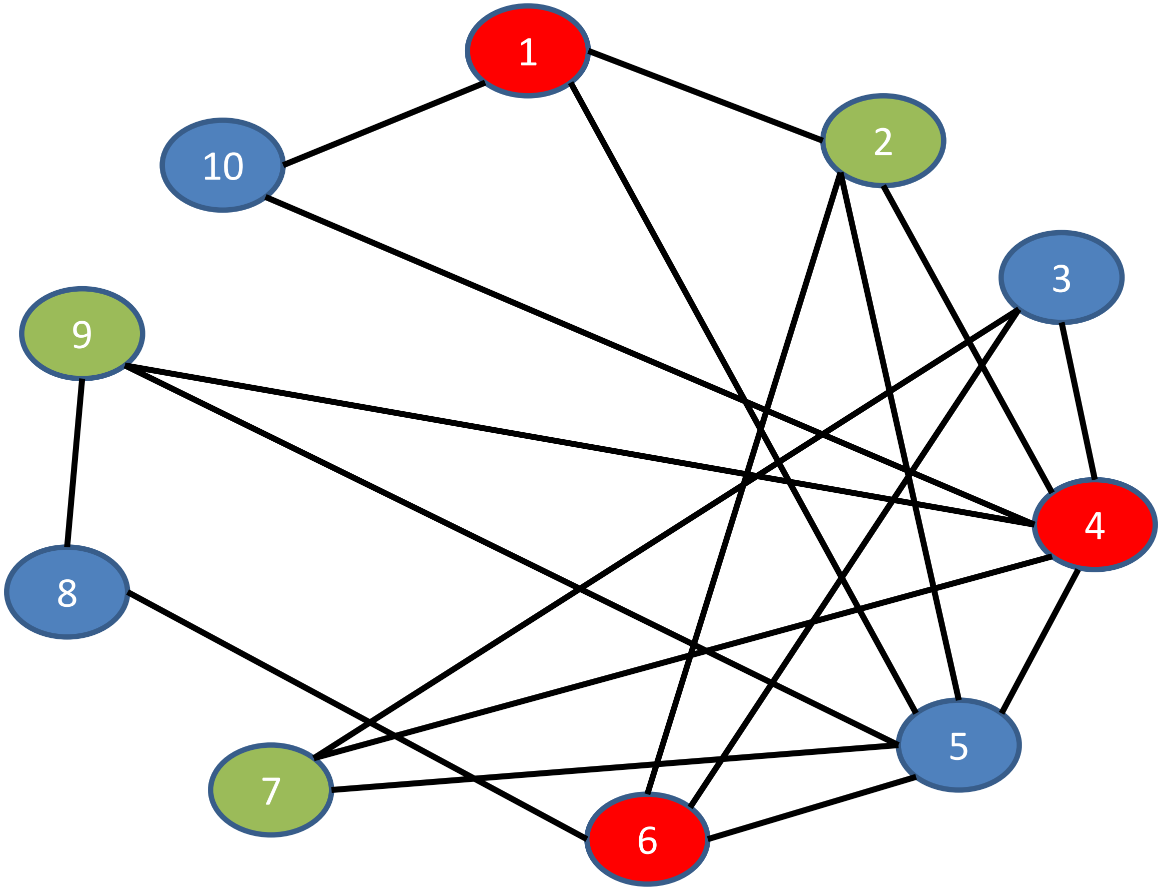


Non-increasing degree order {4,5,6,2,1,3,7,9,8,10}

Order vertices for colouring



Non-increasing degree order {4,5,6,2,1,3,7,9,8,10}
C_1 = {1,4,6}
C_2 = {3,5,8,10}
C_3 = {2,7,9}
k = 3
A tighter bound!

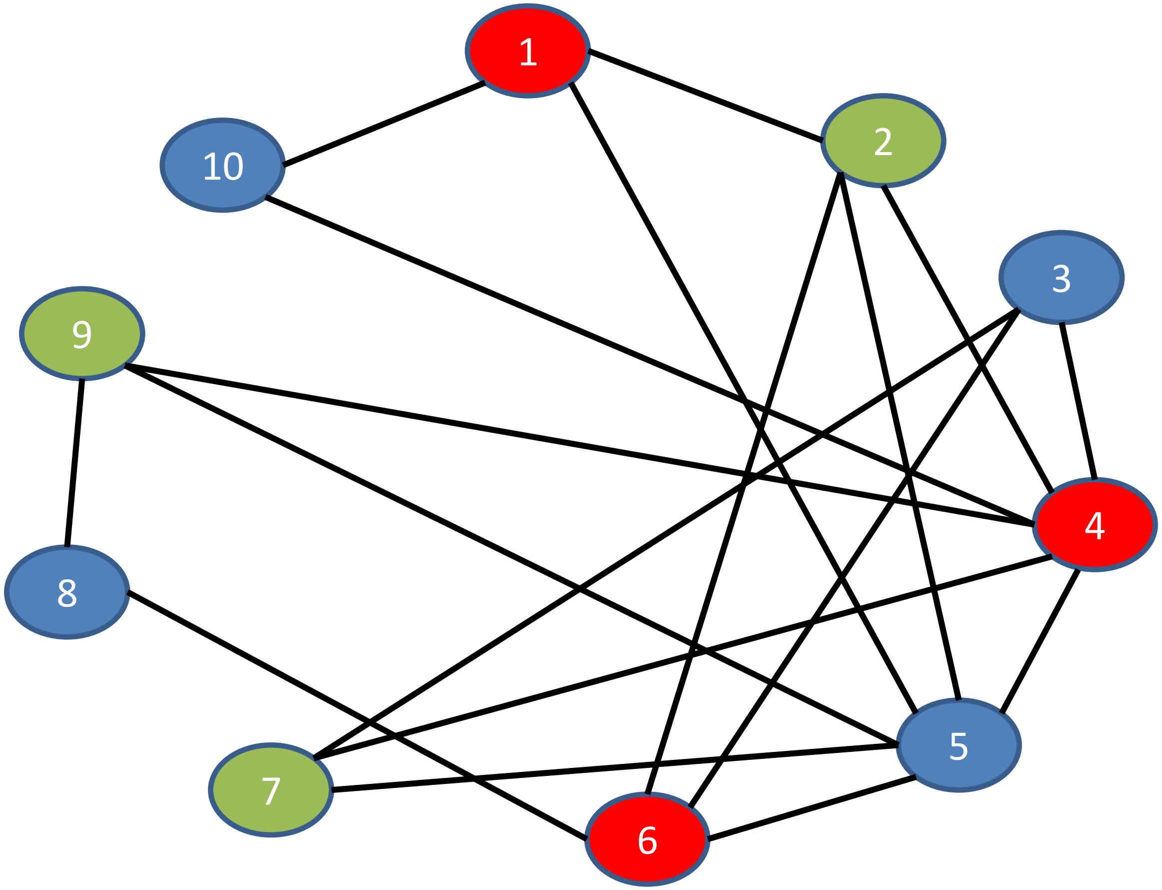


Order vertices for colouring

Non-increasing degree order {4,5,6,2,1,3,7,9,8,10}
C_1 = {1,4,6}
C_2 = {3,5,8,10}
C_3 = {2,7,9}
k = 3
A tighter bound!

There are other orderings
- a Brelaz ordering
- Minimum width ordering
- …

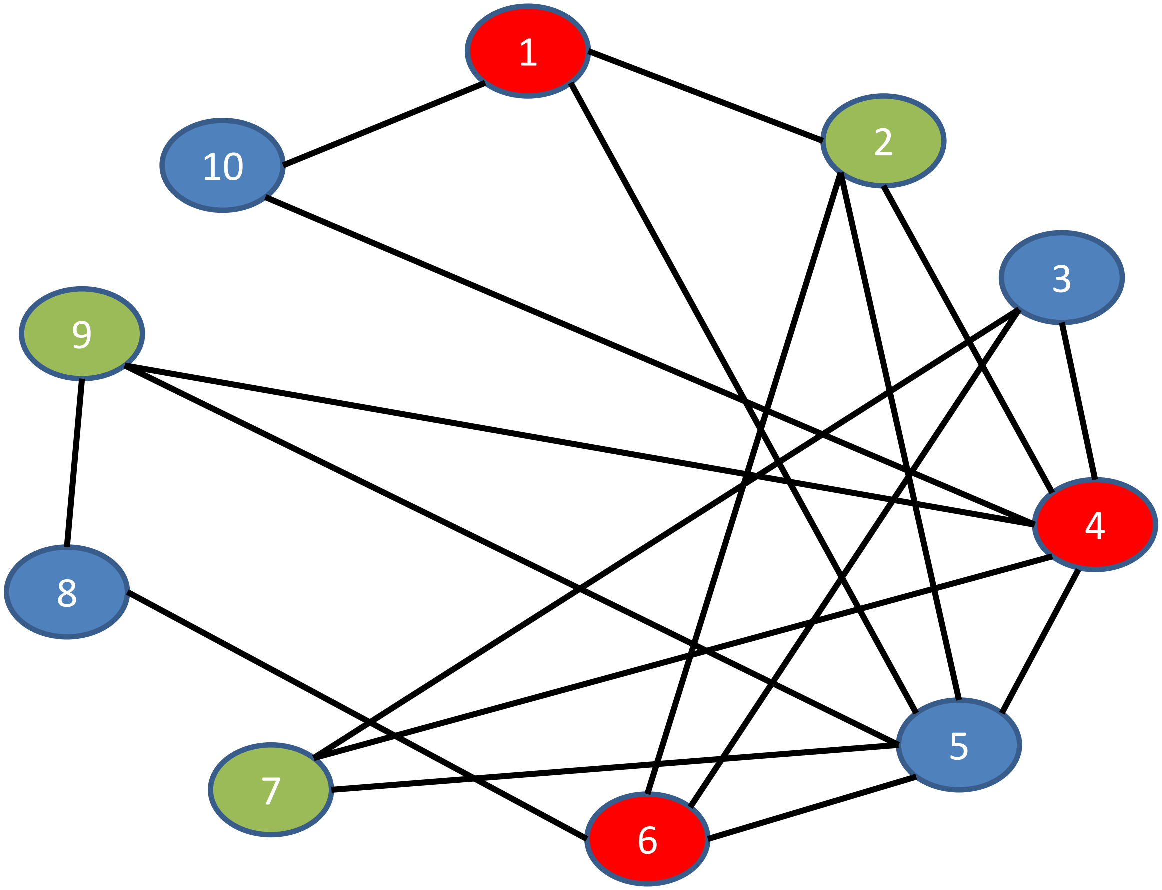
Can we do something with the colour classes, not just the value k?



Visiting colour classes

In expand, iterate over vertices in colour classes from colour class C_k down to C_1 i.e. start with {2,7,9} then {3,5,8,10} finally {1,4,6}.

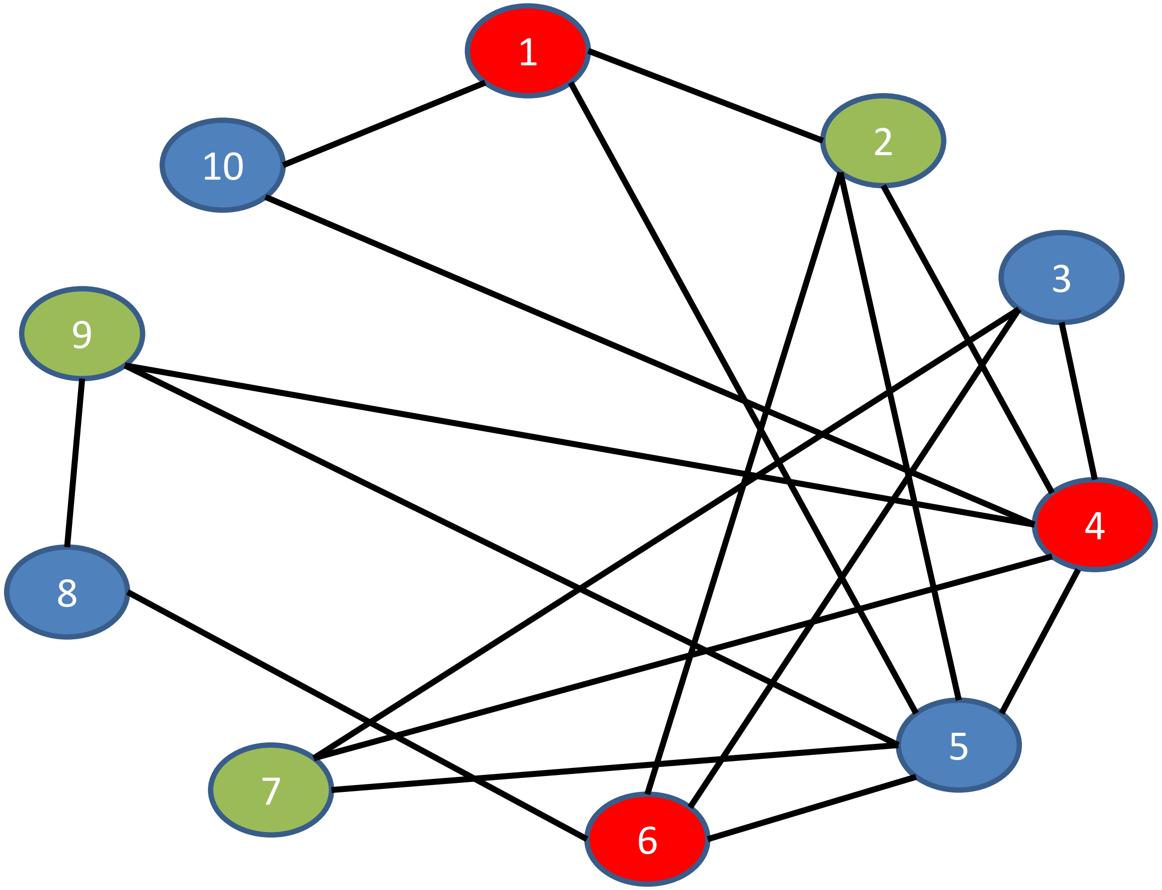
Visiting colour classes



In expand, iterate over vertices in colour classes from colour class C_k down to C_1 i.e. start with {2,7,9} then {3,5,8,10} finally {1,4,6}.

If |C| + index of current colour class ≤ maxSize then backtrack …

Visiting colour classes



In expand, iterate over vertices in colour classes from colour class C_k down to C_1
i.e. start with {2,7,9} then {3,5,8,10} finally {1,4,6}.

If |C| + index of current colour class ≤ maxSize then backtrack …

… because we can only pick one vertex from each of the k colour classes (independent sets)

Small scale statistics

MC1 v MCSa1

brock200_4

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC0 ../data/brock200_4.clq
17 633542730 107632
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC1 ../data/brock200_4.clq
17 14318331 8120
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MCSa1 ../data/brock200_4.clq
17 58730 711
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

17 58730 229
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>
```

MC1: 8,120 ms
MCSa1: 711 ms
speedup: 11

Our second, smartly used, bound

To a man with a hammer everything looks like a nail

- We can only select one vertex from a colour class
- Can we consider a colour class as a variable with a domain?
- If so, what colour class should we pick first?

Having selected v from the candidate set P we create the new candidate set P’ as follows:

1. P’ = {}
2. forall w in P
   2.1 w is adjacent to v then add w to P’

We might have P and P’ as ***BitSet*** and each row of the adjacency matrix a **BitSet** such that having selected vertex v we can compute P’ as follows

P’ = P and A[v]

… that is, give me all vertices in P that are adjacent to vertex v.

This exploits in-processor parallelism, and the resultant algorithm is BBMC

Small scale statistics

MCSa1 v BBMC1

brock200_4

Command Prompt

```
C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC0 ../data/brock200_4.clq
17 633542730 107632
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MC1 ../data/brock200_4.clq
17 14318331 8120
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique MCSa1 ../data/brock200_4.clq
17 58730 711
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>java MaxClique BBMC1 ../data/brock200_4.clq
17 58730 229
12 19 28 29 38 54 65 71 79 93 117 127 139 161 165 186 192

C:\cpM\minizincCPM\cliqueAndColour\distribution\java>_
```

MCSa1: 711 ms
BBMC1: 229 ms
speedup: 3

Good engineering

Rewrite everything in C++

Algorithmic Engineering

Rewrite everything in C++

Go parallel

Want to know more?

Look at Patrick Prosser and Ciaran McCreesh on dblp

TSP

In TSP, as the current tour is being extended we have a set of cities yet to be visited

- Assume we have an incumbent and the cost of that tour, minCost
- Assume we have cost of current partial tour
- Assume we have set N of cities not yet visited
- Compute a MST for N
- If cost of partial tour plus cost of MST is ≥ minCost then backtrack

Generally

In most all optimisation problems where we use exact search there is considerable effort to come up with sharp and economical bounds.

Generally

In most all optimisation problems where we use exact search there is considerable effort to come up with sharp and economical bounds.

I have only presented Max Clique because that's what I've recently been working on with Ciaran

Generally

In most all optimisation problems where we use exact search there is considerable effort to come up with sharp and economical bounds.

I have only presented Max Clique because that’s what I’ve recently been working on with Ciaran …

… and maybe because this is all that I know