ELSEVIER

Discrete Applied Mathematics

Volume 74, Issue 3, 2 May 1997, Pages 217-228

Material copied from:

Contribution

Graph domination, tabu search and the football pool problem

Rowan Davies [a], Gordon F. Royle [b]

⊞ **Show more**

doi:10.1016/S0166-218X(96)00049-2

Abstract

We describe the use of a tabu search algorithm for generating near minimum dominating sets in graphs. We demonstrate the effectiveness of this algorithm by considering a previously studied class of graphs, the so-called "football pool" graphs, and improving many of the known upper bounds for this class.

Note: most journal papers are accessible electronically for free through the UVic library.

Introduction to the tactic:

Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research. Vol. 13, pp. 533-549.

Hansen, P. 1986. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Neighbourhood search algorithms initially impose a neighbourhood structure on the set of configurations X. Starting at a (possibly randomly) chosen element x_0 the algorithm proceeds by repeatedly moving from a configuration to one of its neighbours, with the ultimate aim of finding a configuration of low cost.

Hill climbing and simulated annealing are two popular neighbourhood search techniques that have had some success in combinatorial problems. Tabu search is a more recent technique.

Tabu – the Polynesian concept of something prohibited from being mentioned or touched.

Tabu search algorithm: a heuristic approach that avoids cycling back to local optima

Tabu Algorithm: Start at any x_0 .
At step i choose x_i as a neighbour of x_{i-1} that minimises $c(x^i)$ subject to the constraint that the move from x_{i-1} to x_i does not `undo' any of the t most recent moves. Finish after a number of iterations, returning the x_i which gave the least $c(x_i)$.

The tabu list prevents an immediate return to a local minimum, and with luck the search is forced out of the region of attraction of that local minimum.

Maintains a current solution S which is any subset of V(G) (either a dominating set or a partial dominating set.

Each set S has a cost c(S)= |S| + number of vertices not dominated by S.

Note that S does not have to be a dominating set but S together with the undominated vertices is a dominating set. The reason for not constraining S to be a dominating set is that this allows paths between small dominating sets which might otherwise need to go via a much larger dominating set.

For a solution S, the neighbouring solutions are obtained by either deleting a vertex in S or adding a vertex not in S.

Two moves are defined to undo each other if one is addition and the other deletion of the same vertex.

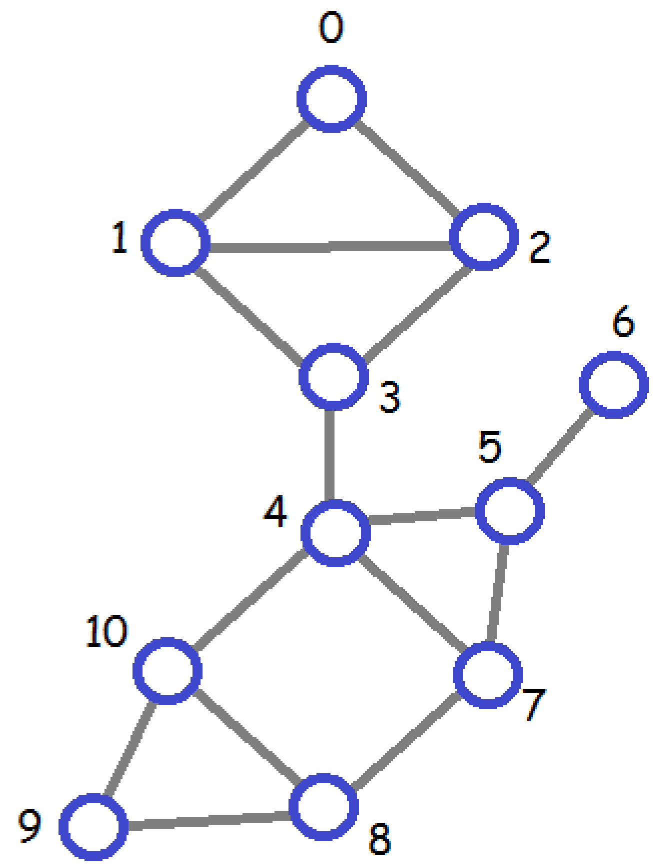
If there are several equally good optimal moves a random choice is made.

For the Tabu list, Rowan and Gordon used Tabu list lengths of between 5 and 8 for most of the runs. The example I give has Tabu list size 3.

A very simple aspiration criterion was also used: if an otherwise tabu move gave an improvement on the best solution found to date, then it was performed anyway.

Trial 0: S is the empty set

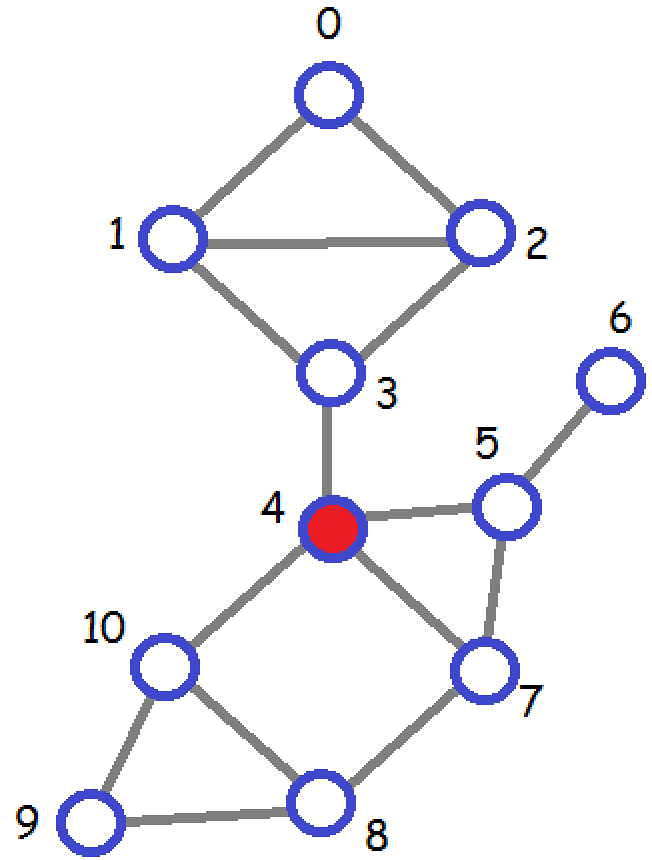
0:	9
1:	8
2:	8
3:	8
4:	7
5:	8
6:	10
7:	8
8:	8
9:	9
10:	8



Minimum cost is 7 for vertex 4

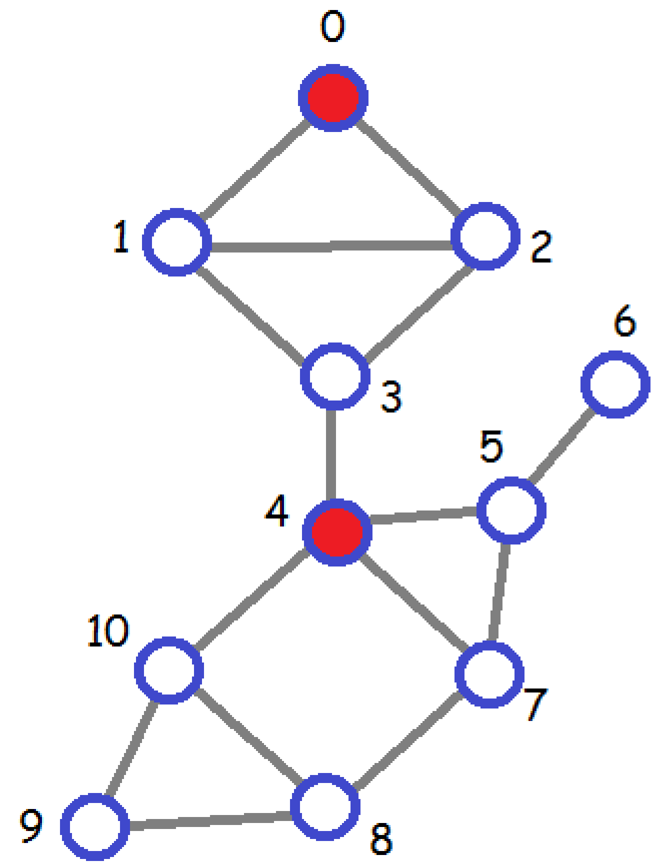
```
Trial 1: S=  4
Tabu:  4 -1 -1

Costs of the vertices:
  0:   5
  1:   5
  2:   5
  3:   6
  5:   7
  6:   7
  7:   7
  8:   6
  9:   6
 10:   6
```

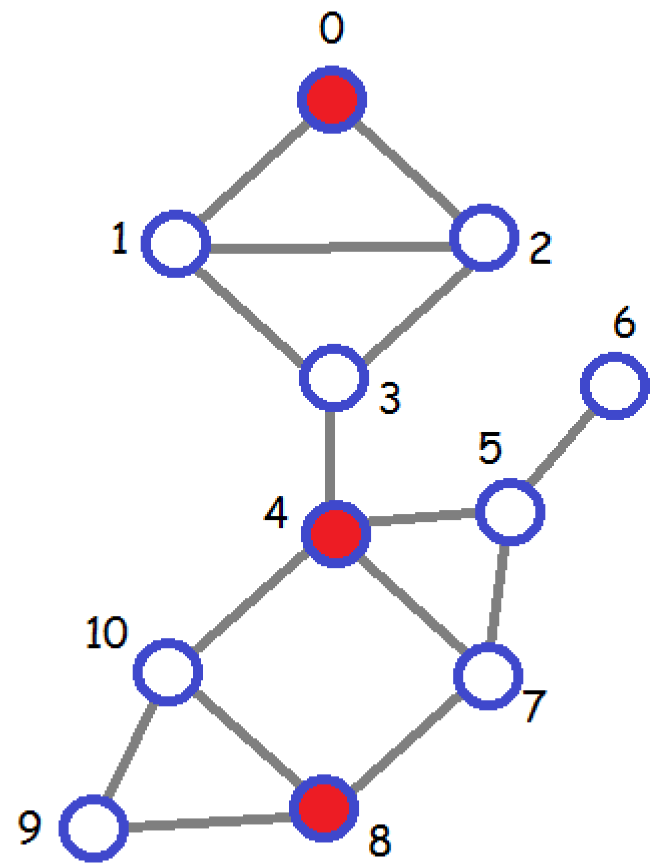


Minimum cost is 5
for vertex 0

```
Trial   2: S=  0  4
Tabu:  4  0 -1
Costs of the vertices:
  1:   6
  2:   6
  3:   6
  5:   5
  6:   5
  7:   5
  8:   4
  9:   4
 10:   4
Minimum cost is   4 for vertex   8
```



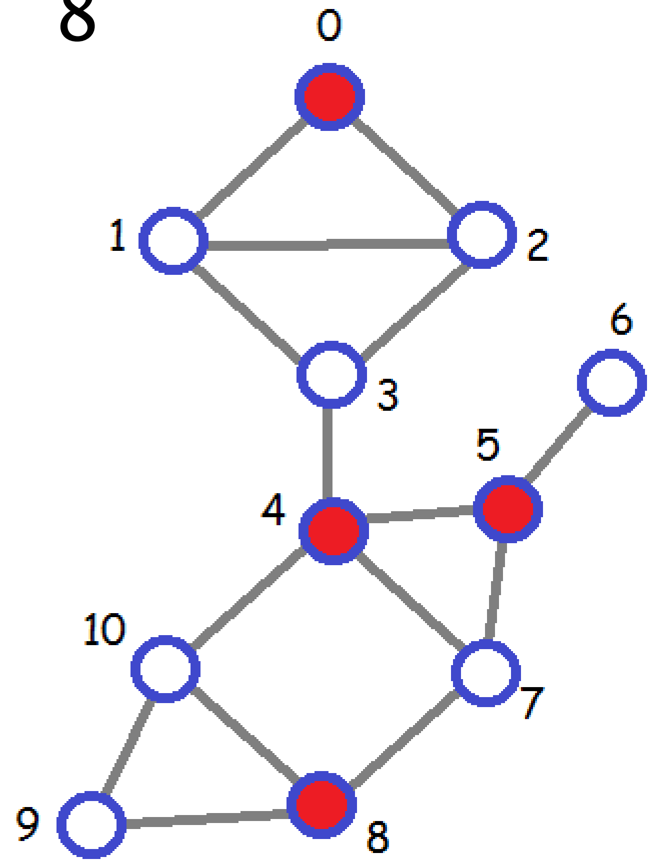
```
Trial    3: S=   0   4   8
Tabu:   4   0   8
Costs of the vertices:
   1:    5
   2:    5
   3:    5
   5:    4
   6:    4
   7:    5
   9:    5
  10:    5
```



Minimum cost is 4
for vertex 5

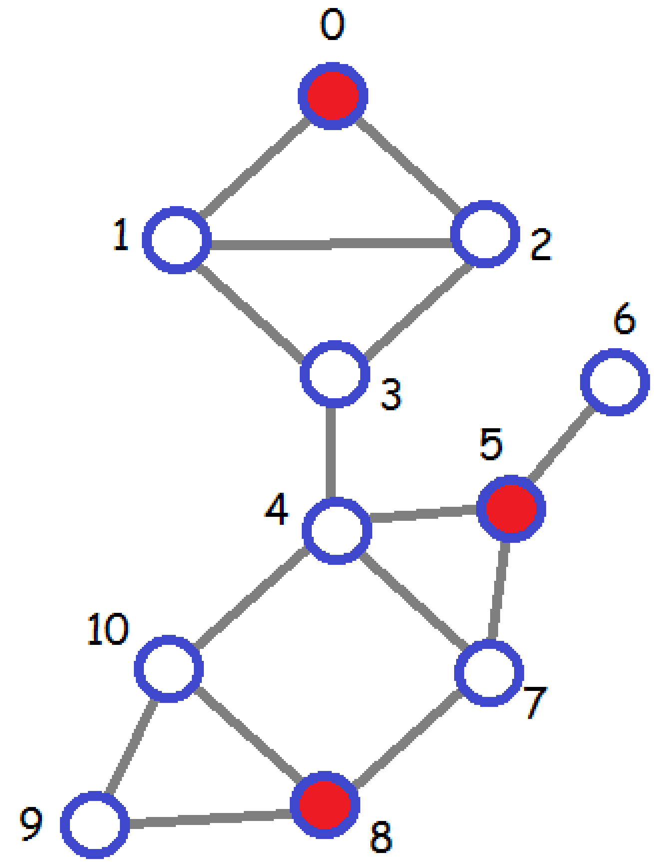
Smaller dominating set: 0 4 5 8

```
Trial   4: S=  0  4  5  8
Tabu:  5  0  8
Costs of the vertices:
  1:   5
  2:   5
  3:   5
  4:   4
  6:   5
  7:   5
  9:   5
 10:   5
```

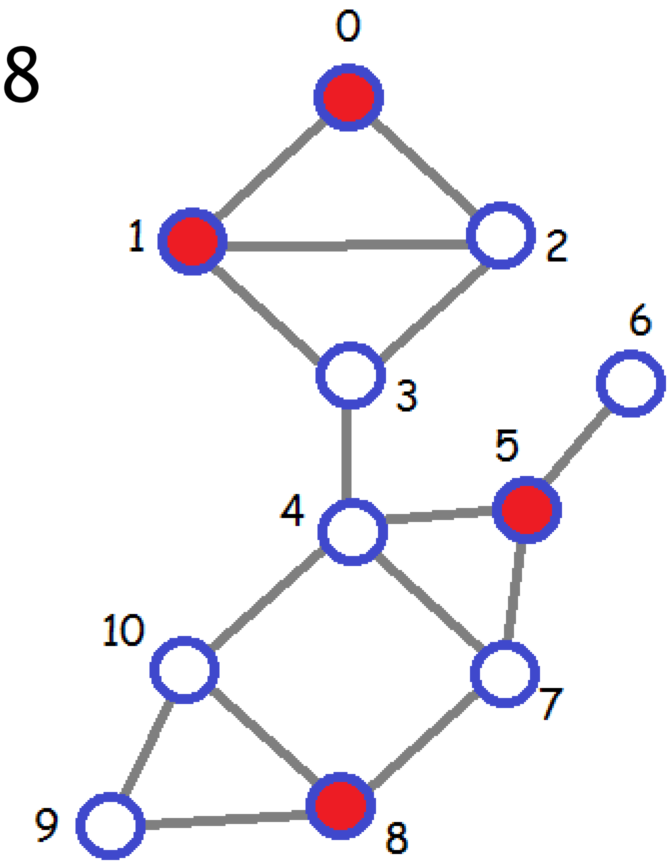



```
Minimum cost is   4 for vertex   4
```

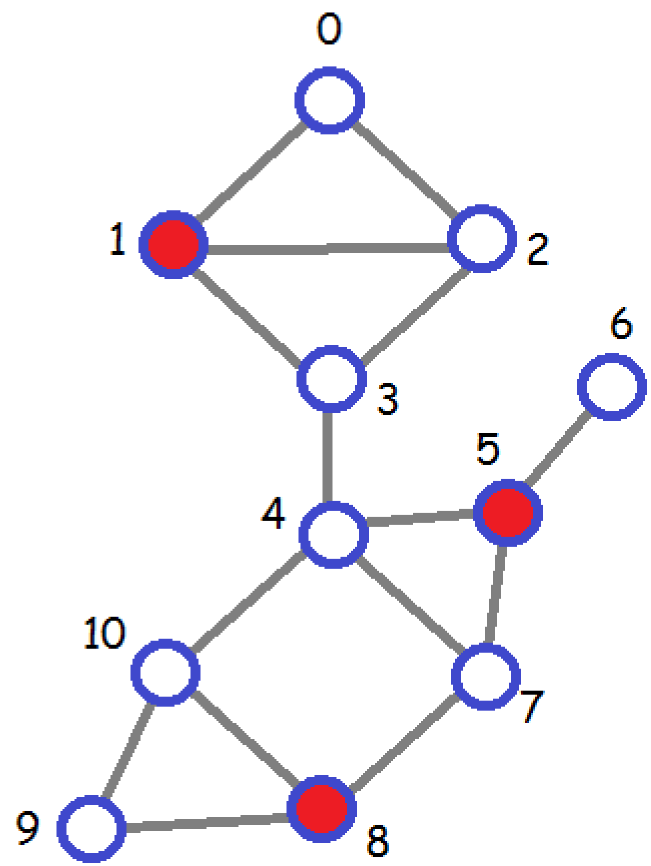
```
Trial    5: S=  0  5  8
Tabu:  5  4  8
Costs of the vertices:
  0:   6
  1:   4
  2:   4
  3:   4
  6:   5
  7:   5
  9:   5
 10:   5
Minimum cost is   4 for vertex   1
```



```
Trial  6: S=  0  1  5  8
Tabu:  5  4  1
Costs of the vertices:
  0:   3
  2:   5
  3:   5
  6:   5
  7:   5
  8:   6
  9:   5
 10:   5
Minimum cost is   3 for vertex   0
Smaller dominating set:  1  5  8
```



```
Trial    7: S=  1  5  8
Tabu:  0  4  1
Costs of the vertices:
  2:   4
  3:   4
  5:   5
  6:   4
  7:   4
  8:   5
  9:   4
 10:   4
Minimum cost is   4 for vertex   7
(vertex number chosen randomly from
those of min cost).
```



Choose in advance the number of iterations to do.

Or do this for a given amount of time.

Or do this until it has been a long time with no improvement to the optimal solution and then maybe randomly restart.

You can start with any set S.
Maybe S= V(G) is an interesting choice?