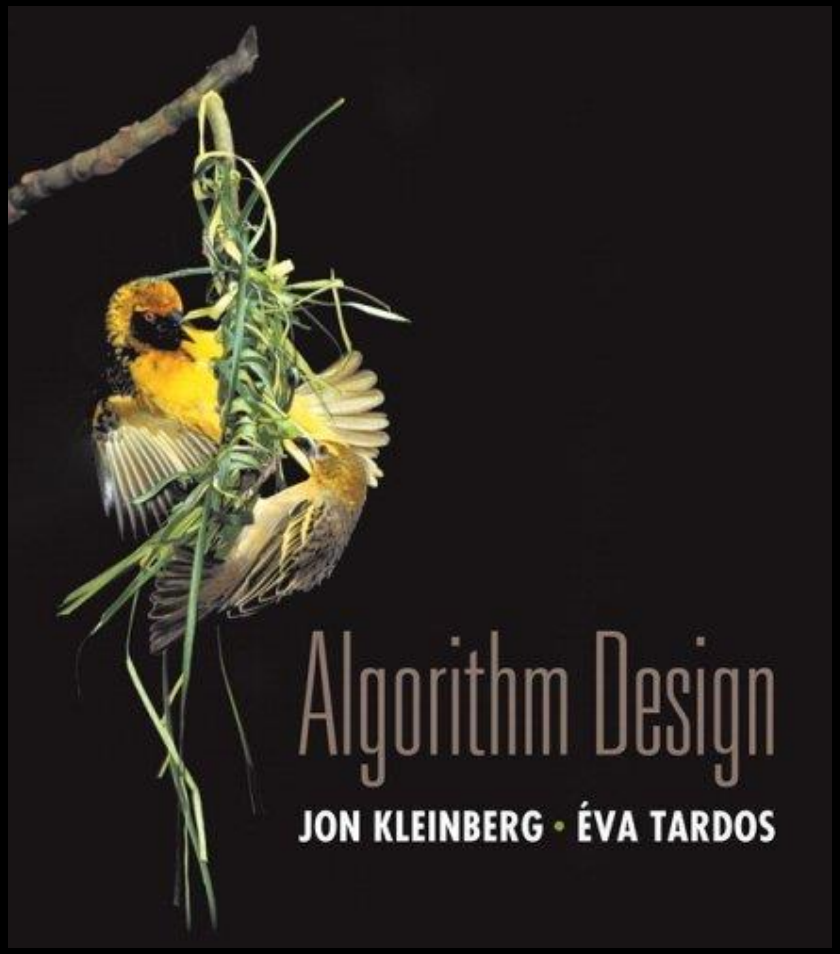


Chapter 12

Local Search

Coping With NP-Hardness

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.

- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

12.1 Landscape of an Optimization Problem

Gradient Descent: Vertex Cover

VERTEX-COVER. Given a graph $G = (V, E)$, find a subset of nodes S of minimal cardinality such that for each u-v in E, either u or v (or both) are in S.

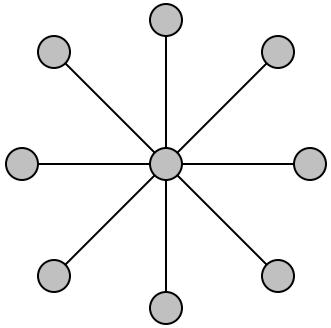
Neighbor relation. $S \sim S'$ if S' can be obtained from S by adding or deleting a single node. Each vertex cover S has at most n neighbors.

Gradient descent. Start with $S = V$. If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S'.

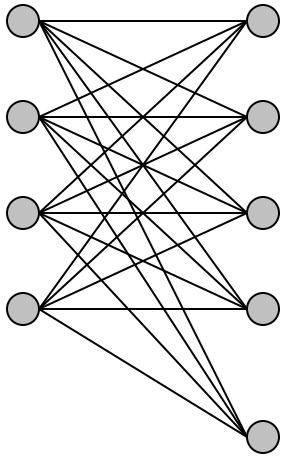
Remark. Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Gradient Descent: Vertex Cover

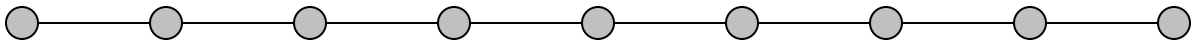
Local optimum. No neighbor is strictly better.



optimum = center node only
local optimum = all other nodes



optimum = all nodes on left side
local optimum = all nodes on right side



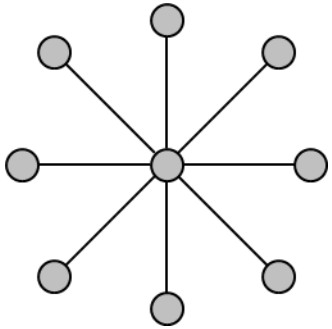
optimum = even nodes
local optimum = omit every third node

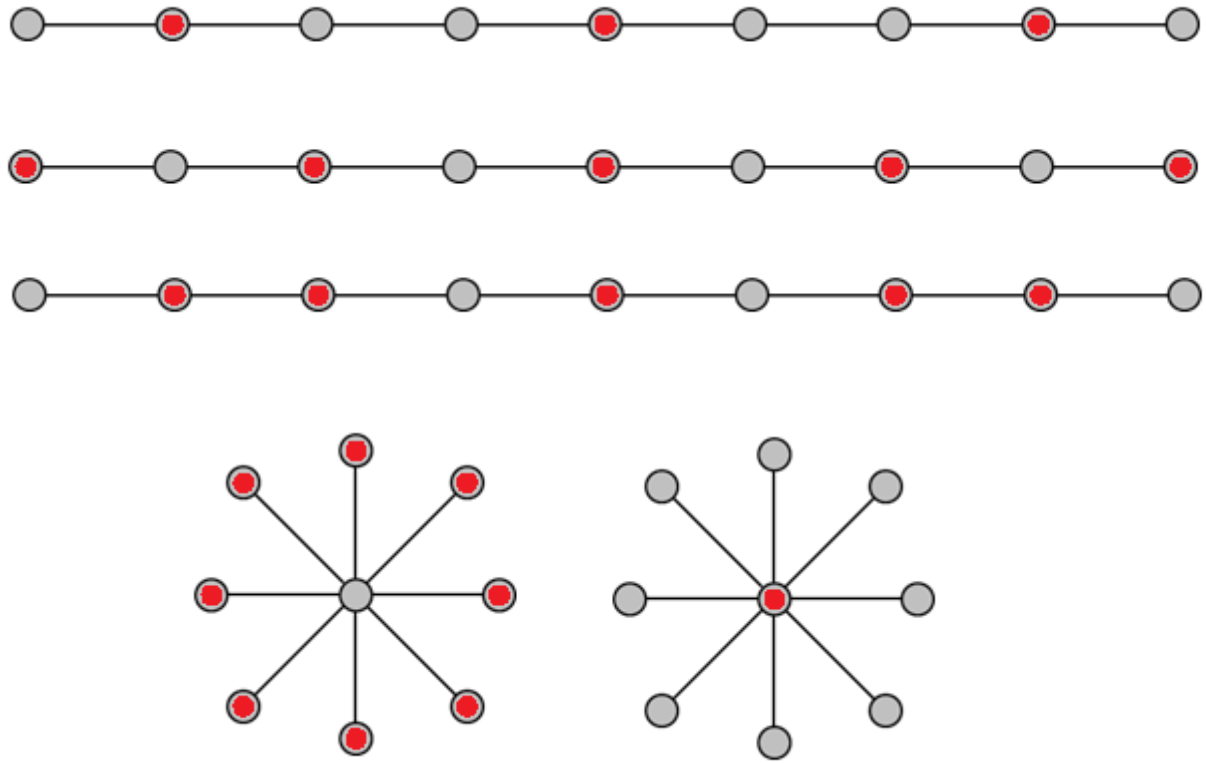
Gradient Descent: Dominating set

Neighbor relation. $S \sim S'$ if S' can be obtained from S by adding or deleting a single node. Each set S has at most n neighbors.

Gradient descent. Start with $S = V$. If there is a neighbor S' that is a dominating set and has lower cardinality, replace S with S'.

Remark. Algorithm terminates after at most n steps since each update decreases the size of the dominating set by one.





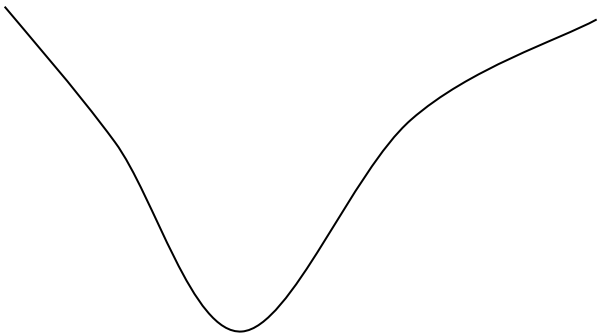
Minimal dominating sets (local optima)

Local Search

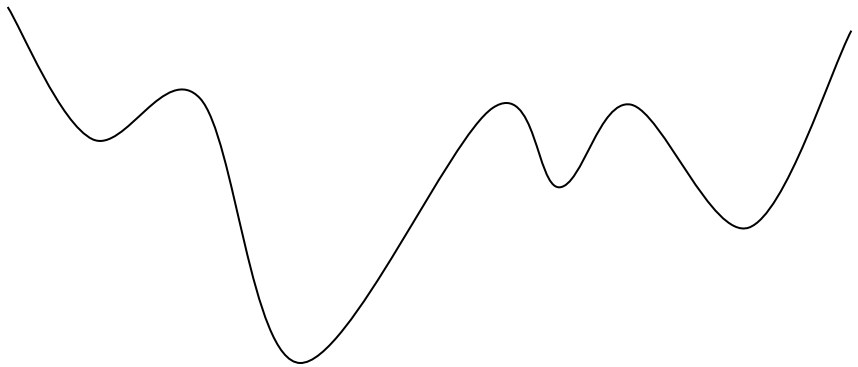
Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbor relation. Let $S \sim S'$ be a neighbor relation for the problem.

Gradient descent. Let S denote current solution. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.



A funnel



A jagged funnel

12.2 Metropolis Algorithm

Metropolis Algorithm

Metropolis algorithm. [Metropolis, Rosenbluth, Rosenbluth, Teller, Teller 1953]

- Simulate behavior of a physical system according to principles of statistical mechanics.
- Globally biased toward "downhill" steps, but occasionally makes "uphill" steps to break out of local minima.

Gibbs-Boltzmann function. The probability of finding a physical system in a state with energy E is proportional to $e^{-E / (kT)}$, where $T > 0$ is temperature and k is a constant.

- For any temperature $T > 0$, function is monotone decreasing function of energy E.
- System more likely to be in a lower energy state than higher one.
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

Metropolis Algorithm

Metropolis algorithm.

- Given a fixed temperature T, maintain current state S.
- Randomly perturb current state S to new state $S' \in N(S)$.
- If $E(S') \leq E(S)$, update current state to S'
  Otherwise, update current state to S' with probability $e^{-\Delta E / (kT)}$,
  where $\Delta E = E(S') - E(S) > 0$.

Theorem. Let $f_S(t)$ be fraction of first t steps in which simulation is in state S. Then, assuming some technical conditions, with probability 1:

$$\lim_{t \to \infty} f_S(t) = \frac{1}{Z} e^{-E(S)/(kT)},$$

$$\text{where } Z = \sum_{S \in N(S)} e^{-E(S)/(kT)}.$$

Intuition. Simulation spends roughly the right amount of time in each state, according to Gibbs-Boltzmann equation.

Simulated Annealing

Simulated annealing.

- T large $\Rightarrow$ probability of accepting an uphill move is large.
- T small $\Rightarrow$ uphill moves are almost never accepted.
- Idea: turn knob to control T.
- Cooling schedule: T = T(i) at iteration i.

Physical analog.

- Take solid and raise it to high temperature, we do not expect it to maintain a nice crystal structure.
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal either.
- Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures.

Simulated annealing algorithm:

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.

A move is chosen randomly.

The move is done if it improves the cost.

Otherwise, it is done with a certain probabilty that depends on the current temperature and how much it increases the cost. The temperature and hence the probability of making a bad move decreases with time.

```
for (i=0; i < n; i++)
{
    num_dom[i]=0;
    min_dom[i]=1;
    S[i]=0;
 }
 min_size=n;
 size=0;

 for (i=0; i < n; i++)
    A[i][i]=1;

 current_cost=n;
```

```
for (trial_num=0; trial_num  < n_trial; trial_num++)
 {
        temperature= 0.98 * temperature;
        random_v= wrand(n);
        new_cost= find_cost(random_v, n, A, size, S, num_dom);

        if (new_cost <= current_cost) do_move=1;
        else
        {
            p1= exp( -1 * (double) (new_cost - current_cost)/
                                                        temperature);
            p2= (double) rand() / (double) RAND_MAX;

            if (p1 > p2) do_move=1;
            else do_move=0;
        }
```

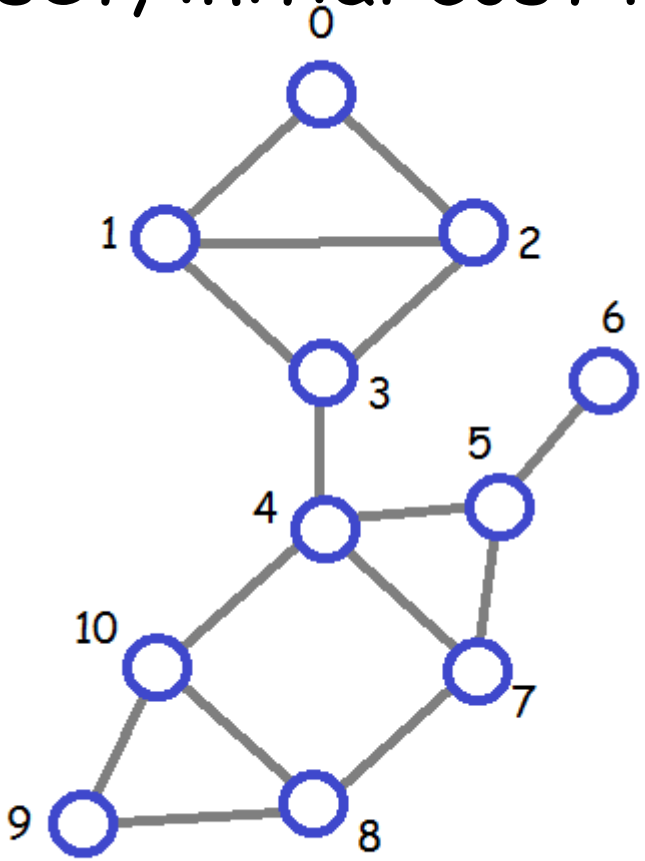
```
if (do_move)
{
      current_cost= new_cost;
      if (S[random_v])
      {
           S[random_v]=0; add= -1; size--;
      }
      else
      {
            S[random_v]=1; add= 1; size++;
      }
      add_to_num_dom(add, n, num_dom, A[random_v]);

      if (is_dom(n, num_dom))
      {
         if (size < min_size)
         {
            min_size=size;
            for (i=0; i < n; i++) min_dom[i]= S[i];
         }
      }
   }
}
```

Alternate formulas

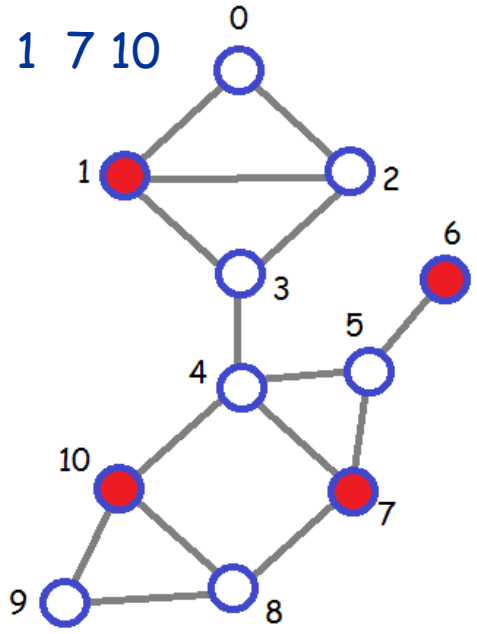
```
for (trial_num=0; trial_num  < n_trial; trial_num++)
 {
        temperature= 1.0 * (n_trial - trial_num) / n_trial;
        random_v= wrand(n);
        new_cost= find_cost(random_v, n, A, size, S, num_dom);

        if (new_cost <= current_cost) do_move=1;
        else
        {
            p1= temperature * 0.5 / (new_cost - current_cost);

            p2= (double) rand() / (double) RAND_MAX;

            if (p1 > p2) do_move=1;
            else do_move=0;
        }
```

Trial 0: S is the empty set, initial cost n.

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

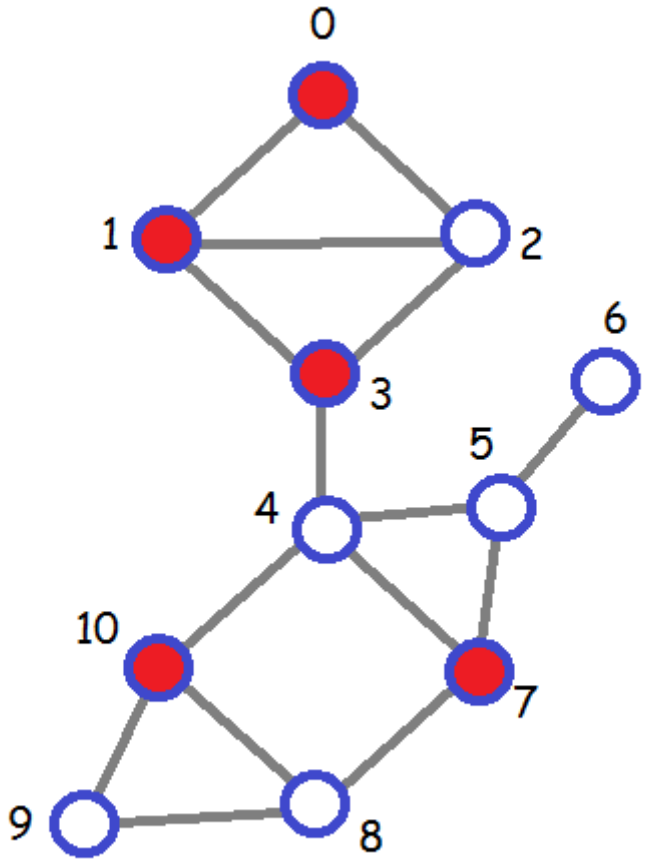


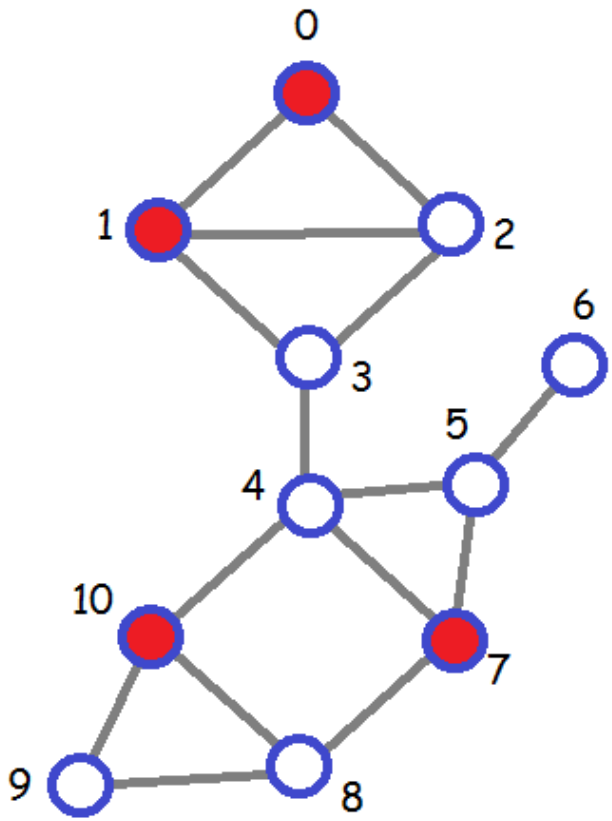
```
Trial   0 Temp 0.78400  11 ->   8: Add  1 BETTER  S=
Trial   1 Temp 0.76832   8 ->   5: Add  7 BETTER  S=  1
Trial   2 Temp 0.75295   5 ->   6: Add  3   0.265   0.798   NO     S=  1  7
Trial   3 Temp 0.73789   5 ->   5: Add  5 SAME    S=  1  7
Trial   4 Temp 0.72314   5 ->   5: Del  5 SAME    S=  1  5  7
Trial   5 Temp 0.70867   5 ->   4: Add 10 BETTER  S=  1  7
Trial   6 Temp 0.69450   4 ->   4: Add  6 SAME    S=  1  7 10
Step    6: Smaller dominating set:  1  6  7 10
```



```
Trial  7 Temp 0.68061   4 ->  4: Del  6 SAME    S=  1  6  7 10
Trial  8 Temp 0.66700   4 ->  7: Del  1   0.011   0.477 NO      S=  1  7 10
Trial  9 Temp 0.65366   4 ->  5: Add  4   0.217   0.365 NO      S=  1  7 10
Trial 10 Temp 0.64059   4 ->  5: Add  2   0.210   0.952 NO      S=  1  7 10
Trial 11 Temp 0.62777   4 ->  5: Add  2   0.203   0.636 NO      S=  1  7 10
Trial 12 Temp 0.61522   4 ->  5: Add  3   0.197   0.142 YES     S=  1  7 10
Trial 13 Temp 0.60291   5 ->  6: Add  2   0.190   0.016 YES     S=  1  3  7 10
Trial 14 Temp 0.59086   6 ->  7: Add  0   0.184   0.137 YES     S=  1  2  3  7 10
Trial 15 Temp 0.57904   7 ->  6: Del  2 BETTER  S=  0  1  2  3  7 10
Trial 16 Temp 0.56746   6 ->  7: Del  7   0.172   0.401 NO      S=  0  1  3  7 10
Trial 17 Temp 0.55611   6 ->  5: Del  3 BETTER  S=  0  1  3  7 10
```

Trial 17 Temp 0.55611 6 -> 5: Del 3 BETTER S= 0 1 3 7 10





Trial 18 Temp 0.54499 5 -> 6: Add 4 0.160 0.999 NO S= 0 1 7 10

Trial 19 Temp 0.53409 5 -> 6: Del 10 0.154 0.513 NO S= 0 1 7 10

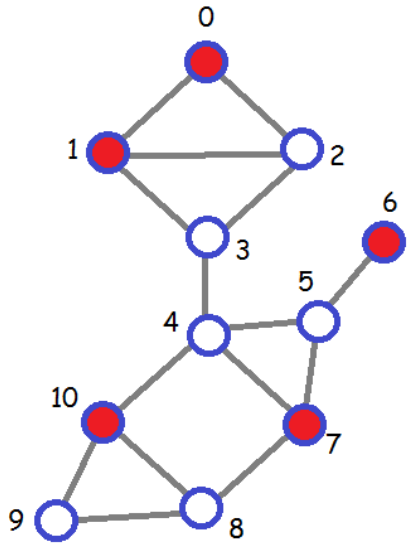
Trial 20 Temp 0.52340 5 -> 6: Add 8 0.148 0.613 NO S= 0 1 7 10

Trial 21 Temp 0.51294 5 -> 6: Add 4 0.142 0.638 NO S= 0 1 7 10

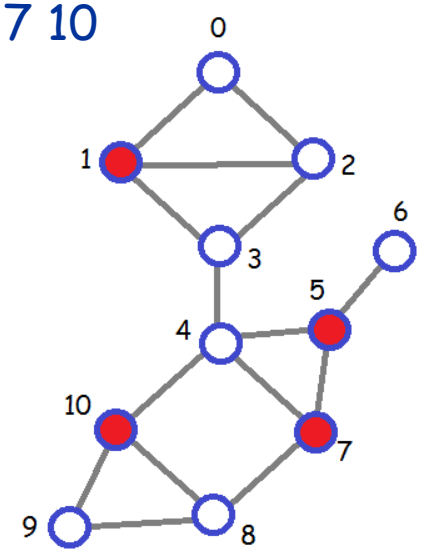
Trial 22 Temp 0.50268 5 -> 6: Add 3 0.137 0.494 NO S= 0 1 7 10

Trial 23 Temp 0.49262 5 -> 5: Add 5 SAME S= 0 1 7 10

Another dominating set: 0 1 5 7 10



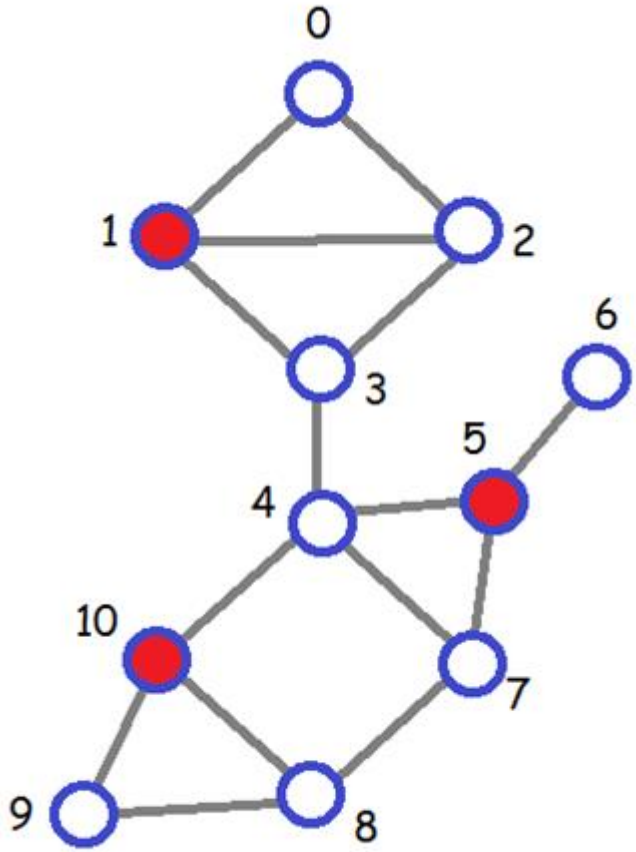
Trial 24 Temp 0.48277 5 -> 5: Del 1 SAME S= 0 1 5 7 10
Trial 25 Temp 0.47312 5 -> 5: Del 5 SAME S= 0 5 7 10
Trial 26 Temp 0.46365 5 -> 6: Add 9 0.116 0.770 NO S= 0 7 10
Trial 27 Temp 0.45438 5 -> 5: Add 1 SAME S= 0 7 10
Trial 28 Temp 0.44529 5 -> 4: Del 0 BETTER S= 0 1 7 10
Trial 29 Temp 0.43639 4 -> 5: Add 2 0.101 0.352 NO S= 1 7 10
Trial 30 Temp 0.42766 4 -> 5: Add 4 0.096 0.919 NO S= 1 7 10
Trial 31 Temp 0.41911 4 -> 5: Add 4 0.092 0.949 NO S= 1 7 10
Trial 32 Temp 0.41072 4 -> 4: Add 5 SAME S= 1 7 10
Another dominating set: 1 5 7 10



Trial 33 Temp 0.40251 4 -> 5: Add 0 0.083 0.192 NO S= 1 5 7 10

Trial 34 Temp 0.39446 4 -> 3: Del 7 BETTER S= 1 5 7 10

Step 34: Smaller dominating set: 1 5 10



Trial 35 Temp 0.38657 3 -> 4: Add 2 0.075 0.349 NO S= 1 5 10

Trial 36 Temp 0.37884 3 -> 4: Add 8 0.071 0.020 YES S= 1 5 10

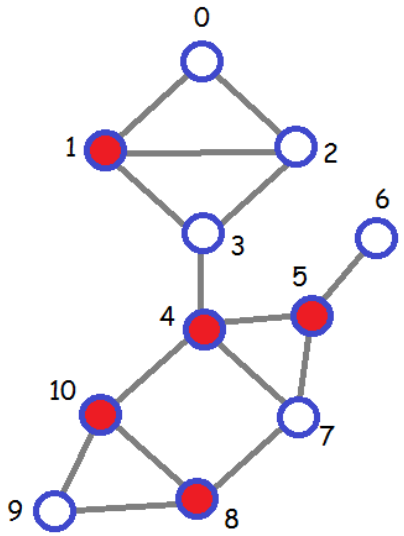
Another dominating set: 1 5 8 10

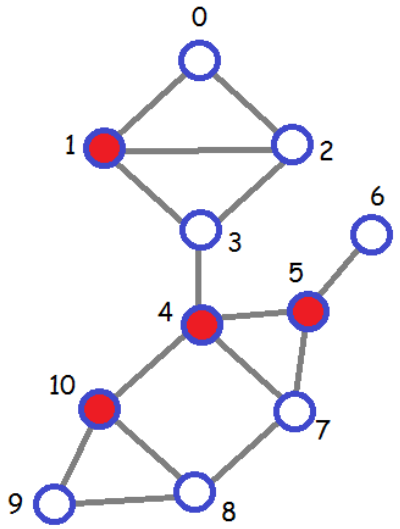
Trial 37 Temp 0.37126 4 -> 5: Add 4 0.068 0.063 YES S= 1 5 8 10

Another dominating set: 1 4 5 8 10

Trial 38 Temp 0.36384 5 -> 4: Del 8 BETTER S= 1 4 5 8 10

Another dominating set: 1 4 5 10





The following results represent running for 100,000 iterations.

This took only 19 seconds to do all the graphs in all the input files.

The BFS and random heuristics were run for 1 second per graph.

C80 fullerenes:

#	n :	k 1	k 2	k 3	k 4	k 5
1	80 :	20	22+	20-	22+	22+
2	80 :	20	22	20-	22	23+
3	80 :	20	22	20-	23+	22
...						
12	80 :	21	22	21-	23+	22
13	80 :	21	22	21-	23+	22
14	80 :	21	22	21-	23+	23+
...						
23	80 :	22	22-	22-	24+	23
24	80 :	22	23	22-	22-	24+
25	80 :	22	22-	22-	23+	23+

C80 fullerenes:

File 1 : c80.txt
File 2 : 1_random/oc80
File 3 : 2_bfs/oc80
File 4 : 3_sim/oc80
File 5 : 4_wsim/oc80

Rank 1: Student 3 with score 0
Rank 2: Student 2 with score 41
==========
Rank 3: Student 4 with score 54
Rank 3: Student 5 with score 54

```
Football distance 1:
File  1 : football_1.txt File  2 : 1_random/of1
File  3 : 2_bfs/of1      File  4 : 3_sim/of1
File  5 : 4_wsim/of1
  #     n :  k 1    k 2    k 3    k 4    k 5
  1     9 :    3      3-     3-     3-     3-
  2    27 :    5      5-     5-     6+     5-
  3    81 :    9      9-     9-    12+     9-
  4   243 :   27     35     36     29-    37+
  5   729 :   73    105+   104     91-   102
  6  2187 :  186    302    290    251-   305+

Rank  1: Student  4 with score  89
Rank  2: Student  3 with score 144
Rank  3: Student  2 with score 156
=====================================
Rank  4: Student  5 with score 158
```

```
Football distance 2:

File  1 : football_2.txt   File  2 : 1_random/of2
File  3 : 2_bfs/of2        File  4 : 3_sim/of2
File  5 : 4_wsim/of2
  #     n :  k 1    k 2    k 3    k 4    k 5
  1    27 :    3      3-     3-     3-     3-
  2    81 :    3      3-     3-     5+     3-
  3   243 :    8      9-     9-    12+    10
  4   729 :   17     25     25     22-    30+
  5  2187 :   34     65     70     58-    72+

Rank  1: Student  4 with score  35
Rank  2: Student  2 with score  40
=====================================
Rank  3: Student  3 with score  45
Rank  4: Student  5 with score  53
```

```
Football distance 3:

File  1 : football_3.txt
File  2 : 1_random/of3
File  3 : 2_bfs/of3
File  4 : 3_sim/of3
File  5 : 4_wsim/of3
  #      n :  k 1    k 2    k 3    k 4    k 5
  1     81 :    3      3-     3-     3-     3-
  2    243 :    3      3-     3-     5+     3-
  3    729 :    6      7-     8      8     10+
  4   2187 :   12     19-    21     21     27+

Rank  1: Student  2 with score   8
=======================================
Rank  2: Student  3 with score  11
Rank  3: Student  4 with score  13
Rank  4: Student  5 with score  19
```

```
ypercube distance 1:

File  1 : hypercube_1.txt  File  2 : 1_random/oh1
File  3 : 2_bfs/oh1        File  4 : 3_sim/oh1
File  5 : 4_wsim/oh1
  #      n :  k 1    k 2    k 3    k 4    k 5
  1      8 :    2      2-     2-     2-     2-
  2     16 :    4      4-     4-     4-     4-
  3     32 :    7      7-     7-     7-     7-
  4     64 :   12     12-    12-    13+    13+
  5    128 :   16     16-    22+    22+    16-
  6    256 :   32     45+    44     40-    45+
  7    512 :   62     88+    85     72-    84
  8   1024 :  120    169    164    141-   194+
  9   2048 :  192    335    316    276-   712+
Rank  1: Student  4 with score 130
Rank  2: Student  3 with score 209
Rank  3: Student  2 with score 231
=====================================
Rank  4: Student  5 with score 630
```

Hypercube distance 2:

File 1 : hypercube_2.txt File 2 : 1_random/oh2
File 3 : 2_bfs/oh2 File 4 : 3_sim/oh2
File 5 : 4_wsim/oh2

#	n :	k 1	k 2	k 3	k 4	k 5
1	32 :	2	2-	2-	2-	2-
2	64 :	4	4-	4-	5+	4-
3	128 :	7	7-	7-	8+	8+
4	256 :	12	13-	13-	13-	15+
5	512 :	16	24-	24-	24-	27+
6	1024 :	30	45	43	42-	46+
7	2048 :	44	81	80	72-	91+

Rank 1: Student 4 with score 51
Rank 2: Student 3 with score 58
Rank 3: Student 2 with score 61
==========
Rank 4: Student 5 with score 78

Hypercube distance 3:

File 1 : hypercube_3.txt
File 2 : 1_random/oh3
File 3 : 2_bfs/oh3
File 4 : 3_sim/oh3
File 5 : 4_wsim/oh3

#	n	k 1	k 2	k 3	k 4	k 5
1	128	2	2-	2-	4+	2-
2	256	4	4-	4-	5+	4-
3	512	7	8-	8-	10+	9
4	1024	12	15-	16	15-	18+
5	2048	16	26-	28	26-	34+

Rank 1: Student 2 with score 14
==========
Rank 2: Student 3 with score 17
Rank 3: Student 4 with score 19
Rank 4: Student 5 with score 26

Kneser Graphs:

#	n	k 1	k 2	k 3	k 4	k 5
1	35	7	7-	7-	7-	7-
2	56	7	7-	7-	7-	7-
3	84	7	7-	7-	8+	7-
4	120	6	6-	6-	6-	6-
5	165	5	5-	5-	5-	6+
6	220	4	4-	4-	6+	5
7	286	4	4-	4-	5+	5+
8	126	26	28	28	26-	29+
9	210	15	22	23+	15-	21
10	330	15	19-	19-	20+	19-
11	495	12	17	17	16-	18+
12	715	10	15-	15-	15-	16+
13	462	66	103+	102	89-	98
14	792	56	78	86+	69-	80
15	1287	39	61	66+	56-	66+

```
Rank  1: Student  4 with score  71
Rank  2: Student  2 with score 104
=====================================
Rank  3: Student  5 with score 111
Rank  4: Student  3 with score 117
```

Queen Graphs:

#	n :	k 1	k 2	k 3	k 4	k 5
1	16 :	2	2-	2-	2-	2-
2	25 :	3	3-	3-	3-	3-
3	36 :	3	3-	3-	4+	3-
4	49 :	4	4-	4-	5+	5+
5	64 :	5	5-	5-	6+	5-
6	81 :	5	5-	5-	7+	6
7	100 :	5	5-	6	7+	7+
8	121 :	5	5-	7	8+	8+
9	144 :	6	8-	8-	8-	9+
10	169 :	7	8-	9	10+	10+
11	196 :	8	9-	10	11+	11+
12	225 :	9	10-	10-	12+	11
13	256 :	9	11-	12+	12+	12+
14	289 :	9	13	12-	13	14+
15	324 :	9	14+	14+	13-	13-

```
 16   361 :   10     14-    14-    14-    16+
 17   400 :   11     16-    16-    17+    17+
 18   441 :   11     17     16-    18+    17
 19   484 :   12     18     18     17-    19+
 20   529 :   12     19     20     17-    21+
 21   576 :   13     20-    21+    20-    21+
 22   625 :   13     22+    21-    21-    22+
 23   676 :   14     21-    24+    22     23
 24   729 :   14     23     24     22-    26+
 25   784 :   15     25     26     23-    27+
 26   841 :   15     25     26+    23-    25
 27   900 :   15     27     28     24-    30+

Rank  1: Student  2 with score 108
=====================================
Rank  2: Student  4 with score 115
Rank  3: Student  3 with score 120
Rank  4: Student  5 with score 139
```