**CSI 5165 Combinatorial Algorithms**
**Assignment 2**
Wei Li 0300113733 report date: 3/12/2021

# 1 Environment

System: Windows 10
Hardware: Intel core I7 9700; 8+8 GB speed 2667;
Language: Python 3.8.3

# 2 Question 1.a

Develop a hill-climbing algorithm, a simulated annealing algorithm, a tabu search algorithm and a genetic algorithm for the Maximum Clique problem (the problem of finding a clique of maximum cardinality in a graph).

- choices for neighbourhood function
- pseudocode for each algorithm
- mating scheme, mutation, etc, for genetic algorithm
- each algorithm, write a paragraph explaining which parameter variations you recommend

## 2.1 Formulate Maximum Clique as Heuristic Problem

In this section we define the problem and fromulate the problem as a heuristic problem for Hill climbing , Tabu Search, and simulated Annealing. The Genetic Algorithm use as different represent and there is a separate discussion will be on section 2.5.

We use the following notation for Maximum Clique Problem (MCP). Given a undirected graph $G = (V, E)$, the subgraph $SG = (V', E')$ of $G$ is one such that $V' \subseteq V$ and $E' \subseteq E$. Clique is complete subgraphs i.e. whose vertices are all adjacent. We want to find a clique whose cardinality is maximized. We notate a vertex as $v$, and $AS(v)$ is its adjacent vertices set.

For the first three heuristic solutions, we limit the search space to the set of all possible clique. By considering only and feasible clique, a solution $S$ can be completely represented by a subset of V: $S \subseteq V$ as all edges are connected. The fitness function is the cardinality of the solution $f(S) = |S|$. Given a solution $S$, $CS(S)$ is the set of vertices in $V - S$ that connect to all vertices in S, ie the choose set. The neighbourhood structure of $S$ can be defined as $N(S) = N^-(S) \cup N^+(S)$, where $N^-(S) = \{S' | S' = S - \{v\}, v \in S\}$ ie. the set of all solutions by dropping one vertex from $S$, and $N^+(S) = \{S' | S' = S \cup \{v\}, v \in CS(S)\}$, ie. the set of all solutions by adding one vertex from $CS(S)$ to $S$.

This notation and neighbourhood function will be used in hill-climbing algorithm, simulated annealing algorithm and tabu search algorithm.

### 2.2 Hill Climbing

One observation to the neighbourhood structure is that choosing a neighbourhood solution from $N^+(S)$ can provide plus 1 profit. As Hill Climbing dose not consider down-hill operation, we always draw solutions from $N^+(S)$. As the fitness function gives no clue on the selection $S'$ from $N^+(S)$, one way is uniformly sample one solution from $N^+(S)$. Another way is greedily choose the next solution $S'$ such that its candidate set is maximized,

$$S' = \arg\max_{S' \in N^+(S)} CS(S')$$

$CS(S')$ can be computed from $CS(S') = CS(S) \cup AS(v)$.

We randomly initialize the initial solution (single $v \in V$) and do local hill climbing $ms$ times and take the best local solution.

In algorithm 2, line [9 - 16] is where we do greedy search to find which $v$ in $CS(S)$, after adding

to $S'$, lead to the largest $CS(S')$.

**Algorithm 1:** HillClimb_random(ms)

**Result:** $S_{best}$

$c \leftarrow 0$;
$l_{best} \leftarrow 0$;
$S_{best} \leftarrow \phi$;
**while** $c < ms$ **do**
  $S \leftarrow \{v\}$ , $v$ randomly selected from $V$ $l \leftarrow 1$;
  $CS_{cur} \leftarrow AS(v)$;
  **while** $CS_{cur}$ *is not* $\phi$ **do**
    $S \cup \{v\}$ , $v$ randomly selected from $CS_{cur}$;
    $l \leftarrow l + 1$;
    $CS_{cur} \leftarrow CS_{cur} \cap AS(v)$;
  **end**
  **if** $l > l_{best}$ **then**
    $l_{best} \leftarrow l$;
    $S_{best} \leftarrow S$;
  **end**
  $c \leftarrow c + 1$
**end**

**Algorithm 2:** HillClimb_greedy(ms)

**Result:** $S_{best}$

$c \leftarrow 0$;
$l_{best} \leftarrow 0$;
$S_{best} \leftarrow \phi$;
**while** $c < ms$ **do**
  $S \leftarrow \{v\}$ , $v$ randomly selected from $V$;
  $l \leftarrow 1$;
  $CS_{cur} \leftarrow AS(v)$;
  **while** $CS_{cur}$ *is not* $\phi$ **do**
    $Y_{best} \leftarrow \phi$;
    $best\_size \leftarrow 0$;
    **for** $v \in CS_{cur}$ **do**
      **if** $|CS_{cur} \cap AS(v)| > best\_size$ **then**
        $Y_{best} \leftarrow \{v\}$;
        $CS\_size \leftarrow |CS_{cur} \cap AS(v)|$
      **end**
    **end**
    $Y \leftarrow Y_{best}$;
    $S \cup Y$;
    $l \leftarrow l + 1$;
    $CS_{cur} \leftarrow CS_{cur} \cap AS(v)$;
  **end**
  **if** $l > l_{best}$ **then**
    $l_{best} \leftarrow l$;
    $S_{best} \leftarrow S$;
  **end**
  $c \leftarrow c + 1$
**end**

The only parameter in Hill Climbing is the max number of search $ms$.

### 2.3 Simulated Annealing

In Simulated Annealing, besides $N^+(S)$ (adding operation) we also consider $N^-(S)$ (dropping operation) that lead to a inferior fitness.

When doing dropping, one vertex $v$ from $S$ is randomly select and deleted from $S$. However, unlike adding, dropping requires recalculate $CS(S)$ from scratch. Given a $CS(S)$ and one vertex $v$ dropped, the next $CS(S')$ can be calculated by:

$$CS(S') = V \bigcap_{v_i \in S - \{v\}} AS(v_i)$$

*(PS:We tried to make computing $CS(S')$ after a dropping more efficient (ie. without the need to compute all vertices in clique) by utilizing some data structure, but the discussion is not complete. The incomplete discussion is given in Appendix.)*

In maximum clique, the higher the profit the more limited the search space. On the other hand, droping vertices from $S$ offers a way to widen the search space. We want to explore more in early phase of search and gradually converge to local search.

In the algorithm, we make both adding and dropping stochastic. Flipping a coin, the search either go up-hill or down-hill, but the possibility of go down-hill should decrease as the search proceeding.
The conditions to choose solution from $N^-(S)$ are:

- uniformly sample $r$ from $[0, 1]$. If $r$ is smaller than $e^{\frac{-2}{T}}$, where $T$ is the temperature
- the solution cannot be improved anymore, ie. $CS(S) = \phi$
- there is no hope solutions in the branch to exceed the current optimal solution. This condition is defined by: $|S| + |CS(S)| \leq l_{best}$. It is obvious that $|S| + |CS(S)|$ is an upper bound to $|S'|$ from the current solution $S$.

In other case, the next solution is chosen from $N^+(S)$.
The second and third conditions are also used in Tabu Search.

**Algorithm 3:** SimulatedAnnealing($T_0, \alpha, ms$)

**Result:** $S_{best}$

$S \leftarrow \{v\}$ , $v$ randomly selected from $V$;
$S_{best} \leftarrow S$;
$l \leftarrow 1$;
$l_{best} \leftarrow 1$;
$CS_{cur} \leftarrow AS(v)$;
$T \leftarrow T_0$;
$c \leftarrow 0$;
**while** $c < ms$ **do**
    $r \leftarrow random(0, 1)$;
    **if** *($r < e^{\frac{-2}{T}}$) OR ($CS_{cur}$ is $\phi$) OR ($|S| + |CS_{cur}| \leq l_{best}$)* **then**
        **if** $S$ *is not* $\phi$ **then**
            $S \leftarrow S - \{v\}$, $v$ randomly selected from $S$;
            $CS_{cur} = V$;
            **for** $v \in S$ **do**
                $CS_{cur} \leftarrow CS_{cur} \cap AS(v)$;
            **end**
            $l \leftarrow l - 1$
        **else**
            Continue to next iteration;
        **end**
    **else**
        $S \leftarrow S \cup \{v\}$, $v$ randomly selected from $CS_{cur}(S)$;
        $CS_{cur} = CS_{cur} \cap AS(v)$;
        $l \leftarrow l + 1$;
        **if** $l > l_{best}$ **then**
            $l_{best} \leftarrow l$;
            $S_{best} \leftarrow S$;
        **end**
    **end**
    $c \leftarrow c + 1$;
    $T \leftarrow \alpha \times T$;
**end**

Hyperparameters used in Simulated Annealing are: 1) $T_0$ initial temperature. 2) $\alpha \in (0, 1)$ the linear cooling factor. 3) max number of update $ms$. A combination that works is $T_0 = 20$, $\alpha = 0.98$, $max\_iterations = 50000$.

## 2.4 Tabu Search

In tabu search, the tabu list $L$ is: the last $|L|$ vertices that have been added or dropped recently (both in $N^+(S)$ or $N^-(S)$ direction). The tabu list is shared by both adding and dropping so one cannot add a vertex directly after dropping it, and vice versa.

We define the way to select $S'$ from $N^+(S)$ and $N^-(S)$ as follow:

- if $CS(S) = \phi$, **randomly select** $k$ **vertices** from $S$ and drop them, provided that the vertices not in the Tabu list. This change will not be added to Tabu List. The $k$ should

adapt to $|S|$. (eg. $k = \lfloor \frac{|S|}{3} \rfloor$)

- else,
  - exclude vertices in tabu list $L$ from $N^+(S)$ and $N^-(S)$
  - if $|S| + |CS_{cur}| \leq l_{best}$, randomly drop one vertex and recalculate $CS(S)$
  - else flip a coin, with a possibility $r$

$$r = \begin{cases} 0.15 & \text{if } 0 < |S| < \frac{1}{4} l_{best} \\ 0.3 & \text{if } \frac{1}{4} l_{best} \leq |S| < \frac{1}{2} l_{best} \\ 0.5 & \text{if } \frac{1}{2} l_{best} \leq |S| < \frac{3}{4} l_{best} \\ 0.1 & \text{if } \frac{3}{4} l_{best} \leq |S| < l_{best} \end{cases}$$

    drop a node at random, else add a vertex at random.

The reason of the two adaptations is of two folds:
1) as discussed before, dropping vertices, comparing to adding vertices, is much more expensive. Also, it is expected that when $|CS(S)|$ is small, dropping should happen more frequently. So we want to decrease the number of recalculations by dropping as a batch when $|CS(S)| = \phi$.
2) When $|S|$ is close to the $l_{best}$ the search space should very limited, since the search space is limited, one may want to exploit the space to see what the best result could be, so it is desirable to choose $N^+(S)$. When $|S|$ is neither too short or too long, we should balance the dropping and adding. Dropping when $|S|$ is small, on the other hand, may easily lead to very different search region, which should also be avoided to happen too frequently. As a result, given the $l_{best}$, the possibility of choosing from $N^-(S)$ should approximately positive correlate the length of solution $|S|$ but reversed when $|S|$ is long enough. We design a simple step function.

In tabuList parameters are 1) $L_0$: the size of tabu list; 2 )$ms$ max search; 3)The distribution of the $r$. Many factors may affect the best $L_0$, intuitively $L_0$ can be proportional to the know best $l$, eg. result from other search methods, or be proportional the average choose set size. A suggest $r$ distribution is given in previous discussion, one may replace it with other distributions or functions.

**Algorithm 4:** TabuSearch($L_0, ms$)

**Result:** $S_{best}$

$S \leftarrow \{v\}$ , $v$ randomly selected from $V$;
$S_{best} \leftarrow S$;
$l \leftarrow 1$;
$l_{best} \leftarrow 1$;
$CS_{cur} \leftarrow AS(v)$;
$L \leftarrow$ empty queue of max size $L_0$;
$c \leftarrow 0$;
**while** $c < ms$ **do**
  **if** $CS_{cur} = \phi$ **then**
    $k \leftarrow \lfloor \frac{|S|}{3} \rfloor$;
    $Y = \{v_1, \ldots, v_k\}$ $v_1, \ldots, v_k$ randomly choose from $S - L$;
    $S \leftarrow S - Y$;
    $CS_{cur} = V$;
    **for** $v \in S$ **do**
      $CS_{cur} \leftarrow CS_{cur} \cap AS(v)$;
    **end**
    $l \leftarrow l - k$
  **else**
    $N^+(S) \leftarrow CS_{cur} - L$;
    $N^-(S) \leftarrow S - L$;
    $r \leftarrow random(0, 1)$;
    **if** $(0 < |S| < \frac{1}{4} l_{best}$ *AND* $r < 0.15)$ *OR* $(\frac{1}{4} l_{best} \leq |S| < \frac{1}{2} l_{best}$ *AND* $r < 0.3)$ *OR* $(\frac{1}{2} l_{best} \leq |S| < \frac{3}{4} l_{best}$ *AND* $r < 0.5)$ *OR* $(\frac{3}{4} l_{best} \leq |S| < l_{best}$ *AND* $r < 0.1)$ *OR* $(|S| + |CS_{cur}| \leq l_{best})$ **then**
      $S \leftarrow S - \{v\}$, $v$ randomly selected from $N^-(S)$;
      $CS_{cur} = V$;
      **for** $v \in S$ **do**
        $CS_{cur} \leftarrow CS_{cur} \cap AS(v)$;
      **end**
      $l \leftarrow l - 1$;
      **if** $|L| + 1 > L_0$ **then**
        L.pop();
      **end**
      L.add(v);
    **else**
      $S \leftarrow S \cup \{v\}$, $v$ randomly selected from $N^+(S)$;
      $CS_{cur} = CS_{cur} \cap AS(v)$;
      $l \leftarrow l + 1$;
      **if** $l > l_{best}$ **then**
        $l_{best} \leftarrow l$;
        $S_{best} \leftarrow S$;
      **end**
      **if** $|L| + 1 > L_0$ **then**
        L.pop();
      **end**
      L.add(v);
    **end**
  **end**
  $c \leftarrow c + 1$
**end**

## 2.5 Genetic Algorithm

In genetic algorithm we no longer search in feasible space, but also consider infeasible solutions, in other word, subgraphs $SG = (V', E')$ of $G$.

The reason to search in $SG$ is that, in addition to the fact that clique are difficult to identify, it is hard to make sure a clique population, after crossing and mutation, is still feasible. Also, the neighborhood structure of clique is too constrained, thus make it hard to utilize the advantage of population.

We still represent a solution as a set of vertices $S$, although the edge set of the subgraph may no longer be complete.

### 2.5.1 Mutation

To address the non-feasibility issue of search, we consider Marchiori's repair heuristic. (Marchiori 2002) The algorithm is consisted of two phases: repair and extension. Repair takes a subgraph represented as a vertices set $U$ and a probability $\alpha$ as input and output a clique $S$ in $G$.

**Algorithm 5:** repair($U$, $\alpha$)

```
   Result: S
 1 W ← U; S ← U;
 2 while W ≠ φ do
 3 |  W ← W − {k}, k randomly selected from W;
 4 |  r ← random(0, 1);
 5 |  if r < α then
 6 |  |  S ← S − {k};
 7 |  else
 8 |  |  for {v|v ∈ S and (v, k) ∉ E} do
 9 |  |  |  W ← W − {v};
10 |  |  |  S ← S − {v};
11 |  |  end
12 |  end
13 end
```

The algorithm is basically randomly select a node $k$ and flip a coin with probability $\alpha$. Based on the result either remove $k$ from $S$ or remove vertices that not connect to $k$ from $U$. Repeat removing until the set $S$ is a clique. $\alpha$ should be small so that there is minimal lose of size from $U$ to $S$.

Then the extension phase is extending a clique $S$ to its maximal size at random with minimal cost. It is easy to see *extension(S)* can be replaced by the HillClimbing local search or other light-cost local searches for better local search quality.

**Algorithm 6:** extension($S$)

**Result:** $S$

$W \leftarrow V - S$; **while** $W \neq \phi$ **do**
  $W \leftarrow W - \{k\}$, $k$ randomly selected from $W$;
  **if** *$k$ connects to all vertices in $S$* **then**
    $S \leftarrow S + \{k\}$;
  **end**
**end**

### 2.5.2 Mating

To do mating, we sort solutions in a population $P$ based on their length(fitness), and randomly swap $m$ vertices between two adjacent solutions.

The reason to sort the solution is of two fold, firstly, the length of solutions in a population can vary wildly. Doing sorting allows that adjacent solutions have balanced length. Secondly, although naive, this makes better solutions able to mate with each other.

Given two adjacent solutions $S_i$, $S_{i+1}$ in a sorted population, we set $m$, the number of vertices to be swap, as $m = \beta \times min\{|S_i|, |S_{i+1}|\}$. ($0 < \beta < 1$, means take a fraction of the smaller length of two parents, eg. $\beta = \frac{1}{4}$).

Assuming even number of solution in a population, the whole mating algorithm is:

**Algorithm 7:** mating(*parents*)

**Result:** $offspring$

$parents \leftarrow list(parents)$;
sort *parents* based on the length of solution;
$offspring \leftarrow \phi$;
**for** *($i = 0$ ; $i < |parents|$; $i = i + 2$)* **do**
  $m = \frac{1}{4} \times min\{|S_i|, |S_{i+1}|\}$;
  $u \leftarrow$ randomly selected $m$ vertices in $S_i$;
  $v \leftarrow$ randomly selected $m$ vertices in $S_{i+1}$;
  $S_i \leftarrow (S_i - u) \cup v$ ;
  $S_{i+1} \leftarrow (S_{i+1} - v) \cup u$ ;
  $offspring \cup \{S_i, S_{i+1}\}$
**end**

To sum up, we use random exchange as mating scheme, and use the Marchiori's repair heuristic and local search as mutation. The selection is simply pick $|P|$ best individuals from parents and offspring. Given population size $ps$ and max generation $mg$, The whole algorithm is:

[1-6] population initialization; [12] mating; [14-18] mutation; [19-21] selection;

**Algorithm 8:** geneticAlgorithm($ps, mg, \alpha$)

```
   Result: S_best
 1 P ← φ;
 2 for i in range 0 to ps − 1 do
 3 |   S_i ← {v}, v randomly selected from V;
 4 |   S_i ← extension(S_i);
 5 |   P ∪ {S_i}
 6 end
 7 S_best ← arg max_{S∈P} |S|;
 8 l_best ← |S_best|;
 9 gen ← 0;
10 while gen < mg do
11 |   parents ← P;
12 |   offspring ← mating(parents);
13 |   Q ← emptylist;
14 |   for S ∈ offspring do
15 |   |   S ← repair(S, α);
16 |   |   S ← extension(S);
17 |   |   Q.add(S);
18 |   end
19 |   add all solutions in parent to Q;
20 |   sort Q based on the length of solutions |S|;
21 |   P ← first ps solutions in Q;
22 |   if |Q[0]| > l_best then
23 |   |   S_best ← Q[0];
24 |   |   l_best ← |Q[0]|;
25 |   end
26 |   gen ← gen + 1;
27 end
```

Parameters are

- 1)the possibility of repair $\alpha$, recommend to be small as per Marchiori 2002.
- 2) $m$ number of vertices to be swap in mating (recommend to be proportional to the two parent's size ($\beta \times min\{|S_i|, |S_{i+1}|\}$)
- 3) $ps$ population size: maybe a fraction to the total possible number of Subgragh, also subject to hardware and computation resource
- 4) $mg$ max generation: according to computation resource

# 3 Question 1.b

Choose 2 of the 4 algorithms developed for A2-Q3 and implement them.

- find maximum cliques for each of the 6 given graphs
- each graph and each algorithm, experiment with several parameter variations for the algorithm

- a table summarizing your results (tabulate time, number of iterations, largest clique found, etc for each algorithm and parameter variation considered)
- a conclusion

We implement the Hill Climbing and Simulated Annealing. For hill climbing, we compare the steepest hill climbing and the vanilla hill climbing. For Simulated Annealing, we compare different inital temperature and $T_0$ and cooling factor $\alpha$ combination. The implementation can be found in Colab.

### 3.1 Experiment

We run local hill climbing and steepest hill climbing 10000 times and reports the best result ($max\_l$), the average result ($avg\_l$), and the total runtime ($rtime$).

For Simulated Annealing, we experiment all combination of $T_0$ and $\alpha$ from $T_0 = \{5, 20, 50\}$, $\alpha = \{0.9, 0.98, 0.995\}$. For each run the max number of update $ms$ is set to 10000. For each problem and parameter combination we run Simulated Annealing 100 times and report the best result ($max\_l$), the average result ($avg\_l$), and the average runtime ($rtime$) of the 100 searches. The combinations are encoded as $[T\_0, \alpha]$ in the table. There are 9 total parameters combinations, the result are most of time homogeneous and the differences are negligible, so only interesting results are reported.
The result are reported in Table 1.
Most of time, at least 1 out of the 10000 trials, the hill climbing algorithm(both variation) can find the optimum solution. The two exception occurs in randv100d30 and randv100d70 searched by steepest hill climbing.
Also, on average, hill climbing always generate inferior result to simulated annealing. Comparing the two hill climbing variation, the steepest hill climbing tend to work better and run faster when the graph size is small, but deteriorate quickly when the graph size and edges grows.

For the first 5 problems, all simulated annealing combination can find optimum solution 100% of times. the only difference is the problem *randv100d70*. We report the two parameter combinations that obtains the worst average length and three parameter combinations that obtains the best average length. It can be observed that the worst two are exactly extreme cases, i.e. highest initial temperature cooling at the slowest speed and lowest initial temperature cooling at the fastest speed.

## 4 Appendix: incomplete discussion on improving dropping efficiency

We consider both $N^+(S)$ (adding operation) and $N^-(S)$, ie. dropping operation.
To update the choose set $CS(S)$ more efficiently without recomputing it from scratch after a dropping, we maintain a set $OM$ ie One Missing, which stores the set of vertices that connect to $|S| - 1$ vertices of $S$, provided that the $\forall v \in OM, v \notin S$, and which vertices they are not connecting to. In other word, it is a set with a partition $OM = MV_0 + MV_1 + \cdots + MV_i + \ldots$ where every two subsets does not intersect. The number of subsets should be the same as the number of vertices in $S$, provided the subset is not empty.

A trival example illustrate the way to update $OM$ after adding operation. Suppose $S$ only consist of one vertex $\{1\}$ in time step 1, $OM^1$ is all vertices that does not connect to vertex 1,

| | optimum | | | max_l | avg_l | rtime | optimum_find |
|---|---|---|---|---|---|---|---|
| graphv16_m30 | 3 | HC | rand | 3 | 2.430 | 0.091 | 1 |
| | | | steepest | 3 | 2.369 | 0.042 | 1 |
| | | SA | [5, 0.9] | 3 | 3 | 0.080 | 1 |
| | | | [50, 0.995] | 3 | 3 | 0.063 | 1 |
| | | | [50, 0.98] | 3 | 3 | 0.073 | 1 |
| | | | [5, 0.995] | 3 | 3 | 0.077 | 1 |
| | | | [20, 0.98] | 3 | 3 | 0.076 | 1 |
| graphv16_m60 | 5 | HC | rand | 5 | 3.973 | 0.168 | 1 |
| | | | steepest | 5 | 4.001 | 0.083 | 1 |
| | | SA | [5, 0.9] | 5 | 5 | 0.082 | 1 |
| | | | [50, 0.995] | 5 | 5 | 0.066 | 1 |
| | | | [50, 0.98] | 5 | 5 | 0.080 | 1 |
| | | | [5, 0.995] | 5 | 5 | 0.081 | 1 |
| | | | [20, 0.98] | 5 | 5 | 0.079 | 1 |
| graphv16_m90 | 7 | HC | rand | 7 | 5.714 | 0.249 | 1 |
| | | | steepest | 7 | 6.442 | 0.160 | 1 |
| | | SA | [5, 0.9] | 7 | 7 | 0.084 | 1 |
| | | | [50, 0.995] | 7 | 7 | 0.068 | 1 |
| | | | [50, 0.98] | 7 | 7 | 0.085 | 1 |
| | | | [5, 0.995] | 7 | 7 | 0.084 | 1 |
| | | | [20, 0.98] | 7 | 7 | 0.084 | 1 |
| rand_v100d30 | 6 | HC | rand | 6 | 4.224 | 0.213 | 1 |
| | | | steepest | 5 | 4.247 | 0.371 | **0** |
| | | SA | [5, 0.9] | 6 | 6 | 0.099 | 1 |
| | | | [50, 0.995] | 6 | 6 | 0.082 | 1 |
| | | | [50, 0.98] | 6 | 6 | 0.097 | 1 |
| | | | [5, 0.995] | 6 | 6 | 0.099 | 1 |
| | | | [20, 0.98] | 6 | 6 | 0.098 | 1 |
| rand_v100d50 | 9 | HC | rand | 9 | 6.524 | 0.372 | 1 |
| | | | steepest | 9 | 6.532 | 1.258 | 1 |
| | | SA | [5, 0.9] | 9 | 9 | 0.113 | 1 |
| | | | [50, 0.995] | 9 | 9 | 0.094 | 1 |
| | | | [50, 0.98] | 9 | 9 | 0.107 | 1 |
| | | | [5, 0.995] | 9 | 9 | 0.112 | 1 |
| | | | [20, 0.98] | 9 | 9 | 0.110 | 1 |
| rand_v100d70 | 15 | HC | rand | 15 | 10.523 | 0.665 | 1 |
| | | | steepest | 12 | 9.851 | 2.981 | **0** |
| | | SA | [5, 0.9] | 15 | 14.660 | 0.141 | 1 |
| | | | [50, 0.995] | 15 | 14.640 | 0.115 | 1 |
| | | | [50, 0.98] | 15 | **14.840** | 0.134 | 1 |
| | | | [5, 0.995] | 15 | 14.800 | 0.141 | 1 |
| | | | [20, 0.98] | 15 | 14.760 | 0.139 | 1 |

Table 1

ie. $OM^1 = MV_1^1 = V - AS(1) = CS(S^0) - CS(S^1)$, here we use $S^0$ and $S^1$ to indicate solution in the first and the second time step, $MV_j^i$ is set of missing vertex $j$ in $OM$ of time step $i$. When we add a new vertex $\{2\}$ to $S$ in time step 2, we update $CS(S^2)$ as $CS(S^2) = CS(S^1) - AS(2)$, ie set of vertices that connects to 1 and 2. It is expected that $OM^2$ should consist of two disjoint subset, one is $MV_1^2$, of vertices that only miss the edge to 1 and the other is $MV_2^2$, set of vertices that only miss the edge to 2. The computation is:

- $MV_1^2 = MV_1^1 \cap AS(2)$
- $MV_2^2 = CS(S^1) - CS(S^2)$, or set that connect every other nodes except the one just added (2)

Generally, computing $OM^i$ from $OM^{i-1}$ after adding vertex $v$ to $S^i$ should be:

- $MV_k^{i+1} = MV_k^i \cap AS(v)$ for each vertex $k \in S^i$
- $MV_v^{i+1} = CS(S^i) - CS(S^{i+1})$

With $OM$ we can update $CS(S)$ after a dropping operation. After dropping a vertex with index $k$ from $S^i$, the size of choose set $|CS(S^{i+1})|$ should be non-decreasing as some vertices previous not available become available. The set becomes available is exactly $MV_k^i$. So $CS(S^{i+1}$ should be updated as:

$$CS(S^{i+1}) = CS(S^i) \cup MV_k^i$$

After dropping, $OM$ also need to be updated. The update seems impossible to achieve efficiently right now due to insufficient information.

# References

[1] E. Marchiori, "Genetic, iterated and multistart local search for the maximum clique problem," in *EvoWorkshops*, 2002.