by Patrick
August 7th, 2018

# Combining Cliquer and MCSa

## Introduction

Cliquer [2] is essentially a Russian doll search [7]. Given a graph $G$, Cliquer finds the largest clique in the set of vertices $\{1\}$ and records the size of this clique in $c[1]$. Cliquer then searches for the largest clique in the set of vertices $\{1, 2\}$ and records the size of this in $c[2]$. On the $i^{th}$ iteration Cliquer finds the largest clique in the set of vertices $\{1..i\}$ and records this size in $c[i]$. Within an iteration, if a vertex $v$ is selected from the candidate set and $c[v]$ plus the size of the currently growing clique cannot displace the incumbent search can be abandoned. One hard restriction on Cliquer is that vertices in the candidate set must be visited in decreasing order.

MCSa is essentially one of Tomita's algorithms [6, 4, 5]. At each recursive call, the candidate set is coloured, with each vertex given a colour number. Vertices are then selected in non-increasing colour order (i.e. from one colour class at a time, so as to exhaust one colour class before going on to another). Consequently, if the colour number of a selected vertex $v$ plus the size of the currently growing clique cannot displace the incumbent search can be abandoned.

Therefore we have two algorithms, one computing an exact bound but committed to a static vertex selection strategy, the other performing a colouring step at each recurisve call and dynamically selecting vertices in a coloured order. These two algorithms can be combined, with some compromise. This new algorithm is presented below along with a small empirical study.

## MCSCliquer = Cliquer + MCSa

The trick used to combine these two approaches is to associate with each vertex a colour and to associate with each colour, the number of times that colour has been used. In addition we maintain a count of the number of colours used. For example, if we have a candidate set $P = \{a, b, c, d, e\}$ and these vertices are coloured red, green, blue, red and green respectively then we have used three colours. If vertices $a$ is removed from the candidate set then the number of times red is used in $P$ is now one. If vertex $d$ is later removed from $P$ then red is not used at all, consequently we can reduce the count of the number of colours used, down from three to two.

Algorithm 1 has an outer loop, lines 17 to 19, from a plain vanilla Cliquer algorithm, i.e. at iteration $i$ we find the largest clique in the set of vertices $\{1, \ldots, i\}$ via the call to *expand* in line 18, where the the growing clique is seeded with vertex $i$ and the candidate set is the set of vertices in $\{1, \ldots, i\}$ that are adjacent to vertex $i$ (and we assume that there are no self-loops in the graph). On returning from the call to *expand*, at line 19 we record the size of the largest clique found in the set of vertices $\{1, \ldots, i\}$.

Actual search takes place courtesy of the *expand* function of lines 1 to 12. It takes as arguments the growing clique $C$ and the candidate set $P$. On entering *expand*, if a new largest clique is found we take note of this (line 3, and we might also save off that clique). In line 4 the candidate set is greedily coloured, such that adjacent vertices are placed in different colour classes. Consequently, colour classes are independent sets (composed of non-adjacent vertices) and the size of the clique is bounded from above by the number of colours used. The colouring function returns a triple: the number of colours used, a vector giving the colour assigned to each vertex, a vector giving for each colour the number of times it was used. Vertices are then visited in Cliquer order, i.e. from highest index to lowest index. We now have two bounds that we can use to limit search. The first is a colour bound, line 6, where the number of colours used might not suffice to unseat the incumbent. The next is the Cliquer bound, line 7, where

we test if the largest clique found, in previous iterations of lines 18 and 19, using vertices $\{1,\ldots,v\}$ is insufficient to unseat the incumbent. In lines 8 and 9 we update the colour bound, by decrementing the number of times the colour of the current vertex has been used (lines 8 and 9) and then conditionally decrementing the number of colours used (line 10). A recursive call to *expand* is then made, line 11, with the current vertex $v$ added to the growing clique and a new candidate set made of the vertices in $P$ that are adjacent to $v$ (again, we assume no self-loops). On return from the recursive call, the current vertex $v$ is removed from the candidate set (line 12).

**Algorithm 1:** MCSCliquer

```
1  expand(C, P)
2  begin
3      maxSize ← max(maxSize, |C|)
4      {numberOfColours, colourOfVertex, timesColourUsed} ← colourVertices(P)
5      for v ∈ reverse(P) do
6          if |C| + numberOfColours ≤ maxSize then return
7          if |C| + c[v] ≤ maxSize then return
8          colour ← colourOfVertex[v]
9          timesColourUsed[colour] ← timesColourUsed[colour] − 1
10         if timesColourUsed[colour] = 0 then numberOfColours ← numberOfColours − 1
11         expand(C ∪ {v}, neighbourhood(v) ∩ P)
12         P.remove(v)

13 MCSCliquer(G)
14 begin
15     maxSize ← 0
16     for i ∈ {1,...,n} do c[i] ← 0
17     for i ∈ {1,...,n} do
18         expand({i}, neighbourhood(i) ∩ {1,...,i})
19         c[i] ← maxSize
```

It is worth noting that if we remove line 6, the edited algorithm explores the same search space as Cliquer.

## A small empirical study

We would hope that at least, our new algorithms would be better than Cliquer and maybe not much worse that MCSa, i.e. its performance would place it between Cliquer and MCSa. To investigate this we use a subset of the DIMACS instances [1], affectionately referred to as *the Goldilocks instances*, i.e. those that are not to hard and not too easy [3]. The experiments were run on Intel (R) Xeon(R) CPU E5-2660 @ 2.20GHz processors with 20480 KB of cache, 132 GB of memory, Scientific Linux release 6.10, and all algorithms coded in Java 1.8.051. Since Cliquer dictates that vertices be visited in decreasing order, the adjacency matrix was permuted and vertices renamed such that that verices are visited in non-decreasing degree order. The results of the experiments are shown in Table 1 below. Runtimes were capped at 14400 seconds, i.e. 4 hours. If the algorithm failed to terminate in that time we have a table entry of a dash. What we see is that Cliquer is truly hopeless, whereas MCSCliquer in the same ball park as MCSa.

| instance | Cliquer | seconds | MCSCliquer | seconds | MCSa | seconds |
|---|---|---|---|---|---|---|
| brock200-1 | 116,595,420 | 77.6 | 707,438 | 4.4 | 524,723 | 4.6 |
| brock400-1 | 15,822,180,389 | - | 326,117,237 | 3,442 | 198,359,829 | 3,862 |
| brock400-2 | 15,462,284,410 | - | 203,817,831 | 2,433 | 145,597,994 | 2,930 |
| brock400-3 | 16,131,413,323 | - | 100,020,998 | 996 | 120,230,513 | 2,074 |
| brock400-4 | 15,705,637,222 | - | 78,972,196 | 769 | 54,440,888 | 1,076 |
| brock800-4 | 15,227,709,741 | - | 1,419,549,129 | - | 640,444,536 | - |
| MANN-a27 | 23,985,802,214 | - | 67,735 | 11.6 | 38,019 | 8.4 |
| MANN-a45 | 10,202,369,582 | - | 11,590,734 | - | 2,851,572 | 4,482 |
| p-hat1000-1 | 7,991,346 | 6.9 | 423,903 | 3.2 | 176,576 | 2.7 |
| p-hat1000-2 | 11,208,331,187 | - | 81,114,517 | 2,562 | 34,473,978 | 1,784 |
| p-hat1500-1 | 72,967,408 | 62.5 | 3,188,018 | 25.6 | 1,184,526 | 11.1 |
| p-hat300-3 | 3,720,813,417 | 4,630 | 719,086 | 13.1 | 624,947 | 17.8 |
| p-hat500-2 | 469,855,484 | 653 | 132,003 | 3.1 | 114,009 | 2.7 |
| p-hat500-3 | 12,813,959,841 | - | 68,433,164 | 2,424 | 39,260,458 | 1,959 |
| p-hat700-2 | 11,687,983,900 | - | 1,385,300 | 42.6 | 750,903 | 43.6 |
| san1000 | 45,875,765,421 | - | 713,601 | 5.3 | 150,725 | 11.9 |
| san200-0.9-2 | 16,313,740,281 | - | 759,195 | 10.3 | 229,567 | 4.4 |
| san200-0.9-3 | 14,918,177,622 | - | 34,784,225 | 526 | 6,815,145 | 135 |
| san400-0.7-1 | 30,211,449,997 | - | 2,189,970 | 15.2 | 119,356 | 3.7 |
| san400-0.7-2 | 33,056,676,679 | - | 3,614,133 | 43.7 | 889,125 | 29.9 |
| san400-0.9-1 | 15,681,737,224 | - | 958,746,155 | - | 4,536,723 | 672 |
| sanr200-0.9 | 14,063,255,062 | - | 47,959,328 | 797 | 14,921,850 | 336 |
| sanr400-0.5 | 13,957,283 | 12.3 | 362,527 | 3.2 | 320,110 | 4.3 |
| sanr400-0.7 | 15,705,823,814 | - | 123,148,404 | 1,111 | 64,412,015 | 920 |

Table 1: DIMACS *Goldilocks* instances, calls to *expand* and run time in seconds.

# Conclusion

The new algorithm's performance is interesting in that although it is algorithmically different from MCSa and Cliquer it's performance is quite similar to MCSa. Although MCSa is restricted to visiting vertices in colour order, by removing this restriction and using colour-counting performance appears to be little affected. This might suggest further investigation. Also, there has been some Cliquer-like algorithms coming from the max weight clique community, and this might also be revisited.

# References


[1] DIMACS clique benchmark instances. ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique.

[2] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[3] F. Prefect and P. Prosser. Empirical algorithmics: draw your own conclusions. *CoRR*, abs/1412.3333, 2014.

[4] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique and computational experiments. *Journal of Global Optimization*, 37:95–111, 2007.

[5] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding maximum clique. In *WALCOM 2010, LNCS 5942*, pages 191–203, 2010.

[6] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363:28–42, 2006.

[7] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1.*, pages 181–187, 1996.