University of Glasgow | School of Computing Science

Review of the Bron-Kerbosch algorithm and variations

Alessio Conte

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — May 05, 2013

Abstract

This report describe six different variations of the Bron-Kerbosh algorithm (which solves the problem of listing all maximal cliques in a graph) and compares their performance.

Contents

Chapter 1

Introduction

The Bron-Kerbosch algorithm is used to find all maximal cliques in an undirected graph.

1.1 Implementation

The algorithms were implemented in Java, which is not a fast language since it runs on a virtual machine. On top of that memory consumption wasn't optimized since at each step of the search new data structures are created for each recursive call. This makes these implementation not optimal for usage in a real scenario, though good enough for the purpose of this report, since the focus is on comparing the algorithms rather than speeding up the process. The different implementations are all extension of the same class, which gives the basic structure of the execution. This guarantees that differences in the performance are due to the algorithms themselves rather than differences in the implementation.

Chapter 2

The algorithms

2.1 Classic Bron-Kerbosch

This is a simple implementation of the Bron-Kerbosch algorithm.

Algorithm 1: Classic Bron-Kerbosch algorithm

```
   BronKerbosch($R, P, X$)
1  if $P$ and $X$ are both empty then
2      report $R$ as a maximal clique
3  for each vertex $v$ in $P$ do
4      BronKerbosch($R \cup v, P \cap N(v), X \cap N(v)$)
5      $P \leftarrow P \setminus v$
6      $X \leftarrow X \cup v$
```

On the first call R and X are set to $\emptyset$, and P contains all the vertexes the graph. R is the temporary result, P the set of the possible candidates and X the excluded set. $N(v)$ indicates the neighbors of the vertex v.

The algorithm can be textualized as follwing: Pick a vertex v from P to expand. Add v to R and remove its non-neighbors from P and X. Then pick another vertex from the new P set and repeat the process. Continue until P is empty. Once P is empty, if X is empty then report the content of R as a new maximal clique (if it's not then R contains a subset of an already found clique). Now backtrack to the last vertex picked and restore P,R and X as they were before the choice, remove the vertex from P and add it to X, then expand the next vertex. If there are no more vertexes in P then backtrack to the superior level.

2.2 Bron-Kerbosch with degeneracy ordering

This algorithm is very similar to the classic Bron-Kerbosch. The difference is that on the outermost level of recursion the vertexes are picked in a specific order, called $degeneracyordering$. Given d, degeneracy of the graph a $degeneracyordering$ of the graph is an ordering in which each vertex has d or less neighbors which follow it in the ordering (it doesn't matter how many neighbors precede it). A degeneracy ordering of a graph can be computed in linear time, so computing it doesn't impact the computational comlpexity of the classic Bron-Kerbosh (which is exponential on the number of vertexes in the graph). David Eppstain demonstrated that a modified version of the Bron-Kerbosch that visits the graph using a $degeneracyordering$ can be bound in complexity to $O(d * n * 3^{d/3})$ (exponential on d), instead of the classic Bron-Kerbosch bound ($O(3^{n/3})$, exponential on n). This algorithm however, is not the one created by Eppstein et al.; as said before it is a classic Bron-Kerbosch with a modified outermost level of recusrion. This was made to test how the ordering itself affects the execution of the algorithm.

2.3 Tomita

Algorithm 2: Tomita et al.

```
   **Tomita**$(R, P, X)$
1  **if** *$P$ and $X$ are both empty* **then**
2      report $R$ as a maximal clique
3  choose the pivot vertex $u$ in $P \cup X$ as the vertex with the highest number of neighbors in $P$
4  **for** *each vertex $v$ in $P \setminus N(u)$* **do**
5      Tomita$(R \cup v, P \cap N(v), X \cap N(v))$
6      $P \leftarrow P \setminus v$
7      $X \leftarrow X \cup v$
```

Tomita is a modified version of the Bron-Kerbosch algorithm developed by Tomita et al. It uses a specific pivoting policy to cut computational branches. The pivoting consists in the following: instead of iterating at each expansion on the P set, chose a pivot. The results will have to contain either the pivot or one of its non-neighbors, since if none of the non-neighbors of the pivot is included, then we can add the pivot itself to the result. Hence we can avoid iterating on the neighbors of the pivot at this step (they will still be expanded in the inner levels of recursion). The strategy proposed by Tomita et al. is to chose the pivot as the node in $P \cup X$ with the highest number of neighbors in P.

2.4 Tomita with degeneracy ordering

Tomita's version of the algorithm, with the outermost level of recursion ordered in a $degeneracyordering$. A combination of $Bron - Kerbosch$ with degeneracy ordering and $Tomita$.

2.5 Tomita on vertex cover

Tomita's version of the algorithm, with the first iteration only performed on a subset of the P set. The subset is a vertex cover of the graph, which means that for each edge in the graph, at least one of the vertices is in the cover. It is demonstrated in the appendices that this algorithm returns the same set (not necessarily in the same order) of cliques returned by the classic Bron-Kerbosch algorithm.

Chapter 3

Performance

The algorithms were tested on a set of randomly generated graphs, generated with two parameters: n (number of vertexes) and p (chance of connection between nodes). Graphs of different dimension and denseness were generated to compare the performances in different situations. Due to the not optimized nature of the implementations the algorithms were able to solve graphs with parameters up to n = 1000 and p = 30$\%$. The search on graphs with higher parameters didn't terminate on neither of the implememtations, and was manually interrupted after 20 to 30 minutes. At the end of the chapter a table shows the results got from each algorithm on each graph of the testing set.

3.1 Classic Bron-Kerbosch

The basic algorithm, as expected, didn't perform as well as the extensions (except for $Bron - Kerbosch$ with degeneracy ordering) in terms of both execution time and nodes expanded.

3.2 Bron-Kerbosch with degeneracy ordering

This version of the algorithm didn't gain any advantage over the basic one. It expanded exactly the same number of nodes as $ClassicBron - Kerbosch$, and run in a slightly bigger time (due to the additional computation of the degeneracy ordering).

3.3 Tomita

$Tomita$ gave overall the best performance. It outperformed all the other algorithms on all of the graphs in terms of computational time, and was the second-best for number of expanded nodes. Compared to $ClassicBron - Kerbosch$, the nodes expanded were reduced by a factor varying from approx. 2 to approx. 2000. The time was always smalles, sometimes very similar and sometimes by an approx. 2 factor. The only algorithm that expanded less nodes than $Tomita$ was $Tomita$ with degeneracy ordering.

3.4 Tomita with degeneracy ordering

Though the use of a degeneracy ordering didn't change anything when applied to $ClassicBron - Kerbosch$, it did make a difference when combined with $Tomita$. This algorithm expanded the smallest amount of nodes overall, though the number was always very close to the one made by $Tomita$ (the reduction went up to a factor of just 1.05). Though it expanded less nodes, the execution time was always slightly bigger compared to $Tomita$.

3.5 Tomita on vertex cover

As demonstrated in the appendices, this version of the algorithm were able to produce a complete result by starting the first iteration on a sub-set of the vertexes of the graph (instead of the full graph). The purpose was to expand a smaller amount of nodes. Surprisingly, though, the total amount was always slightly higher compared to the simple $Tomita$ (as well as the running time as well). If on one hand, the vertex cover was almost as big as the graph (95% to 99% of the vertexes), when another algorithm was implemented to calculate a smaller cover (85% to 98% of the vertexes), the number of nodes expanded increased even more. Hence this solution, while accurate, didn't prove to gain any advantage on neither sparse or dense graphs in the testing set. A performance test on a minimal vertex cover was not made, but it would prove to be useless in terms of speedup, since computing a minimal vertex cover is NP-complete and would add a heavy load on the computational time.

Alg	10-70-00	30-90-00	50-70-00	100-20-00	100-60-00	1000-20-00	1000-30-00
BK	cliques: 10 Max size: 5 Nodes: 108 Cpu Time: 1	cliques: 266 Max size: 18 Nodes: 1640624 Cpu Time: 972	cliques: 4879 Max size: 11 Nodes: 215701 Cpu Time: 78	cliques: 879 Max size: 5 Nodes: 2484 Cpu Time: 1	cliques: 57064 Max size: 12 Nodes: 1419083 Cpu Time: 563	cliques: 1218685 Max size: 8 Nodes: 5105221 Cpu Time: 2128	cliques: 15299046 Max size: 9 Nodes: 103448254 Cpu Time: 47221
BK Degen	cliques: 10 Max size: 5 Nodes: 108 Cpu Time: 2	cliques: 266 Max size: 18 Nodes: 1640624 Cpu Time: 964	cliques: 4879 Max size: 11 Nodes: 215701 Cpu Time: 86	cliques: 879 Max size: 5 Nodes: 2484 Cpu Time: 8	cliques: 57064 Max size: 12 Nodes: 1419083 Cpu Time: 580	cliques: 1218685 Max size: 8 Nodes: 5105221 Cpu Time: 2410	cliques: 15299046 Max size: 9 Nodes: 103448254 Cpu Time: 48435
Tomita	cliques: 10 Max size: 5 Nodes: 20 Cpu Time: 1	cliques: 266 Max size: 18 Nodes: 731 Cpu Time: 13	cliques: 4879 Max size: 11 Nodes: 11549 Cpu Time: 57	cliques: 879 Max size: 5 Nodes: 1658 Cpu Time: 1	cliques: 57064 Max size: 12 Nodes: 141818 Cpu Time: 91	cliques: 1218685 Max size: 8 Nodes: 2841410 Cpu Time: 1930	cliques: 15299046 Max size: 9 Nodes: 40176790 Cpu Time: 26843
Tomita Degen	cliques: 10 Max size: 5 Nodes: 20 Cpu Time: 1	cliques: 266 Max size: 18 Nodes: 699 Cpu Time: 15	cliques: 4879 Max size: 11 Nodes: 11773 Cpu Time: 56	cliques: 879 Max size: 5 Nodes: 1652 Cpu Time: 12	cliques: 57064 Max size: 12 Nodes: 141526 Cpu Time: 101	cliques: 1218685 Max size: 8 Nodes: 2815789 Cpu Time: 2054	cliques: 15299046 Max size: 9 Nodes: 39796407 Cpu Time: 26967
Tomita Cover	cliques: 10 Max size: 5 Nodes: 26 Cpu Time: 1 Cov. size: 8	cliques: 266 Max size: 18 Nodes: 840 Cpu Time: 18 Cov. size: 29	cliques: 4879 Max size: 11 Nodes: 12321 Cpu Time: 36 Cov. size: 48	cliques: 879 Max size: 5 Nodes: 1718 Cpu Time: 3 Cov. size: 95	cliques: 57064 Max size: 12 Nodes: 145363 Cpu Time: 103 Cov. size: 98	cliques: 1218685 Max size: 8 Nodes: 2851815 Cpu Time: 2149 Cov. size: 995	cliques: 15299046 Max size: 9 Nodes: 40369606 Cpu Time: 29718 Cov. size: 996