Fast maximal cliques enumeration by locally optimal vertex ordering

2022-03-02

Contents

1 Algorithm

1.1 Preliminaries

1. A simple undirected graph is denoted as $G(V, E)$.
2. For a vertex $v \in V$, $\Gamma(v)$ denotes all vertices adjacent to v in G. Note that $v \notin \Gamma(v)$.

1.2 Algorithm design

The algorithm described here is used for fast maximal cliques enumeration (MCE) of undirected graphs. Most optimizations are attributed to "locally optimal vertex ordering", so we named the new algorithm as "`LOVO`".

`LOVO`, inspired by Bron-Kerbosch's algorithm, operated three disjoint sets of vertices during the search for maximal cliques:

X: $x \in X$ are searched vertices. Maximal cliques that contain $\forall x \in X$ have been enumerated.

R: $\forall r \in R$ links to each other.

P: $\forall p \in P$ links to all vertices in R, but $p \in P$ not necessarily links to each other. When vertex p_1 deleted in P, the set $\{p \in P | p \neq p_1\}$ is denoted as $\{P \setminus p_1\}$.

Status s is defined as a combination of X, R, and P.

$$s : \{x \in X\}, \{r \in R\}, \{p \in P\}$$

1.2.1 Raw Bron-Kerbosch's algorithm

Raw Bron-Kerbosch's algorithm could be expressed as:

- Step1: randomly select a vertex p_1 in P, then generate a new status s_{new}:

$$s_{new} : \{x \in X\}, \{r \in R, p_1\}, \{\Gamma(p_1) \cap P\} \quad (1)$$

- Step2: iterate `Step1` until $P = \emptyset$:

$$s_{stop} : \{x \in X\}, \{r \in R, p_1, \ldots, p_n\}, \{\}$$

Check if R, a clique, is a subset of any $\Gamma(x | x \in X)$. If not, then R is a new maximal clique.

- Step3: backtrack to a updated status s:

$$s_{updated} : \{x \in X, p_1\}, \{r \in R\}, \{P \setminus p_1\} \quad (2)$$

It means for status s, p_1 has been completely searched. Then, repeat `Step1` and `Step2` until all vertices in P are searched, in another words, $P = \emptyset$, so we get all maximal cliques from status s.

The initial status of $G(V, E)$ could be set as:

$$s_{init} : \{\}, \{\}, \{v \in V\}$$

1.2.2 LOVO

`LOVO` is designed based on the fact that: $\forall p \in P$ are searched in status s, and cliques in s_{stop} is checked by $\Gamma(x)$, so for a given $x_n \in X$ if P containing only $P \cap \Gamma(x_n)$ can be skipped when $R \subset \Gamma(x_n)$. $R \subset \Gamma(x_n)$ can be meet with high possibility, because X is kept growing in `Step3`. Moreover, in a dense graph $\Gamma(x_n)$ covers most vertices that make the restriction above to be meet more easily.

The skippable status is described as:

$$s_{skip} : \{x \in X\}, \{r \in (R|R \subset \Gamma(x_n))\}, \{P \cap \Gamma(x_n|x_n \in X)\}$$

In `LOVO`, x_n is chosen to maximize $|N(x_n)|$, which is equal to the number of skipped iterations:

$$N(x_n) = (R \cup P) \cap \Gamma(x_n)$$

The `LOVO` is described as:

- step1: select x_n to maximize $|N(x_n)|$,

Case1: $R \not\subset \Gamma(x_n)$, randomly select p_1 from P.

Case2: $R \subset \Gamma(x_n)$ and $P \setminus N(x_n) \neq \emptyset$, randomly select p_1 from $P \setminus N(x_n)$. In this way, $p \in P \setminus N(x_n)$ are always searched in last rank.

Case3: $R \subset \Gamma(x_n)$ and $P \setminus N(x_n) = \emptyset$, skip current status, then backtrack to the second latest status.

The status s_{new} has the same format as **1**. The status s is updated as **2**.

- step2: Use the same vertex selection and ordering method for s_{new}. Note that x'_n in s_{new} may be different from the x_n in s. It means x_n is always chosen according to the current status s and make the $|N(x_n)|$ local optimal. Repeat `step1` until $P = \emptyset$, then check whether current R is a maximal clique.
- step3: backtrack to the second latest status, and repeat `step2` until no status left.

A search tree of status s looks like:

```
{s}
{s_u, s_n} ## update s to s_u, generate s_n from s.
{s_u, s_u1, s_n1} ## update s_n to s_u1, generate s_n1 from s_n.
{s_u, s_u1, s_u2, s_n2}
{s_u, s_u1, s_u2, s_u3, s_n3} ## P in s_n3 is empty, check whether R in s_n is maximal clique.
{s_u, s_u1, s_u2, s_u3', s_n3'} ## update s_u3 to s_n3', generate s_n3' from s_u3.
{s_u, s_u1, s_u2, s_n3'} ## s_n3' is skipped because case3 above occurs.
...
{} ## Stop when no status left.
```

1.3 Optimization

1.3.1 Check before search

Different from Bron-Kerbosch's algorithm, `LOVO` "ordered" vertices in P that roughly split P into two groups $N(x_n)$ and $P \setminus N(x_n)$. Vertex p for next search was randomly selected from $P \setminus N(x_n)$.

`LOVO` always checked the union of R and P is part of searched maximal cliques, so invalid iterations could be avoid. *Case3* in `step1` above described the condition, $R \subset \Gamma(x_n)$ and $P \setminus N(x_n) = \emptyset$ that equals to $(R \cup P) \subset \Gamma(x_n | x_n \in X)$. Besides, maximized $|N(x_n)|$ guaranteed that largest possible searched maximal cliques or parts were checked in high priority.

It is possible that a chosen vertex p_x from $P \setminus N(x_n)$ actually formed a searched maximal cliques. p_x is omitted just because its corresponding $N(x_m)$ is not maximal. Any p_x like below should not be searched:

$$\{\Gamma(p_x) \cap P \cup R, p_x\} \subset \Gamma(x_m | x_m \in X)$$

`LOVO` only did one more search for p_x. Because when p_x was selected, new status would be:

$$s_{new} : \{x \in X\}, \{r \in R, p_x\}, \{\Gamma(p_x) \cap P\}$$

Clearly, s_{new} would be skipped before next search.

1.3.2 Quick skip when $P = \emptyset$

When $P = \emptyset$ in status s, a clique R appeared. A continuously changed P and R may occur in continuous status. The multiple status could be skipped at once. For example:

```
{s_u, s_u1, s_u2, s_u3, s_n3}
s_n3: R is {R, p1, p2, p3}, P is empty
s_u3: R is {R, p1, p2}, P is {p3}
s_u2: R is {R, p1}, P is {p2, p3}
s_u1: R is {R}, P is {p1, p2, p3}
```

Status $\{ s_x | x \in \{last-1, last-2, last-3, ...\}\}$ were immediately skipped in `LOVO`, when:

$$|R_x| + |P_x| < |R_{last}|$$
$$P_{last} = \emptyset$$

1.3.3 `bitset` transformation of linked vertices

Compared to Bron-Kerbosch's algorithm, `LOVO` added the step $N(x_n)$ that handled huge amount of `set intersection` and `set union` operation. Instead of holding a integer vector for vertex indices, `LOVO` implemented a `bitset` for edge representation. A similar strategy called `bitscan` is used.

For example, using integer vector a triangle graph is represented as:

```
0: 1, 2
1: 0, 2
2: 0, 1
```

In `LOVO`, the graph is transformed as:

```
110
101
011
```

`LOVO`, even the operation $\Gamma(p_1) \cap P$ in Bron-Kerbosch's algorithm, benefited greatly from bit `and` and `or` with hardware optimization in different languages. Moreover, `bitset` used much less memory compared to integer vector. For the example above, integer vector needed at least $N \times 6$ bits where N is usually 16 or 32, while `bitset` used 9 bits.

2 Unsolved problems

2.1 Vertices ordering in P

Strategy 1: choose $v \in P$ that maximizes $\Gamma(v) \cap P$, then v will be move to X in the next status s. The search number will be reduced.

2.2 Maximize $|(R \cup P) \cap \Gamma(x_n)|$ or $|P \cap \Gamma(x_n)|$?

3 References