社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－MPS－67 (8)
2007－BIO－11 (8)
2007／12／20

NetMCQ : A Distributed Exact Maximum Clique Solver

SHOHEI URABE[†1] and ETSUJI TOMITA[†1]

Abstract. We present NetMCQ, a distributed, exact and efficient branch-and bound algorithm for finding a maximum clique in an arbitrary graph based on the algorithm MCQ. Computational experiments are carried out to demonstrate its efficacy.

1. Introduction

A *clique* is an undirected graph whose vertices are pairwise adjacent. *The Maximum Clique Problem* on an undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ asks for a maximum subgraph of $\boldsymbol{G}$ which itself is a clique. Despite of its NP-Hardness, this problem has a variety of applications[1]. Efficient algorithms to solve it gains significant impact on both theoretical and practical fields, thus it has been studied by many researchers.

In this note we present a distributed algorithm to exactly solve the maximum clique problem, called NetMCQ. This algorithm is a distributed application for our preceding algorithm MCQ[2] and has quite similar characteristics.

2. Basic Algorithm

Let us consider an undirected graph $\boldsymbol{G}$ and a procedure EXPAND. EXPAND calculates the size of a maximum clique in a particular subgraph. Using EXPAND the maximum of the whole given graph $\boldsymbol{G}$ can be computed as EXPAND($\boldsymbol{G}, \boldsymbol{V}, 0$).

2.1 EXPAND

EXPAND is a recursive procedure which satisfies the following characteristics.

Input $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, a subset of its vertices $\boldsymbol{R} \subseteq \boldsymbol{V}$, and an integer q.

Output Sum of q and the size of the maximum clique in $\boldsymbol{R}$-induced subgraph under $\boldsymbol{G}$.

Our algorithm implicitly maintains a subset of $\boldsymbol{G}$, which is a clique, and explicitly $\boldsymbol{R}$, the candidate of vertices to add to the target clique. We select a certain vertex p from $\boldsymbol{R}$ as a clique component, then compute $\boldsymbol{R_p} = \boldsymbol{R} \cap \boldsymbol{\Gamma}(p)$ and $\boldsymbol{R_x} = \boldsymbol{R} - \{p\}$ as the new candidate sets of the vertices, where $\boldsymbol{\Gamma}(p) = \{q \in \boldsymbol{V} \mid (p, q) \in \boldsymbol{E}\}$ is the *neighbor* of p. We apply recursively EXPAND to both $\boldsymbol{R_p}$ and $\boldsymbol{R_x}$ and return whichever grater as the answer of this procedure. When $\boldsymbol{R} = \emptyset$ is reached, q is the size of the implicitly maintained maximal clique, so we take this as the return value.

```
procedure EXPAND(G, R, q)
begin
    if q > |Q_max|
    then
        update Q_max;
        broadcast q
    fi;
    if R ≠ ∅
    then
        NUMBER_SORT(R);
        p := max R;
        if q + No(p) > |Q_max|
        then
            R_p := R ∩ Γ(p);
            R_x := R − {p};
            concurrently execute following:
            EXPAND(G, R_p, q + 1);
            EXPAND(G, R_x, q)
        fi
    fi
end.
```

Fig. 1 Procedure EXPAND

3. Pruning

In order to avoid unnecessary searching, MCQ equips approximate coloring algorithm NUMBER_SORT[2] to prune the search tree. With this a positive integer or *color* No(p) is assigned for each $p \in \boldsymbol{R}$, and $\boldsymbol{R}$ gets ordered along with

†1 Department of Information and Communication Engineering, The University of Electro-Communications, Tokyo 182-8585, Japan.
E-mail:{shyouhei,tomita}@ice.uec.ac.jp

that number. Assume we maintain global variable $\boldsymbol{Q_{max}}$ to hold the largest clique ever found on the graph, following condition

$$q + \max\{\mathrm{No}(p) \mid p \in \boldsymbol{R}\} \leq |\boldsymbol{Q_{max}}| \quad (1)$$

indicates that this $\boldsymbol{R}$ does not include any cliques larger than $\boldsymbol{Q_{max}}$. Such $\boldsymbol{R}$ can be disregarded.

When selecting a vertex p to compute $\boldsymbol{R_p}$ and $\boldsymbol{R_x}$, we chose a vertex such that the color of that vertex is maximum in the subgraph $\boldsymbol{R}$. As $\boldsymbol{R}$ is ordered, doing so is a simple subtraction of the largest entry of $\boldsymbol{R}$, which is typically O(1).

4. Concurrent Execution

In the basic algorithm above, every call of EXPAND procedure can run concurrently. We can distribute those processes over the network. Simple approach was taken here:

(1) Every network node has its state, namely "free" and "busy". Free state indicates that node is not currently assigned to a specific subproblem, while busy indicates the node is currently executing EXPAND procedure on some subproblem.

(2) If there is a network node which is in free state, then send subproblem $\boldsymbol{R_x}$ (in the list above) to that node, and the original node behaves as if $\boldsymbol{R_x} = \emptyset$ (i.e. ignore sent subproblem). The node which was given a subproblem then set its state to busy state, and execute EXPAND procedure over the given subproblem.

(3) Otherwise EXPAND over $\boldsymbol{R_x}$ is executed in the same network node as the EXPAND-ing node for $\boldsymbol{R_p}$.

(4) When a node in busy state reaches to its end of execution by $\boldsymbol{R} = \emptyset$, It broadcasts its answer (i.e. the clique size found so far), then turning itself into free state, waits for another network nodes to send next subproblem.

4.1 Criteria

Above approach works properly. But in our earlier experiments, running distributed algorithm with the schema turned out to be much slower than MCQ. We observed that a network transfer between two nodes is much expensive compared to an in-node execution of EXPAND. So some heuristics are taken.

We will not distribute a subproblem unless *all* of the following conditions are met:

(1) Let $\boldsymbol{R}$ be a subproblem we are considering, $\rho(\boldsymbol{G})$ be the edge density of $\boldsymbol{G}$, and $\boldsymbol{Q_{max}}$ be a maximum clique found so far. Then, the following condition

$$|\boldsymbol{R}| + q \geq \frac{|\boldsymbol{Q_{max}}|}{\rho(\boldsymbol{G})^3} \quad (2)$$

must be met to distribute $\boldsymbol{R}$.

(2) Let $\boldsymbol{R}$ be a subproblem we are considering. Then, the following condition

$$|\boldsymbol{R}| \geq |\boldsymbol{Q_{max}}| + 100 \quad (3)$$

must be met to distribute $\boldsymbol{R}$.

As the search progress, given a vertex $p \in \boldsymbol{R_p}$, the size of $\boldsymbol{R_p} = \boldsymbol{R} \cap \boldsymbol{\Gamma}(p)$ is roughly expected to be somewhat proportional to the product of the edge density of the graph and the size of $\boldsymbol{R}$.

$$|\boldsymbol{R_p}| \fallingdotseq \rho(\boldsymbol{G})\,|\boldsymbol{R}| \quad (4)$$

So the first condition roughly means that the EXPAND invocation we are considering is expected to recur more than 3 times. This criteria is expected to prevent distributing of small subproblems which appear near the leaf of search tree.

Second condition is simply expected to prevent distributing of too small subproblem.

We call the basic algorithm, pruning algorithm NUMBER_SORT, and the bounding criteria above together a distributed NetMCQ algorithm.

5. Computational Experiments

We implemented the algorithm NetMCQ in the C language and carried out computational experiments. Though it is not trivial to say when an execution of a NetMCQ ended, because a node is in state free does not mean that an exactly maximum clique was found. To ensure the exact answer we have to wait all the engaged nodes to be in the state free. When all nodes are in free state, no EXPAND instances are running, so no cliques can be found any more.

Our environment is Intel Pentium 4 processor 3.06 GHz, with 1024 MiB main memory, Linux Kernel 2.4.31 with GNU libc 2.3.3, GNU C Compiler version 4.1.0. We used 6 computers with all exactly the same CPU, Memory, OS and Compiler.

5.1 Results

Table 1 shows the result for random graphs of n vertices with edge density ρ. The size of the maximum clique is ω and we got cor-

Table 1 Execution time [sec] for random graphs

n	ρ	ω	1 node time[sec]	6 nodes time [sec]	speedup ratio
200	0.7997	25	10.66	9.43	1.13
200	0.9022	41	909.03	221.32	4.11
200	0.9534	63	1,563.99	230.28	6.86
300	0.7006	20	20.86	18.41	1.13
300	0.8001	29	1,393.37	318.10	4.38
400	0.7010	22	309.08	140.04	2.20
400	0.8004	30	74,527.38	15,974.63	4.66
500	0.5996	17	76.06	50.22	1.51
500	0.6994	23	3,166.92	1,066.39	2.97
600	0.5989	18	256.12	202.80	1.26
600	0.6990	23	21,747.93	10,095.03	2.15
700	0.5995	18	938.56	768.62	1.22
800	0.5996	18	3,697.59	1,974.13	1.87
900	0.4993	15	233.57	171.68	1.36
900	0.5998	19	8,389.63	5,490.67	1.53
1,000	0.4999	15	503.55	389.73	1.29

rect results for all instances. In this table instances of higher densities are faster than those of lower ones, indicating the distribution criteria are more effective for dense graphs. Especially in the instance of n=200, ρ=0.9534 the speedup ratio is over 6.0. This happens when the largest clique was found by one of the running nodes while other nodes are finding much smaller cliques. In this case the found largest clique size is broadcasted to all nodes and that value is used for branch pruning as described above, resulting a huge speedup compared to the serial execution.

Table 2 shows the result for benchmark graphs provided in DIMACS[3] challenges. Other than this table, we observed that NetMCQ is slower than MCQ for the instances of roughly $\leq$ 100 sec execution time. For those graphs distribution overheads are the dominating part of the execution. As the 6-node execution is still fast enough, we are not nervous for those results. Also for the instances of MANN series, which are of very high density and takes a long time, NetMCQ is slower than MCQ. This is because all the subproblems distributed from the node initially started execution are large enough to satisfy the criteria, but at the same time they are small enough to be pruned by the coloring. All the distributed subproblems are pruned hereon thus distribution overheads take effect while no branches are executed in parallel.

NetMCQ is faster on other instances shown in Table 2. Results above indicates NetMCQis effective enough.

6. Conclusion

We present NetMCQ, a distributed dialect of algorithm MCQ, and carried out computational experiments on it. This algorithm has quite similar characteristics from MCQ, that is, if MCQ runs fast for a problem NetMCQ also runs fast on it. Moreover it runs faster where MCQ took long time to solve. The distribution schema shown above is rather general and can be applied to other algorithms, such as MCR[2] and others[4]

However it is obvious from computer experiments that NetMCQ has some instability for the input graphs. We assume this is because our distribution criteria do not always fit to the actual branch-and-bound situation. A more strict estimation of branch count should result in a more effective criterion.

References

1) Bomze, I.M., Budinich, M., Pardalos, P.M. and Pelillo, M.: The maximum clique problem, *Handbook of Combinatorial Optimization* (Du, D.Z. and Pardalos, P.M., eds.), Vol.4, Kluwer Academic Publishers, Boston, MA (1999).

2) Kameda, T. and Tomita, E.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, *Journal of Global Optimization*, Vol.37, pp.95–111 (2007).

3) Johnson, D.S. and Trick, M.A.(eds.): *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, the American Mathematical Society (1996).

Table 2 Execution time [sec] for DIMACS benchmark graphs

instance	n	ρ	ω	1 node time[sec]	6 nodes time [sec]	speedup ratio
brock400_1	400	0.7497	27	1,715.23	751.92	2.28
brock400_2	400	0.7492	29	700.64	359.18	1.95
brock400_3	400	0.7478	31	1,516.08	491.59	3.08
brock400_4	400	0.7489	33	675.29	279.39	2.42
MANN_a45	1,035	0.9964	345	4,596.47	6,338.47	0.725
p_hat300-3	300	0.7469	36	16.78	16.32	1.03
p_hat500-3	500	0.7519	50	3,000.39	1,855.10	1.61
p_hat700-2	700	0.4975	44	50.71	45.39	1.12
p_hat1000-2	1,000	0.4901	46	2,874.45	2,528.87	1.14
sanr200_0.9	200	0.8976	42	351.19	76.51	4.59
sanr400_0.7	400	0.7001	21	373.79	164.74	2.27

4) Sutani, Y., Higashi, T., Tomita, E., Takahashi, S. and Nakatani, H.: A faster branch-and-bound algorithm for finding a maximum clique, *Technical Report of IPSJ SIG*, No.2006–AL–108, pp.79–86 (2006).