

# An Exact Branch and Bound Algorithm with Symmetry Breaking for the Maximum Balanced Induced Biclique Problem

Ciaran McCreesh and Patrick Prosser

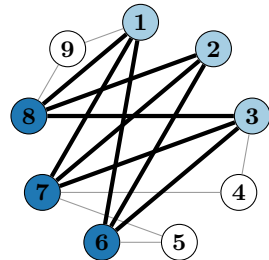
University of Glasgow, Glasgow, Scotland  
c.mccreesh.1@research.gla.ac.uk,  
patrick.prosser@glasgow.ac.uk

**Abstract.** We show how techniques from state-of-the-art branch and bound algorithms for the maximum clique problem can be adapted to solve the maximum balanced induced biclique problem. We introduce a simple and effective symmetry breaking technique. Finally, we discuss one particular class of graphs where the algorithm's bound is ineffective, and show how to detect this situation and fall back to a simpler but faster algorithm. Computational results on a series of standard benchmark problems are included.

## 1 Introduction

Let  $G = (V, E)$  be a graph (by which we always mean finite, undirected and with no loops) with vertex set  $V$  and edge set  $E$ . A *biclique*, or complete bipartite subgraph, is a pair of (possibly empty) disjoint subsets of vertices  $\{A, B\}$  such that  $\{a, b\} \in E$  for every  $a \in A$  and  $b \in B$ . A biclique is *balanced* if  $|A| = |B|$ , and *induced* if no two vertices in  $A$  are adjacent and no two vertices in  $B$  are adjacent. The maximum balanced induced biclique problem is to find a balanced induced biclique of maximum size in an arbitrary graph. We illustrate an example in Fig. 1.

Finding such a maximum is NP-hard [1, Problem GT24], both in bipartite and arbitrary graphs. A naïve exponential algorithm could simply enumerate every possible solution to find a maximum. Here we develop a branch and bound algorithm with symmetry breaking that substantially reduces the search space. We believe that this is the first attempt at tackling this problem. We are not yet aware of any practical applications, but the problem is interesting from an algorithmic perspective.



**Fig. 1.** A graph, with its unique maximum balanced induced biclique of size six,  $\{\{1, 2, 3\}, \{6, 7, 8\}\}$ , shown in light and dark blue

If  $G = (V, E)$  is a graph, we write  $V(G)$  for the vertex set  $V$ . The *neighbourhood* of a vertex  $v$  in a graph  $G$  is the set of vertices adjacent to  $v$ ; we denote this  $N_G(v)$ . The *degree* of a vertex is the cardinality of its neighbourhood.

A graph  $G' = (V', E')$  is a *subgraph* of  $G = (V, E)$  if  $V' \subseteq V$  and  $E' \subseteq E$ ; the subgraph *induced by*  $V'$  is the subgraph with vertex set  $V'$  and all possible edges. A set of vertices, no two of which are adjacent, is called an *independent set*. A set of vertices, all of which are adjacent, is called a *clique*; the size of a maximum clique is denoted  $\omega$ . A *clique cover* is a partition of the vertices in a graph into sets, each of which is a clique. We introduce the symbol  $\tilde{\omega}$  for the size (i.e.  $|A| + |B|$ ) of a maximum balanced induced biclique, which is always even (this simplifies comparisons with unbalanced biclique variants). A graph is *bipartite* if its vertices may be partitioned into two disjoint independent sets.

## 2 A Branch and Bound Algorithm

A very simple branch and bound algorithm for the maximum induced biclique problem is given in Algorithm 1. The algorithm works by recursively building up two sets  $A$  and  $B$  such that  $\{A, B\}$  is a biclique. At each stage,  $P_a$  contains those vertices which may be added to  $A$  whilst keeping a feasible solution (i.e. each  $v \in P_a$  is individually adjacent to every  $b \in P_b$  and nonadjacent to every  $a \in A$ ), and similarly  $P_b$  contains vertices which may be added to  $B$ . Initially,  $A$  and  $B$  are both empty, and  $P_a$  and  $P_b$  both contain every vertex in the graph (line 4).

At each recursive call to **expand**, a vertex  $v$  is chosen from  $P_a$  (line 8) and moved to be in  $A$  instead (lines 10 and 11). The algorithm then considers the implications of  $v \in A$  (lines 12 to 17). A new  $P'_a$  is constructed on line 12 by filtering from  $P_a$  those vertices adjacent to  $v$  (since  $A$  must be an independent set), and a new  $P'_b$  is constructed on line 13 by filtering from  $P_b$  those vertices *not* adjacent to  $v$  (everything in  $B$  must be adjacent to everything in  $A$ ).

Now if  $P'_b$  is not empty, we may grow  $B$  further. Thus we repeat the process with a recursive call on line 17, swapping the roles of  $A$  and  $B$ —we are adding vertices to the two sides of the growing biclique in alternating order.

Having considered the possibility of  $v \in A$ , we then consider  $v \notin A$  (line 18). The algorithm loops back to line 8, selecting a new  $v$  from  $P_a$ , until  $P_a$  is empty. Finally, we backtrack by returning from the recursive call.

We keep track of the largest feasible solution  $\{A_{max}, B_{max}\}$  that we have found so far; this is called the *incumbent*. Initially it is empty (line 3). Whenever we find a potential solution, we compare it to the incumbent (line 14), and if our new solution is larger then the incumbent is unseated (line 15). Note that at this point, the balance condition must be checked explicitly, since either  $|A| = |B|$ , or  $|A| = |B| + 1$  could be true.

Knowing the size of the incumbent allows us to avoid exploring some of the search space—this is the bound part of branch and bound. The condition on line 9 checks how much further we can grow  $A$  and  $B$ : if there are not enough vertices available to potentially unseat the incumbent, search at the current position can be abandoned. (This is not a very good bound, and is only for illustrative purposes. We discuss a more sophisticated bound below.)

---

**Algorithm 1.** A simple, alternating branch and bound algorithm for the maximum balanced induced biclique problem.

---

```

1 simpleBiclique :: (Graph  $G$ )  $\rightarrow$  (Set of Integer, Set of Integer)
2 begin
3    $(A_{max}, B_{max}) \leftarrow (\emptyset, \emptyset)$  // Initially our best solution is empty
4   expand( $G, \emptyset, \emptyset, V(G), V(G), A_{max}, B_{max}$ )
5   return  $(A_{max}, B_{max})$ 

6 expand :: (Graph  $G$ , Set  $A$ , Set  $B$ , Set  $P_a$ , Set  $P_b$ , Set  $A_{max}$ , Set  $B_{max}$ )
7 begin
8   for  $v \in P_a$  do
9     if  $|P_a| + |A| > |A_{max}|$  and  $|P_b| + |B| > |B_{max}|$  then
10       $A \leftarrow A \cup \{v\}$  // Consider  $v \in A$ 
11       $P_a \leftarrow P_a \setminus \{v\}$ 
12       $P'_a \leftarrow P_a \cap \overline{N_G(v)}$  // Remove vertices adjacent to  $v$ 
13       $P'_b \leftarrow P_b \cap N_G(v)$  // Remove vertices not adjacent to  $v$ 
14      if  $|A| = |B|$  and  $|A| > |A_{max}|$  then
15         $(A_{max}, B_{max}) \leftarrow (A, B)$  // We've found a better solution
16      if  $P'_b \neq \emptyset$  then
17        expand( $G, B, A, P'_b, P'_a, B_{max}, A_{max}$ ) // Swap and recurse
18       $A \leftarrow A \setminus \{v\}$  // Now consider  $v \notin A$ 

```

---

*Improving the Algorithm.* We now adapt Algorithm 1 to incorporate symmetry breaking, an improved bound based upon clique covers, and an initial sort order. The end result is Algorithm 2. We have explicitly designed the algorithm to permit a bitset encoding for the data structures. For the maximum clique problem, this technique has allowed an increase in performance of between two and twenty times, without altering the steps taken by the algorithm. We refer to work by San Segundo et al. [2,3] for implementation details.

*Symmetry Breaking.* The search space for Algorithm 1 is larger than it should be: it explores legal *ordered* pairs  $(A, B)$  of vertex sets rather than unordered pairs  $\{A, B\}$ . Having explored every possible solution with  $v \in A$ , the search then considers  $v \notin A$ . But there is nothing to stop it from then considering a new  $v' \in A$ , and later placing  $v \in B$ . This is wasted effort, since if such a solution existed we would already have considered an equivalent with  $A$  and  $B$  reversed.

We may break this symmetry as follows: if, at the top of search, we have considered every possibility with  $v \in A$  then we may eliminate  $v$  from  $P_b$  to avoid considering  $v \in B$ . The modified **expand** function in Algorithm 2 includes this rule: lines 38 to 39 remove symmetric solutions.

This technique may be seen as a special case of the standard lex symmetry breaking technique used in constraint programming [4,5]. A constraint programmer would view  $A$  and  $B$  as binary strings, and impose the constraint  $B \leq A$

---

**Algorithm 2.** An improved alternating branch and bound algorithm for the maximum balanced induced biclique problem.

---

```

1 improvedBiclique :: (Graph  $G$ ) → (Set of Integer, Set of Integer)
2 begin
3    $(A_{max}, B_{max}) \leftarrow (\emptyset, \emptyset)$  // Initially our best solution is empty
4   permute  $G$  so that the vertices are in non-increasing degree order
5   expand( $G, \emptyset, \emptyset, V(G), V(G), A_{max}, B_{max}$ )
6   return  $(A_{max}, B_{max})$  (unpermuted)

7 cliqueSort :: (Graph  $G$ , Set  $P$ ) → (Array of Integer, Array of Integer)
8 begin
9   bounds ← an Array of Integer
10  order ← an Array of Integer
11   $P' \leftarrow P$  // vertices yet to be allocated
12   $k \leftarrow 1$  // current clique number
13  while  $P' \neq \emptyset$  do
14     $Q \leftarrow P'$  // vertices to consider for the current clique
15    while  $Q \neq \emptyset$  do
16       $v \leftarrow$  the first element of  $Q$  // get next vertex to allocate
17       $P' \leftarrow P' \setminus \{v\}$ 
18       $Q \leftarrow Q \cap N(G, v)$  // remove non-adjacent vertices
19      append  $k$  to bounds
20      append  $v$  to order
21     $k \leftarrow k + 1$  // start a new clique
22  return (bounds, order)

23 expand :: (Graph  $G$ , Set  $A$ , Set  $B$ , Set  $P_a$ , Set  $P_b$ , Set  $A_{max}$ , Set  $B_{max}$ )
24 begin
25   (bounds, order) ← cliqueSort( $G, P_a$ )
26   for  $i \leftarrow |P_a|$  downto 1 do
27     if bounds[ $i$ ] +  $|A| > |A_{max}|$  and  $|P_b| + |B| > |B_{max}|$  then
28        $v \leftarrow order[i]$ 
29        $A \leftarrow A \cup \{v\}$  // Consider  $v \in A$ 
30        $P_a \leftarrow P_a \setminus \{v\}$ 
31        $P'_a \leftarrow P_a \cap \overline{N_G(v)}$  // Remove vertices adjacent to  $v$ 
32        $P'_b \leftarrow P_b \cap N_G(v)$  // Remove vertices not adjacent to  $v$ 
33       if  $|A| = |B|$  and  $|A| > |A_{max}|$  then
34          $(A_{max}, B_{max}) \leftarrow (A, B)$  // We've found a better solution
35       if  $P'_b \neq \emptyset$  then
36         expand( $G, B, A, P'_b, P'_a, B_{max}, A_{max}$ ) // Swap and recurse
37        $A \leftarrow A \setminus \{v\}$  // Now consider  $v \notin A$ 
38       if  $B = \emptyset$  then
39          $P_b \leftarrow P_b \setminus \{v\}$  // Avoid symmetric solutions

```

---

(or the other way around—after all, the order of  $A$  and  $B$  is arbitrary). We are doing the same thing, by saying that if the first  $n$  bits of  $A$  are 0 then the first  $n$  bits of  $B$  must also be 0. Unlike adding a lex constraint, this approach does not interfere with the search order and does not introduce the risk of disrupting ordering heuristics [6]. Additionally, this constraint always removes symmetric solutions from the search tree as early as possible [7].

*Bounding.* We know that  $A$  and  $B$  must be independent sets. Finding a maximum independent set is a well studied NP-hard problem (although the literature usually discusses finding a maximum clique, which is a maximum independent set in the complement graph), and the main inspiration for our algorithm comes from a series of maximum clique algorithms due to Tomita [8,9,10]. These are branch and bound algorithms which use graph colouring (i.e. a clique cover in the complement graph) both as a bound and an ordering heuristic.

If we can cover a graph  $G$  using  $k$  cliques, we know that  $G$  cannot contain an independent set of size greater than  $k$  (each element in an independent set must be in a different clique). Finding an optimal clique cover is NP-hard, but a greedy clique cover may be found in polynomial time. This gives us a bound on  $P_a$  which can be much better than simply considering  $|P_a|$ : we construct a greedy clique cover of the subgraph induced by  $P_a$ , and consider its size instead.

Constructing a clique cover gives us more information than just a bound on the size of an independent set in all of  $P_a$ . This is the main benefit of Tomita's approach: a constructive greedy clique cover gives us an ordering heuristic and a way of reducing the number of clique covers which must be computed.

Tomita has considered ways of producing and using greedy colourings; we refer to a computational study by Prosser [11] for a detailed comparison. Our greedy clique cover bound and ordering routine is presented in Algorithm 2. The approach we have taken is a variation by San Segundo [2,3] which allows a bitset encoding to be used.

The `cliqueSort` function in Algorithm 2 produces two arrays. The *bounds* array contains bounds on the size of a maximum independent set: the subgraph induced by vertices 1 to  $n$  of *order* cannot have a maximum independent set of size greater than  $bounds[n]$ . The *order* array contains the vertices of  $P$  in some order, and is to be traversed from right to left, repeatedly removing the rightmost value for the choice branching vertex  $v$ .

These arrays are constructed in the `cliqueSort` function as follows: the variable  $P'$  tracks which vertices have yet to be allocated to a clique, and initially (line 11) it contains every vertex in the parameter  $P$ . While there are unallocated vertices (line 13), we greedily construct a new clique. The variable  $Q$  (line 14) tracks which vertices may legally be added to this growing clique. On line 16 we select a vertex  $v$  from  $Q$ , add it to the clique, and on line 18 we remove from  $Q$  any vertices which are not adjacent to  $v$  (so every vertex remaining in  $Q$  is adjacent to every vertex in the growing clique). We continue adding vertices to the growing clique until  $Q$  is empty (line 15), indicating we can go no further. We then start a new clique (line 21, looping back to line 13) if some vertices remain unallocated.

To integrate this bound, we make the following changes: we begin by using `cliqueSort` to obtain the *bounds* and *order* variables (line 25). We explicitly iterate over *order* from right to left (lines 26 and 28), rather than drawing  $v$  from  $P_a$  arbitrarily. And we make use of the bound on  $P_a$ , rather than using  $|P_a|$  (line 27).

*Search Order.* We use a static ordering for constructing clique covers, so the initial order of vertices must also be considered—experiments show that, as for the maximum clique problem, a static non-increasing degree order fixed at the top of search is a good choice. We achieve this ordering by permuting the graph (again, to allow the possibility of a bitset encoding).

*Detecting when the Bound is Useless.* Our bound considers how far  $A$  can grow, based upon what is in  $P_a$ , and how far  $B$  can grow based upon what is in  $P_b$ . If both  $P_a$  and  $P_b$  are independent sets, this does not help, and constructing the clique cover ordering is a substantial overhead. This situation occurs in particular if the input is a bipartite graph, or close to one. We can at least detect when  $P_a$  is an independent set: this happens precisely if  $bounds[i] = i$  (assuming *bounds* is 1-indexed), since if the graph contains at least two non-adjacent vertices then at least one such pair will be placed in the same clique [12, Proposition 2].

Ideally we would be able to switch to a better bound in the case that both  $P_a$  and  $P_b$  are (potentially overlapping) independent sets. However the authors have been unable to find a better bound which is sufficiently cheap to compute to provide a benefit—approaches which reduce the search space but increase runtime include the use of degrees, indirect colouring, or the fact that finding an (unbalanced) induced biclique in a bipartite graph can be done in polynomial time via a matching algorithm. However, we may still decay to a version of the algorithm which includes symmetry breaking and uses cardinality bounds as in Algorithm 1. We do not demonstrate this technique in Algorithm 2, but it is simple to incorporate.

### 3 Computational Experiments

We now present experimental results on a range of standard benchmark problems. The algorithm was implemented using C++, with a bitset encoding. The experiments were run on a machine with four AMD Opteron 6366 HE processors, and single-threaded runtimes are given. The implementation does include detection for independent sets, and falls back to a simple algorithm when this happens. Timing results include pre-processing and the initial sorting step, but do not include the time taken to read a graph in from a file. For the maximum clique problem, a sequential implementation previously described by the authors [13] was used.

In Table 1 we present results from four datasets. First is all the graphs from the Second DIMACS Implementation Challenge<sup>1</sup>. Many of these graphs are dense,

<sup>1</sup> <http://dimacs.rutgers.edu/Challenges/>

and designed to be computationally challenging for maximum clique algorithms. The second dataset is the smallest family of graphs for BHOSLIB<sup>2</sup>. These graphs contain a hidden clique of known size; again, these are challenging for maximum clique algorithms. Thirdly, we look at some large sparse graphs from BioGRID [14]. Finally, we include some large sparse graphs from a collection by Mark Newman<sup>3</sup>. For each instance we show results for both maximum clique and maximum balanced induced biclique: we show the size of the result, the time taken, and the number of search nodes (recursive calls made). Longer-running problems were aborted after one day; such results are shown in parentheses.

Sometimes  $\tilde{\omega} = \omega$ , sometimes it is larger, and sometimes it is smaller. Often finding  $\tilde{\omega}$  was easier than finding  $\omega$  (and there are no problems where the biclique search was aborted after a day but where the clique succeeded), but not always.

Further experiments show that the symmetry breaking technique is successful in reducing both runtimes and the size of the search space. In many instances the gain approaches 50% (this is expected: halving the number of solutions will not halve the size of the search space). In other cases the interaction of the bound and symmetry breaking reduces the benefit (sometimes to zero, when the bound can already eliminate symmetric solutions), but it is never a penalty.

Detecting when the bound is useless and decaying to a simpler algorithm provides a measurable benefit for several of the “p\_hat” family of graphs and for “san1000”, but does not generally make a substantial difference. On the other hand, for random bipartite graphs, this technique avoids a factor of five slowdown from the overhead of calculating a useless bound.

## 4 Conclusion and Future Work

We have shown that max clique techniques generalise to other graph-related problems, although not always in the most obvious way—despite the name, finding a biclique involves finding independent sets, not cliques. Unlike the maximum clique problem, symmetry is an issue, but we provided a very simple and effective way of avoiding this problem. We do not have a good bound for the case where both sides are already independent sets, although we can detect this and fall back to a faster algorithm; this limitation is this work’s main weakness.

More detailed computational experiments would be beneficial, particularly with random and (once the weakness is addressed) random bipartite graphs. We intend to look in more detail at “where the hard problems are” for this problem [15]: there is a conflict between wanting to create two independent sets, and requiring those independent sets be interconnected, which means it is not obvious how the density of a random graph would affect the difficulty.

Finally, this approach can likely be extended to exploit multi-core parallelism—the sequential algorithms upon which this work is based have been threaded successfully [13,16].

<sup>2</sup> <http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm>

<sup>3</sup> <http://www-personal.umich.edu/~mejn/netdata/>

**Table 1.** Results for the balanced biclique problem in DIMACS, BHOSLIB and large sparse graphs from BioGRID and Mark Newman. For each we show the size of a maximum clique, the time taken to obtain this result, and the number of search nodes (recursive calls made). We then give the same information for maximum balanced induced bicliques. Results in parentheses were aborted after one day.

Problem	ω			ω			ω			ω			ω							
	Size	Time	Nodes	Size	Time	Nodes	Size	Time	Nodes	Size	Time	Nodes	Size	Time	Nodes					
C125.9	34	91ms	50240	8	1ms	920	1 day 2.0×10 <sup>10</sup>	(53)	12	35ms	16388	san400-0.7	30	4.0s	8.9×10 <sup>5</sup>	28	33ms	7973		
C250.9	44	3043s	1.1×10 <sup>9</sup>	8	12ms	12448	hamming6-2	32	4	0ms	32	4	0ms	2.3s	5.2×10 <sup>5</sup>	38	38ms	10361		
C500.9	(53)	1 day 2.0×10 <sup>10</sup>	10	174ms	1.1×10 <sup>5</sup>	hamming6-4	4	0ms	82	14	1ms	1806	san400-0.9	100	52.3s	4.5×10 <sup>6</sup>	10	38ms	19054	
C1000.9	(58)	1 day 1.3×10 <sup>10</sup>	10	13.9s	7.4×10 <sup>6</sup>	hamming8-2	128	2ms	128	4	1ms	303	san1000	15	3.5s	1.5×10 <sup>5</sup>	134	167ms	10778	
C2000.9	(16)	1 day 1.4×10 <sup>10</sup>	(16)	1 day 2.9×10 <sup>10</sup>	hamming8-4	16	80ms	36452	32	2ms	303	sanr200-0.7	18	235ms	1.5×10 <sup>5</sup>	10	96ms	1.3×10 <sup>5</sup>		
C2000.9	(62)	1 day 5.5×10 <sup>9</sup>	12	1478s	3.2×10 <sup>8</sup>	hamming10-2	512	56ms	512	4	21ms	4	sanr200-0.9	42	45.2s	1.5×10 <sup>7</sup>	8	4ms	4095	
C4000.5	(17)	1 day 7.7×10 <sup>9</sup>	(18)	1 day 1.4×10 <sup>10</sup>	johnson10-4	(38)	1 day 1.0×10 <sup>10</sup>	40	390s	4.5×10 <sup>7</sup>	sanr400-0.5	13	543ms	3.2×10 <sup>5</sup>	14	1.4s	1.4×10 <sup>7</sup>	8	4ms	4095
DSJC500.5	13	1.8s	1.2×10 <sup>6</sup>	14	63.4s	6.8×10 <sup>7</sup>	johnson8-2-4	4	0ms	24	0ms	460	sanr400-0.7	21	159s	6.4×10 <sup>7</sup>	14	4.3s	3.4×10 <sup>6</sup>	
DSJC1000.5	15	222s	7.7×10 <sup>7</sup>	16	12996s	8.9×10 <sup>9</sup>	johnson8-4-4	14	0ms	126	10	0ms	211	1165s	2.9×10 <sup>8</sup>	30	58ms	15361		
MANN <sub>29</sub>	16	0ms	71	6	0ms	32	johnson16-2-4	(16)	1 day 1.4×10 <sup>11</sup>	(30)	1 day 4.2×10 <sup>11</sup>	frb30-15-1	30	2187s	5.6×10 <sup>8</sup>	30	63ms	17091		
MANN <sub>27</sub>	126	533ms	38019	6	4ms	1407	johnson32-2-4	(16)	1 day 1.7×10 <sup>5</sup>	8	97ms	2.6×10 <sup>5</sup>	frb30-15-2	30	655s	1.7×10 <sup>8</sup>	30	59ms	16120	
MANN <sub>45</sub>	345	383s	2.9×10 <sup>6</sup>	6	56ms	9852	keller4	11	17ms	13725	18	69ms	82646	frb30-15-4	30	3575s	9.9×10 <sup>8</sup>	30	61ms	16694
MANN <sub>81</sub>	(1100)	1 day 8.7×10 <sup>7</sup>	6	974ms	53902	keller5	(27)	1 day 1.8×10 <sup>10</sup>	32	7294s	3.6×10 <sup>9</sup>	frb30-15-4	30	1056s	2.8×10 <sup>8</sup>	30	55ms	14850		
brock200.1	21	868ms	5.2×10 <sup>5</sup>	10	45ms	57931	keller6	(53)	1 day 2.6×10 <sup>9</sup>	(62)	1 day 6.0×10 <sup>9</sup>	frb30-15-5	30	1056s	2.8×10 <sup>8</sup>	30	55ms	14850		
brock200.2	12	5ms	3826	12	111ms	1.7×10 <sup>5</sup>	p-hat300-1	8	4ms	4256	12	195ms	2.8×10 <sup>5</sup>	208	50ms	208	12	110ms	33253	
brock200.3	15	23ms	14565	12	92ms	1.2×10 <sup>5</sup>	p-hat300-2	25	18ms	4456	12	268ms	2.8×10 <sup>5</sup>	frb30-yeast	12	50ms	208	12	110ms	33253
brock200.4	17	85ms	58730	12	76ms	1.0×10 <sup>5</sup>	p-hat500-1	36	2.0s	6.2×10 <sup>5</sup>	12	265ms	2.3×10 <sup>5</sup>	fruitfly	7	518ms	47	16	584ms	11538
brock400.1	27	508s	2.0×10 <sup>8</sup>	12	2.3s	1.8×10 <sup>6</sup>	p-hat500-3	9	18ms	9777	12	3.3s	3.9×10 <sup>6</sup>	human	13	897ms	13	18	1.0s	10300
brock400.2	29	362s	1.5×10 <sup>8</sup>	12	2.0s	1.8×10 <sup>6</sup>	p-hat500-2	36	461ms	1.1×10 <sup>5</sup>	14	6.4s	5.9×10 <sup>6</sup>	mouse	7	21ms	7	10	22ms	1267
brock400.3	31	287s	1.5×10 <sup>8</sup>	12	2.1s	1.8×10 <sup>6</sup>	p-hat500-3	50	201s	3.9×10 <sup>7</sup>	12	9.0s	6.4×10 <sup>6</sup>	plant	9	29ms	9	10	31ms	1578
brock400.4	33	140s	5.4×10 <sup>7</sup>	12	2.0s	1.8×10 <sup>6</sup>	p-hat700-1	11	65ms	26649	12	36.8s	4.3×10 <sup>7</sup>	worm	7	122ms	7	12	130ms	3778
brock800.1	23	7725s	2.2×10 <sup>9</sup>	14	1424s	9.5×10 <sup>8</sup>	p-hat700-2	44	5.0s	7.5×10 <sup>7</sup>	14	56.3s	3.5×10 <sup>7</sup>	yeast	33	375ms	68	14	13.4s	2.5×10 <sup>5</sup>
brock800.2	24	7715s	2.2×10 <sup>9</sup>	14	1367s	9.1×10 <sup>8</sup>	p-hat700-3	62	2666s	2.8×10 <sup>5</sup>	14	67.9s	3.1×10 <sup>7</sup>	astro	5	0ms	17	6	0ms	207
brock800.3	25	7138s	2.1×10 <sup>9</sup>	14	1448s	9.6×10 <sup>8</sup>	p-hat1000-1	10	454ms	1.8×10 <sup>5</sup>	14	295s	2.5×10 <sup>8</sup>	adipon	5	2.7s	57	6	3.5s	28143
brock800.4	26	2705s	6.4×10 <sup>8</sup>	14	1401s	9.4×10 <sup>8</sup>	p-hat1000-2	46	251s	3.4×10 <sup>7</sup>	16	546s	3.6×10 <sup>8</sup>	celegans	8	1ms	32	8	2ms	1853
c-fat200-1	12	0ms	24	2	0ms	214	p-hat1000-3	(63)	1 day 8.9×10 <sup>9</sup>	14	1300s	5.6×10 <sup>8</sup>	condmat	30	17.2s	30	6	20.2s	63980	
c-fat200-2	24	0ms	24	2	1ms	353	p-hat1500-1	12	6.9s	1.2×10 <sup>6</sup>	16	11859s	5.2×10 <sup>9</sup>	dolphins	5	0ms	10	4	0ms	66
c-fat200-5	58	1ms	139	2	3ms	927	p-hat1500-2	65	43166s	2.0×10 <sup>9</sup>	16	23677s	6.8×10 <sup>9</sup>	football	9	0ms	9	4	0ms	422
c-fat500-1	14	3ms	14	2	3ms	523	san1500-3	(79)	1 day 3.2×10 <sup>9</sup>	16	25745s	5.5×10 <sup>9</sup>	international	17	5.1s	50	10	5.5s	24477	
c-fat500-2	26	3ms	26	2	4ms	619	san200-0.7	10	31ms	13399	14	6ms	4330	karate	5	0ms	5	4	0ms	31
c-fat500-5	64	4ms	64	2	7ms	1398	san200-0.7-2	18	3ms	464	24	3ms	1939	lesmis	10	0ms	10	4	0ms	77
c-fat500-10	126	4ms	126	2	41ms	4219	san200-0.9-1	70	206ms	87329	8	3ms	1850	polblogs	20	24ms	20	4	26ms	1184
gen200-p0.9-44	44	4.8s	1.8×10 <sup>6</sup>	10	3ms	2628	san200-0.9-2	60	7699ms	2.3×10 <sup>5</sup>	8	4ms	2085	polbooks	20	23ms	60	12	83ms	36693
gen200-p0.9-55	55	461ms	1.7×10 <sup>5</sup>	8	4ms	4201	san200-0.9-3	44	19.3s	6.8×10 <sup>6</sup>	10	3ms	1315	polbooks	6	0ms	11	4	0ms	168
gen400-p0.9-255	(50)	1 day 2.2×10 <sup>10</sup>	16	21ms	8562	san400-0.5-1	13	15ms	2453	62	9ms	1315	polbooks	6	0ms	11	4	0ms	168	
gen400-p0.9-65	(49)	1 day 2.3×10 <sup>10</sup>	14	26ms	11709	san400-0.7-1	40	459ms	1.2×10 <sup>5</sup>	20	54ms	16229	power	6	235ms	6	4	252ms	4623	



## References

1. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York (1979)
2. San Segundo, P., Rodríguez-Losada, D., Jiménez, A.: An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.* 38(2), 571–581 (2011)
3. San Segundo, P., Matia, F., Rodríguez-Losada, D., Hernando, M.: An improved bit parallel exact maximum clique algorithm. *Optimization Letters* (2011)
4. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. In: *KR 1996: Principles of Knowledge Representation and Reasoning*, pp. 148–159. Morgan Kaufmann (1996)
5. Gent, I.P., Petrie, K.E., François Puget, J.: Symmetry in constraint programming. In: *Handbook of Constraint Programming*, pp. 329–376. Elsevier (2006)
6. Gent, I.P., Harvey, W., Kelsey, T.: Groups and constraints: Symmetry breaking during search. In: Van Hentenryck, P. (ed.) *CP 2002*. LNCS, vol. 2470, pp. 415–430. Springer, Heidelberg (2002)
7. Backofen, R., Will, S.: Excluding symmetries in constraint-based search. *Constraints* 7(3-4), 333–349 (2002)
8. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) *DMTCS 2003*. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003), [http://dx.doi.org/10.1007/3-540-45066-1\\_22](http://dx.doi.org/10.1007/3-540-45066-1_22)
9. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization* 37(1), 95–111 (2007)
10. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Rahman, M. S., Fujita, S. (eds.) *WALCOM 2010*. LNCS, vol. 5942, pp. 191–203. Springer, Heidelberg (2010)
11. Prosser, P.: Exact Algorithms for Maximum Clique: A Computational Study. *Algorithms* 5(4), 545–587 (2012)
12. Batsyn, M., Goldengorin, B., Maslov, E., Pardalos, P.M.: Improvements to MCS algorithm for the maximum clique problem. *Journal of Combinatorial Optimization*, 1–20 (2013)
13. McCreesh, C., Prosser, P.: Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms* 6(4), 618–635 (2013)
14. Stark, C., Breitkreutz, B.J., Regulý, T., Boucher, L., Breitkreutz, A., Tyers, M.: Biogrid: A general repository for interaction datasets. *Nucleic Acids Research* 34(suppl. 1), D535–D539 (2006)
15. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI 1991*, vol. 1, pp. 331–337. Morgan Kaufmann Publishers Inc., San Francisco (1991)
16. Depolli, M., Konc, J., Rozman, K., Trobec, R., Janežič, D.: Exact parallel maximum clique algorithm for general and protein graphs. *Journal of Chemical Information and Modeling* 53(9), 2217–2228 (2013)